

WA2402 Introduction to Responsive Web Development with HTML5, CSS3, JavaScript and jQuery



Web Age Solutions Inc.
USA: 1-877-517-6540
Canada: 1-866-206-4644
Web: <http://www.webagesolutions.com>

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

IBM, WebSphere, DB2 and Tivoli are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

For customizations of this book or other sales inquiries, please contact us at:

USA: 1-877-517-6540, email: getinfousa@webagesolutions.com

Canada: 1-866-206-4644 toll free, email: getinfo@webagesolutions.com

Copyright © 2018 Web Age Solutions Inc.

This publication is protected by the copyright laws of Canada, United States and any other country where this book is sold. Unauthorized use of this material, including but not limited to, reproduction of the whole or part of the content, re-sale or transmission through fax, photocopy or e-mail is prohibited. To obtain authorization for any such activities, please write to:

Web Age Solutions Inc.
439 University Ave
Suite 820
Toronto
Ontario, M5G 1Y8

Table of Contents

Chapter 1 - HTML5 Overview.....	15
1.1 What Is HTML5.....	15
1.2 HTML5 Goals.....	15
1.3 HTML Specs, Past and Present.....	16
1.4 How Is HTML5 Different From HTML4?.....	16
1.5 HTML5 Is Not Based On SGML.....	18
1.6 More Differences.....	19
1.7 HTML5 Defines Required Processing For Invalid Documents.....	19
1.8 The Doctype Declaration.....	19
1.9 Current Browser Support for HTML5.....	20
1.10 Detecting Support for HTML5.....	20
1.11 Detecting Support for HTML5.....	21
1.12 Detecting Support for HTML5.....	21
1.13 New Features of HTML5.....	22
1.14 New Features of HTML5.....	23
1.15 New Features of HTML5.....	23
1.16 The Function over Form Philosophy.....	24
1.17 Semantic Elements.....	24
1.18 HTML4 Layouts.....	25
1.19 HTML5 Semantic Layouts.....	26
1.20 Nesting Semantics.....	27
1.21 Replacing Flash with HTML5.....	27
1.22 Summary.....	28
Chapter 2 - Forms.....	29
2.1 The form Attribute.....	29
2.2 The placeholder Attribute.....	29
2.3 New Form Field Types.....	30
2.4 New Form Field Types.....	31
2.5 Forms and Validation.....	31
2.6 The required Attribute.....	31
2.7 The number input type.....	32
2.8 The pattern Attribute.....	32
2.9 The range and date input types.....	32
2.10 The <datalist> Element.....	33
2.11 The autofocus and oninput Attributes.....	34
2.12 The autofocus and oninput Attributes.....	34
2.13 HTML5 CSS Pseudo-Classes.....	35
2.14 Summary.....	35
Chapter 3 - Canvas.....	37
3.1 The <canvas> Element.....	37
3.2 <canvas> vs. <svg>.....	38
3.3 <canvas> vs. <svg>.....	38
3.4 Browser Support for <canvas>.....	39
3.5 Creating the Canvas.....	39

3.6 Using the Context.....	40
3.7 Using the Context.....	40
3.8 Using Color.....	41
3.9 Painting Gradients.....	41
3.10 Drawing Paths.....	42
3.11 Drawing Paths.....	42
3.12 Painting Patterns.....	43
3.13 Transformers.....	43
3.14 Summary.....	44
Chapter 4 - Video and Audio.....	45
4.1 HTML5 Video/Audio Overview.....	45
4.2 New Elements for Video/Audio.....	45
4.3 Using the <audio> Element.....	46
4.4 The <video> Element.....	46
4.5 Specifying More Than One Audio or Video File.....	47
4.6 The poster Attribute.....	47
4.7 Other <audio> and <video> Attributes.....	48
4.8 JavaScript and Media Elements.....	48
4.9 Summary.....	48
Chapter 5 - Introduction to CSS3.....	51
5.1 What is a Style?.....	51
5.2 What are Cascading Style Sheets?.....	51
5.3 CSS and the Evolution of Web Development.....	52
5.4 The CSS Standardization Process.....	53
5.5 CSS and HTML.....	53
5.6 CSS Compatibility.....	54
5.7 CSS Rules.....	55
5.8 New in CSS3.....	55
5.9 Summary.....	56
Chapter 6 - Applying CSS Styles.....	57
6.1 Inline Styles.....	57
6.2 Embedded Styles.....	57
6.3 External Styles.....	58
6.4 Selectors.....	59
6.5 Combinator Selectors.....	60
6.6 Combinator Selectors.....	60
6.7 Universal Selector.....	61
6.8 Style Classes.....	62
6.9 Style Classes.....	62
6.10 Pseudo-Classes.....	62
6.11 Inheriting From a Parent.....	63
6.12 Declaring !important Styles.....	64
6.13 CSS Cascade Order.....	64
6.14 Summary.....	65
Chapter 7 - Styling Text.....	67
7.1 Web Typography.....	67

7.2 Generic Font Families.....	67
7.3 Font-Stack and Understudy Fonts.....	68
7.4 Web Fonts.....	68
7.5 Using Web Fonts.....	69
7.6 Font Size.....	70
7.7 Font Weight.....	71
7.8 Italics and Underlining.....	72
7.9 Capitalization.....	73
7.10 Line Height.....	73
7.11 Multiple Font Values.....	73
7.12 Text Spacing.....	74
7.13 Aligning Text.....	74
7.14 Summary.....	75
Chapter 8 - Box Model and Effects.....	77
8.1 Element Box Model.....	77
8.2 Parts of the Box Model.....	78
8.3 Setting Width and Height.....	78
8.4 IE Box Size Bug.....	79
8.5 Controlling Flow in Position.....	79
8.6 Hiding Content.....	80
8.7 Overflowing Content.....	81
8.8 Floating Elements.....	81
8.9 Using Float for Multiple Columns.....	82
8.10 Margins.....	83
8.11 Padding.....	84
8.12 Border.....	84
8.13 Outline.....	85
8.14 CSS 3 - Rounding Border Corners.....	86
8.15 CSS 3 - Using a Border Image.....	87
8.16 Border Image Example.....	88
8.17 Summary.....	89
Chapter 9 - Introduction to JavaScript.....	91
9.1 What JavaScript Is.....	91
9.2 What JavaScript Is.....	92
9.3 What JavaScript Is Not.....	92
9.4 Not All JavaScripts are Created Equal	93
9.5 ECMAScript Language and Specification.....	93
9.6 What JavaScript Can Do.....	94
9.7 What JavaScript Can't Do.....	94
9.8 JavaScript on the Server-side.....	95
9.9 Elements of JavaScript.....	95
9.10 Values, Variables and Functions.....	96
9.11 Embedded Scripts.....	97
9.12 External Scripts.....	97
9.13 Browser Dialog Boxes.....	98
9.14 What is AJAX?.....	99

9.15 Summary.....	99
Chapter 10 - JavaScript Fundamentals.....	101
10.1 Variables.....	101
10.2 JavaScript Reserved Words	102
10.3 Dynamic Types.....	102
10.4 JavaScript Strings.....	103
10.5 Escaping Control Characters.....	103
10.6 What is False in JavaScript?.....	104
10.7 Numbers.....	104
10.8 The Number Object	105
10.9 Not A Number (NaN) Reserved Keyword.....	105
10.10 JavaScript Objects.....	105
10.11 Operators.....	106
10.12 Primitive Values vs Objects.....	107
10.13 Primitive Values vs Objects.....	107
10.14 Flow Control.....	108
10.15 'if' Statement.....	108
10.16 'if...else' Statement.....	109
10.17 'switch' Statement.....	110
10.18 'for' Loop.....	111
10.19 'for / in' Loop.....	112
10.20 'while' Loop.....	113
10.21 'do...while' Loop.....	114
10.22 Break and Continue.....	114
10.23 Labeled Statements.....	115
10.24 The undefined and null Keywords.....	116
10.25 Checking for undefined and null.....	117
10.26 Checking Types with typeof Operator	117
10.27 Date Object.....	117
10.28 Document Object.....	118
10.29 Other Useful Objects.....	119
10.30 Browser Object Detection.....	119
10.31 The eval Function	120
10.32 Enforcing Strict Mode	120
10.33 Summary.....	121
Chapter 11 - JavaScript Functions.....	123
11.1 Functions Defined.....	123
11.2 Declaring Functions.....	123
11.3 Function Arguments.....	124
11.4 More on Function Arguments.....	124
11.5 Return Values.....	125
11.6 Multiple Return Values in ECMAScript 6	125
11.7 Optional Default Parameter Values.....	126
11.8 Emulating Optional Default Parameter Values.....	126
11.9 Anonymous Function Expressions.....	126
11.10 Functions as a Way to Create Private Scope.....	127

11.11 Linking Functions to Page Elements.....	127
11.12 Local and Global Variables.....	128
11.13 Local and Global Variables.....	129
11.14 Declaring Object Methods.....	129
11.15 The arguments Parameter	130
11.16 Example of Using arguments Parameter.....	130
11.17 Summary.....	131
Chapter 12 - JavaScript Arrays.....	133
12.1 Arrays Defined.....	133
12.2 Creating an Array.....	133
12.3 The length Array Member.....	134
12.4 Traversing an Array.....	135
12.5 Appending to an Array.....	135
12.6 Deleting Elements.....	136
12.7 Inserting Elements.....	136
12.8 Other Array Methods.....	137
12.9 Other Array Methods.....	137
12.10 Accessing Objects as Arrays.....	138
12.11 Summary.....	138
Chapter 13 - Advanced Objects and Functionality in JavaScript.....	141
13.1 Basic Objects.....	141
13.2 Constructor Function.....	142
13.3 More on the Constructor Function.....	142
13.4 Object Properties.....	143
13.5 Deleting a Property.....	143
13.6 Object Properties.....	144
13.7 Constructor and Instance Objects.....	144
13.8 Constructor Level Properties.....	145
13.9 Namespace.....	146
13.10 Functions are First-Class Objects.....	147
13.11 Closures.....	148
13.12 Closure Examples.....	149
13.13 Closure Examples.....	149
13.14 Closure Examples.....	150
13.15 Private Variables with Closures.....	150
13.16 Immediately Invoked Function Expression (IIFE).....	151
13.17 Prototype.....	152
13.18 Inheritance in JavaScript.....	152
13.19 The Prototype Chain	153
13.20 Traversing Prototype Property Hierarchy.....	154
13.21 Prototype Chain.....	155
13.22 Summary.....	156
Chapter 14 - jQuery Overview	157
14.1 What Is jQuery?.....	157
14.2 Benefits of Using a JavaScript Library.....	157
14.3 jQuery Example.....	158

14.4 CSS Selectors.....	158
14.5 How to Use jQuery.....	158
14.6 Practical Usage Notes.....	159
14.7 Background – DOM.....	160
14.8 Background - DOM Ready Events.....	160
14.9 Background - JavaScript Functions.....	160
14.10 The jQuery Function Object.....	161
14.11 What Does the \$() Function Take as Argument?.....	161
14.12 What Does the \$() Function do?.....	162
14.13 The jQuery Wrapper.....	162
14.14 The jQuery Wrapper as an Array-Like Object.....	163
14.15 Note: innerHTML() vs. .html().....	163
14.16 jQuery Wrapper Chaining.....	163
14.17 API Function Notation.....	164
14.18 Handling DOM Ready Event.....	164
14.19 xhtml Note.....	164
14.20 References.....	165
14.21 Summary.....	165
Chapter 15 - Selectors.....	167
15.1 Background: The Sizzle Selector Engine.....	167
15.2 Selecting Elements by Attribute	167
15.3 Pseudo-Selectors.....	168
15.4 Form Pseudo-Selectors.....	168
15.5 Form Pseudo-Selectors.....	168
15.6 Faster Selection.....	169
15.7 Selecting Elements Using Relationships.....	169
15.8 Selecting Elements Using Filters.....	170
15.9 More on Chaining: .end().....	170
15.10 Testing Elements.....	170
15.11 Is the Selection Empty?.....	170
15.12 Saving Selections.....	171
15.13 Iterating Through Selected Elements Using .each().....	171
15.14 JavaScript Methods.....	171
15.15 JavaScript "this".....	172
15.16 Function Context.....	172
15.17 Function Context.....	172
15.18 The Function call() Method.....	173
15.19 .each() Revisited.....	173
15.20 Summary.....	173
Chapter 16 - Style Class Manipulation.....	175
16.1 Two Options.....	175
16.2 Specifying Style Properties.....	175
16.3 Setting Style Properties.....	176
16.4 .addClass() / .removeClass().....	176
16.5 Defining a Stylesheet.....	176
16.6 Setting & Getting Dimensions.....	176

16.7 Attributes.....	177
16.8 Summary.....	177
Chapter 17 - DOM Manipulation.....	179
17.1 The \$ Function Revisited.....	179
17.2 The \$ Function Revisited.....	179
17.3 Getters and Setters.....	180
17.4 The text() Element Method.....	180
17.5 Appending DOM Elements.....	180
17.6 Removing DOM Elements.....	181
17.7 Performance.....	181
17.8 Performance.....	181
17.9 Summary.....	182
Chapter 18 - Events.....	183
18.1 Event Overview.....	183
18.2 Old School: Event Handling Using HTML Element Attributes.....	184
18.3 Unobtrusive JavaScript.....	184
18.4 Unobtrusive JavaScript Example.....	185
18.5 Multiple Handlers.....	185
18.6 Using jQuery Wrapper Event Registration Methods.....	185
18.7 The .on() Method.....	186
18.8 Event Propagation.....	186
18.9 Handlers for Elements Before They Exist!.....	187
18.10 The Event Object.....	187
18.11 Triggering Events.....	187
18.12 Summary.....	188
Chapter 19 - Utility Functions.....	189
19.1 The jQuery Object Revisited.....	189
19.2 Functions May Have Methods.....	189
19.3 A jQuery Utility Function: \$.trim().....	190
19.4 \$.each().....	190
19.5 Example jQuery Utility Functions.....	190
19.6 Example jQuery Utility Functions.....	191
19.7 Example jQuery Utility Functions.....	191
19.8 Summary.....	192
Chapter 20 - Ajax	193
20.1 Ajax Overview.....	193
20.2 The Browser & the Server.....	193
20.3 The Ajax Request.....	194
20.4 The Ajax Response.....	194
20.5 Sending an Ajax Request With jQuery - The General Case.....	195
20.6 When this code is executed.....	195
20.7 Sending an Ajax Request With jQuery - Simpler, Typical Case.....	195
20.8 Data Types.....	196
20.9 The .data() method.....	196
20.10 Summary.....	196
Chapter 21 - Advanced Ajax.....	197

21.1 A Form Example.....	197
21.2 An Ajax Form Example.....	197
21.3 Serialize().....	198
21.4 Get vs. Post.....	198
21.5 More on Query Strings.....	198
21.6 ajaxStart() and ajaxError().....	199
21.7 Summary.....	199
Chapter 22 - Parsing JSON.....	201
22.1 JSON.....	201
22.2 Reading JSON from the Server Using Ajax.....	201
22.3 Example file contents.....	202
22.4 Using the Results.....	202
22.5 Optimized Version.....	202
22.6 Getting More From the Response.....	203
22.7 jqXHR Methods.....	203
22.8 POST vs. GET.....	203
22.9 Invalid JSON.....	204
22.10 Using \$.ajaxSetup().....	204
22.11 Summary.....	205
Chapter 23 - Animations and Effects with jQuery and jQuery UI.....	207
23.1 What is jQuery UI?.....	207
23.2 Can I do Animations and Effects using jQuery only?.....	207
23.3 Hiding Elements with jQuery	208
23.4 Using .hide() and .show() in jQuery	208
23.5 Alternating an Element's Visibility in jQuery	208
23.6 Adjusting the Speed in jQuery	209
23.7 Providing a Handler in jQuery	209
23.8 Using .slideUp() / .slideDown() methods in jQuery	209
23.9 jQuery UI Categories	210
23.10 jQuery UI Interactions: Droppable and Draggable.....	210
23.11 jQuery UI Interactions: Droppable and Draggable.....	211
23.12 Droppable and Draggable More Complete Example	211
23.13 jQuery UI Widgets: Datepicker	212
23.14 jQuery UI Widgets: Autocomplete	212
23.15 Summary.....	213
Chapter 24 - Plugins.....	215
24.1 What is a Plugin?.....	215
24.2 Goal.....	215
24.3 Self-Executing Anonymous Functions.....	216
24.4 Meeting Our Goal.....	216
24.5 Prototype Objects	216
24.6 The jQuery Wrapper Class Revisited.....	217
24.7 Example Plugin.....	217
24.8 Summary.....	218
Chapter 25 - Introduction to Responsive Web Design.....	219
25.1 What is Responsive Web Design?.....	219

25.2 Mobile Browsers Quirks.....	219
25.3 Other Mobile Web Considerations.....	220
25.4 Primary Responsive Design Techniques.....	220
25.5 Elements of Responsive Design.....	220
25.6 Example of Responsive Design.....	221
25.7 Responsive Page Design Schematic.....	222
25.8 Alternatives to Responsive Design.....	222
25.9 Summary.....	223
Chapter 26 - CSS 3 and Responsive Web Design.....	225
26.1 Progressive Enhancement.....	225
26.2 Implementing Progressive Enhancement.....	225
26.3 Media Types.....	226
26.4 CSS Style "Reset".....	227
26.5 Conditional Styles for Internet Explorer.....	227
26.6 What is the Viewport?.....	228
26.7 Adapting the Viewport.....	229
26.8 Specifying the Viewport.....	230
26.9 Media Queries.....	231
26.10 Media Features Used in Media Queries.....	232
26.11 Combining Responsive Design Techniques.....	233
26.12 Testing Responsive Design.....	234
26.13 Summary.....	235
Chapter 27 - Responsive Web Page Layout	237
27.1 The Main Layout Types	237
27.2 Responsive Layouts	237
27.3 Popular Layout Patterns.....	238
27.4 The 'Mostly Fluid' Layout Pattern	238
27.5 The 'Column Drop' Layout Pattern	239
27.6 The 'Layout Shifter' Pattern	239
27.7 Other Layout Techniques	240
27.8 Getting Content Fillers.....	240
27.9 The Float CSS Property.....	241
27.10 Combining CSS Styles.....	242
27.11 The Simple Fluid Layout Example.....	242
27.12 The Simple Fluid Layout Technique	242
27.13 The Results.....	243
27.14 Font Size Units.....	243
27.15 Pixel-Sized vs Em-Sized	243
27.16 Font Size Unit Relationships.....	244
27.17 Pixels to Em Conversion Formula	245
27.18 Other Considerations.....	245
27.19 Looking into the Future	246
27.20 Summary.....	246
Chapter 28 - Responsive Images.....	247
28.1 Responsive Images.....	247
28.2 Performance Considerations.....	247

28.3 Shrinking Images.....	247
28.4 Traditional Image Handling Techniques	248
28.5 Media Queries Don't Always Help With Performance.....	248
28.6 A "Fluid" Pixel.....	249
28.7 The Device Viewports.....	249
28.8 CSS Pixels.....	249
28.9 The Power of Simplicity.....	250
28.10 Using Google App Engine to Resize Images	250
28.11 The Picture Element	251
28.12 The srcset Attribute	251
28.13 More on the srcset Attribute	252
28.14 Examples of the srcset Attribute	252
28.15 What is Picturefill?.....	253
28.16 Using Picturefill	253
28.17 Summary.....	254
Chapter 29 - Bootstrap Overview	255
29.1 What Is Bootstrap	255
29.2 Keywords from package.json.....	255
29.3 Bootstrap History.....	256
29.4 Responsive Web Development.....	256
29.5 Responsive Grid Layout.....	256
29.6 Reusable GUI Components.....	257
29.7 JavaScript.....	257
29.8 The Mobile First Philosophy.....	258
29.9 Why RWD Matters.....	258
29.10 Responsive Page Views	258
29.11 SASS.....	259
29.12 Getting Bootstrap.....	259
29.13 Bootstrap Content Delivery Network.....	260
29.14 Other Setup Options.....	261
29.15 The Bootstrap Core Files.....	261
29.16 To Min or Not to Min	262
29.17 Summary.....	262
Chapter 30 - Using Bootstrap.....	263
30.1 Including Bootstrap CSS Files.....	263
30.2 Including Bootstrap JavaScript Files.....	263
30.3 Viewport Meta Tags.....	264
30.4 Example.....	265
30.5 Layouts.....	265
30.6 Grid.....	266
30.7 Grid Source.....	267
30.8 Grid Explained.....	267
30.9 Navigation.....	268
30.10 Navigation (Desktop).....	268
30.11 Navigation (Mobile).....	269
30.12 Navigation Source.....	270

30.13 Navigation Explained.....	270
30.14 Navigation Elements and Styles.....	270
30.15 Application Icons.....	271
30.16 Summary.....	271
Chapter 31 - Bootstrap Miscellaneous Topics.....	273
31.1 Bootstrap Components.....	273
31.2 Bootstrap Components Web Page.....	273
31.3 Integrating Bootstrap Components with jQuery.....	274
31.4 Identifying the Required Version of jQuery.....	274
31.5 Customizing Bootstrap.....	275
31.6 Customizing Sass variables.....	275
31.7 More Customization.....	276
31.8 Customizing Bootstrap Components.....	276
31.9 Light Customizations Steps	277
31.10 Summary.....	277

Chapter 1 - HTML5 Overview

Objectives

Key objectives of this chapter

- Explain What HTML5 Is
- State the HTML5 Goals
- List Some HTML Specifications
- State Differences Between HTML5 and HTML 4
- Explain How HTML5 is Not Based on SGML
- Explain the Importance of HTML5 Required Processing for Invalid Markup
- Use the DOCTYPE Declaration
- List HTML5 Semantic Elements
- Describe Current Browser Support for HTML5

1.1 What Is HTML5

- A single markup language that can be written in either HTML or XHTML syntax
- An attempt to unify
 - ◇ XHTML 1.1
 - ◇ HTML 4.01
 - ◇ DOM2HTML, in particular, JavaScript
- Implemented by browsers and other "user agents"

1.2 HTML5 Goals

- Unify the standards for HTML, XHTML, JavaScript
- Support multimedia in a standard way
 - ◇ Without requiring proprietary plugins/APIs

- Support graphical content in a standard way
 - ◇ Again, without requiring proprietary plugins/APIs
- Encourage more interoperable/consistent implementations by providing detailed processing models
- Maintain the human readability HTML already has
- Deal with the many errors in existing web documents
- Add APIs for "complex web applications"

1.3 HTML Specs, Past and Present

- HTML5 History
 - ◇ HTML 1990
 - ◇ HTML4 standardized HTML in 1997
- HTML5 immediate predecessors
 - ◇ HTML 4.01
 - Published as a W3C recommendation in Dec 1999
 - ◇ XHTML 1.1
 - The W3C's HTML working group was focused here in the 2000's
- HTML5 defined in two documents
 - ◇ WHAT WG
 - ◇ HTML WG (W3C)

1.4 How Is HTML5 Different From HTML4?

- No longer based on SGML
- New attributes added
- Some elements and attributes changed or removed
- , <center> and other style-focused tags have been deprecated
- Frames and Framesets no longer permitted

- Tables cannot be formatted
- Styling is now done through CSS3
- A variety of changes (client-side storage, baked-in validation, socket-based networking, offline processing, etc.), all aimed at enabling robust web-based applications that approximate a desktop computing experience
- New tags for semantic structuring of content and form data: <header>, <footer>, <article>, <section>, <nav>, <aside>, <figure>, etc.
- Enhanced form management that has a broader range of datatype recognition

How Is HTML5 Different From HTML4?

A complete list of new elements:

- 1 article
- 2 aside
- 3 audio
- 4 bdo
- 5 canvas
- 6 command
- 7 datalist
- 8 details
- 9 embed
- 10 figcaption
- 11 figure
- 12 footer
- 13 header
- 14 hgroup
- 15 keygen
- 16 mark
- 17 meter
- 18 nav
- 19 output
- 20 progress
- 21 rp
- 22 rt
- 23 ruby
- 24 section
- 25 source

26 summary

27 time

28 video

29 wbr

1.5 HTML5 Is Not Based On SGML

- HTML5 is not case-sensitive
 - ◇ Example: <HTML>, <html> and <hTmL> are all the same start tag
 - ◇ The HTML5 convention is to use lowercase
 - ◇ In XHTML element names are lowercase
 - *Lowercase is easier to read*
 - *Many HTML editors insert elements as lowercase by default*
- Quotation marks are not required around attribute values
- If you omit the quotation marks, attribute values cannot contain any unescaped
 - ◇ Double and single quote mark
 - ◇ Space
 - ◇ Equal sign
 - ◇ Greater-than

HTML5 Is Not Based On SGML

Example Correct HTML 5 attribute values

```
<p class=onetwo>
<p class="one two">
<p class=one&#61;two>
```

Example Incorrect HTML 5 attribute values

```
<p class=one two>
    User agent would interpret two as an empty attribute
<p class=one=two>
```

1.6 More Differences

- The DOCTYPE Declaration

```
<!DOCTYPE html>
<html>
    ...
</html>
```

- Don't need to self-close empty elements

- Attribute minimization

```
controls
```

- rather than

```
controls="controls"
```

- <html> <head> and <body> are optional

1.7 HTML5 Defines Required Processing For Invalid Documents

- How deal with invalid markup

- In the past, different browsers converted invalid markup to DOM differently

 - ◇ This is a problem because

 - CSS depends on the DOM

 - Also JavaScript depends on the DOM

- Thus invalid pages could end up looking very different on different browsers

- The error correction that has existed since the beginning of the browser era will be made uniform across browser vendors. This will promote consistency and predictability in web design and content rendering.

1.8 The Doctype Declaration

- The Doctype Declaration

```
<!DOCTYPE html>
```

- ◇ On first line
- ◇ Case-insensitive
- ◇ Triggers "standards mode" in all modern browsers, meaning that the browser will render the page according to the HTML5 and CSS standards
- Without it the browser may operate in "Quirks mode", meaning it will try to maintain backward compatibility with legacy pages created for older browser versions.
- This is not an XML DOCTYPE declaration; HTML5 is not XML!

The Doctype Declaration

You can validate your pages at <http://html5.validator.nu>

1.9 Current Browser Support for HTML5

- The level of support for HTML5 varies based on the browser and is continually evolving
- As of July 2012, Safari 5, Chrome 20, Firefox 13, and Opera 12 provide good support for HTML5
- IE9 provides support for HTML5, but does not run on Windows XP
 - ◇ IE8 provides limited support for HTML5 in the area of web storage
 - ◇ However, many users still use IE7, which provides no support for HTML5

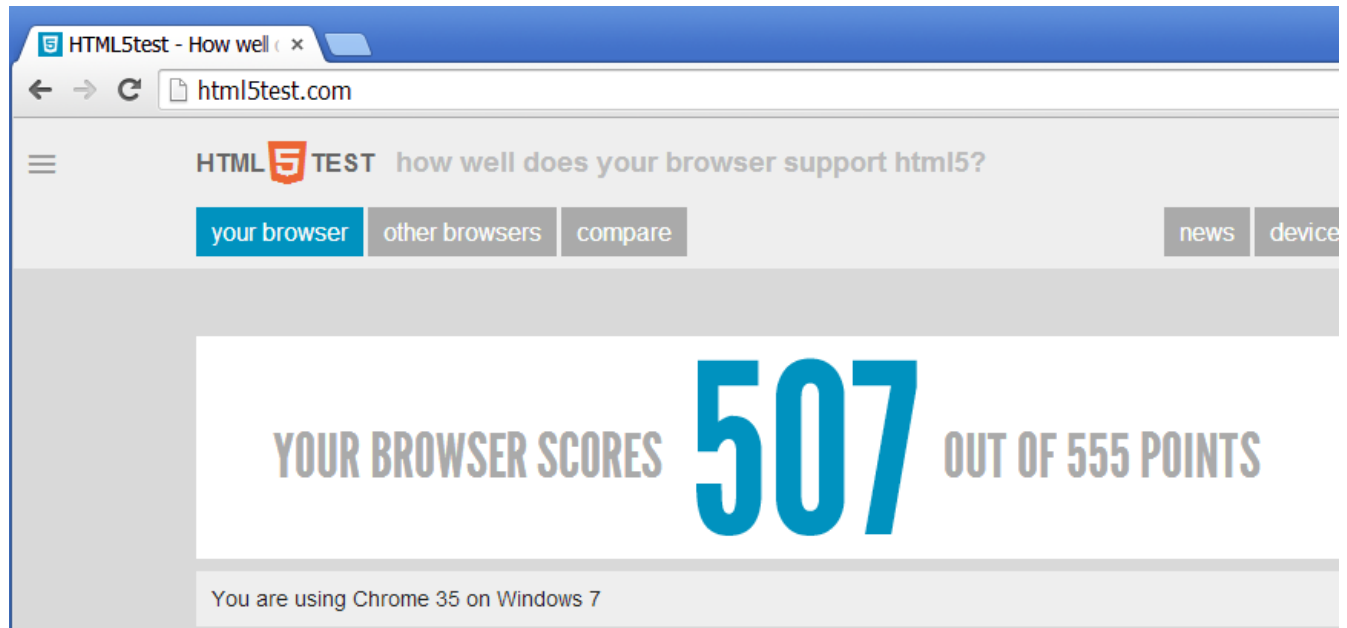
1.10 Detecting Support for HTML5

- Test your browser:
 - ◇ html5test.com
 - ◇ html5demos.com
 - ◇ www.findmebyip.com/litmus
- JavaScript library to test for HTML5 features:

◇ www.modernizr.com

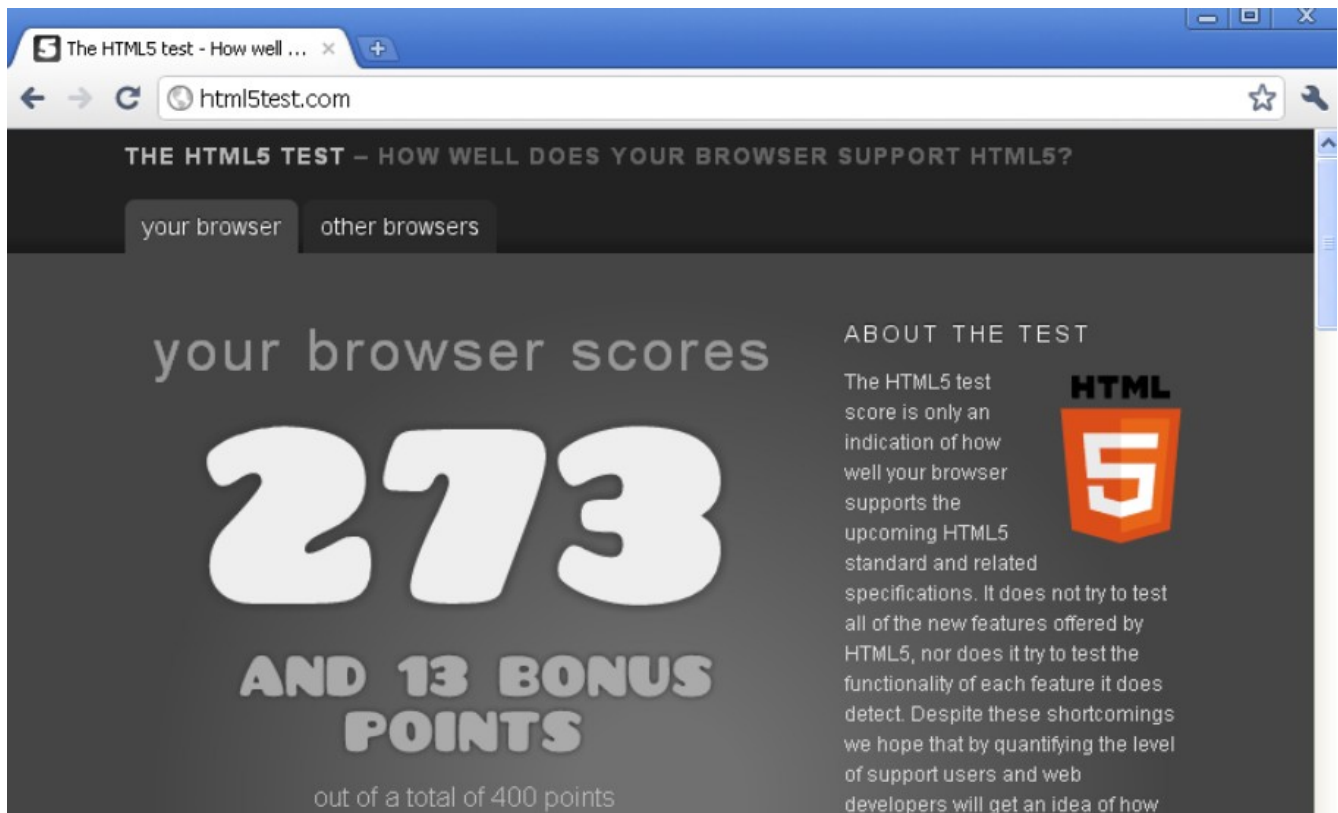
1.11 Detecting Support for HTML5

- Example of running **html5test.com** in Chrome 35 (Summer 2014):
 - ◇ 507 out of 555 points (91% HTML5 feature coverage)



1.12 Detecting Support for HTML5

- Example of running **html5test.com** in Chrome 10 (Spring 2011)
 - ◇ 273 out of 400 points (68% HTML5 feature coverage)



1.13 New Features of HTML5

- Canvas
 - ◇ Currently supports 2D graphics. 3D graphics coming soon.
 - ◇ Used mainly for interactive drawing and rendering dynamic data driven charts
- Audio and Video
 - ◇ Used for playing sounds and movies without a browser plug-in
- Browser side data storage
 - ◇ **Web Storage** – stores small amounts of data using an easy name value pair based API
 - ◇ **IndexedDB** – stores large amounts of structured (table and column oriented) data. No SQL.
 - ◇ **Web SQL Database** – SQL-oriented. May lose out to IndexedDB.

- ◇ Used for offline applications and caching Ajax pulled data to improve performance.

1.14 New Features of HTML5

- Offline Applications
 - ◇ Allows user to be productive when not online
 - ◇ Examples:
 - *E-mail program that allows sending an e-mail while offline*
 - *Employee contact app that works when the browser is offline*
 - ◇ Key strategies to build offline app:
 - *Use a data storage mechanism to retain data locally. Sync local data with server when online.*
 - *Cache static files (e.g., CSS, HTML, JavaScript) in the browser.*
- New input types
 - ◇ Includes e-mail, date, time, datetime, number, url, search and range
 - ◇ Contains built-in validation logic which eliminates the need for JavaScript code

1.15 New Features of HTML5

- Web Workers
 - ◇ Allows an application to perform background work in a different thread
 - *Takes advantage of multi-core machines*
 - ◇ Main GUI thread and the background thread communicate through the messaging API
- WebSocket API
 - ◇ Allows the client and the server to push messages to each other at any given time
 - ◇ Serves as an alternative to AJAX, which only allows a client to send a

message to a server, not vice versa

1.16 The Function over Form Philosophy

- Many of the “old faithful” means of structuring pages have been removed
 - ◇ Frames / Framesets
 - ◇ Table sizing and content alignment
- Similarly, many of the style-oriented HTML elements have been removed
 - ◇ ``
 - ◇ `<center>`
 - ◇ Table styling / formatting attributes
- What's the message?
 - ◇ HTML is for **CONTENT**
 - ◇ CSS is for **STYLE**

Function over Form

Elements that won't validate in HTML5

- `<applet>`
 - Use `<embed>` instead
- `<big>`
- `<blink>`
- `<center>`
- ``
- `<marquee>`
- `<frame>` / `<frameset>`

You can validate your pages at <http://html5.validator.nu>

1.17 Semantic Elements

- Semantic elements in **HTML 4**

- ◇ div
- ◇ span
- **New in HTML5**
 - ◇ header
 - ◇ footer
 - ◇ section
 - ◇ article
 - ◇ aside
- **<article>**
 - ◇ Reusable content
- **<section>**
 - ◇ Not meant to be reusable
 - ◇ Can have header
 - ◇ Can be used as the parts of an article or page

1.18 HTML4 Layouts

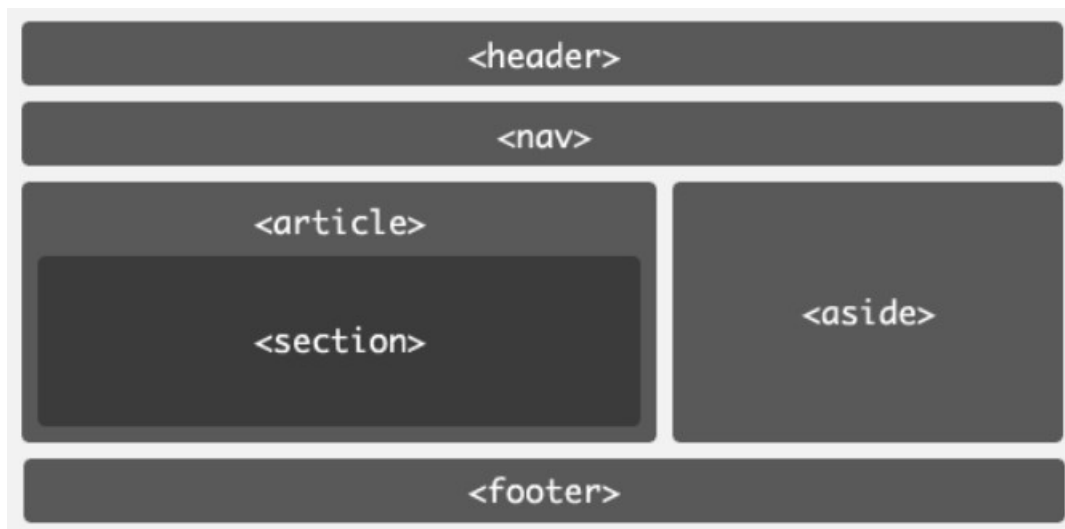
- Traditionally, we relied upon <div> tags to organize content and manage page flow. This had several problems:
 - ◇ We typically re-created the wheel each time, defining very predictable divs EVERY time (i.e. header, footer, nav, content, etc.)
 - ◇ These <div> tags are indistinguishable from each other from a programmatic / robotic perspective (i.e. content assistance devices, bots, crawlers, etc.)



Traditional div-centric layout for HTML4 Pages

1.19 HTML5 Semantic Layouts

- With HTML5, new tags have been introduced to provide clarity, consistency, and semantically rich structure to pages.



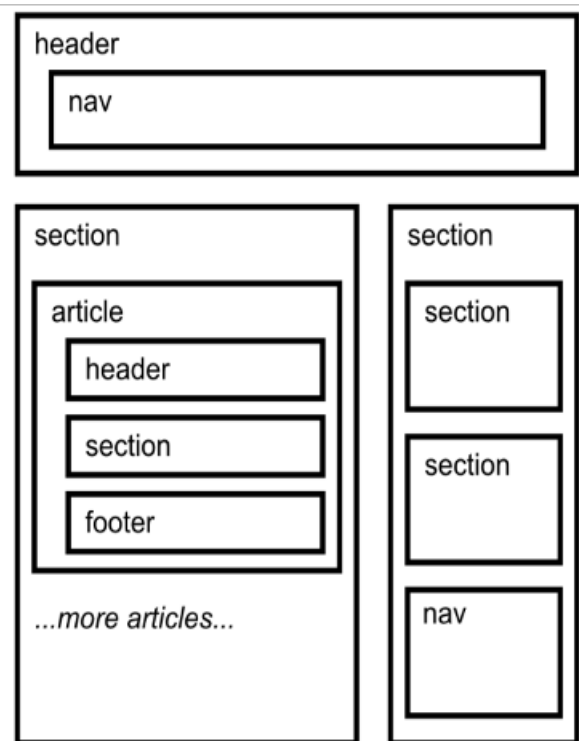
HTML5 Semantic Layouts

While the spec does promote the use of these new tags for organizing content, this should not be interpreted as a replacement for the `<div>` tag. There are still plenty of viable uses for `<div>`. The primary purpose is to support styling and layout.

Although the new elements appear to merely be replacing the role of the <div> tag, they are actually different. They are focused on identifying the nature of the CONTENT. And while it is possible to style them, their primary purpose is semantics. So for scenarios where you need to define style / layout for a chunk of the page, your primary tool will still be CSS + <div>.

1.20 Nesting Semantics

- At first glance, you might assume that you can only have one <header> per page, one <nav>, and so on
- In reality, you can combine and nest these elements as you see fit to represent more complex content structures



1.21 Replacing Flash with HTML5

- Wherever prudent, you can start to replace Flash with HTML5 with the goal to reach the largest customer base, specifically mobile devices.
- **Animation:** Many animation effects can be achieved using CSS3, SVG, Canvas and JavaScript.
- **Video/Audio playback:** For non-commercial use, start using HTML5 video with Flash fallback.
- **Data storage and offline:** HTML5 can replace Flash (and Google Gears) when it comes to browser side data storage and offline support.

Replacing Flash with HTML5

There are many uses of Flash that can not be done easily using HTML5. Complex games are good examples of that. But, there are many features that can be achieved using HTML5, CSS3 and JavaScript. You have to be really careful about using Flash since you run the risk of alienating many mobile users.

One of the most common uses of Flash is video. For non-commercial usage, where DRM (Digital Rights Management) and advertisement reporting is not a requirement, you can start using HTML5 video. As you will learn in the video chapter, you can provide a flash fallback option. If a browser does not support HTML5 video, it will resort to using a Flash player.

Flash is also used for simple animation effects. This is no longer necessary. You can do a lot using CSS3, Canvas and JavaScript. Toolkits like jQuery have good support for animation.

In November 2011, Adobe announced that it would discontinue development of Flash for mobile devices and reorient its efforts in developing tools utilizing HTML5. Its future work with Flash on mobile devices would be focused on enabling Flash developers to package native apps with Adobe AIR for all the major app stores. See the following Adobe announcement for more details:

<http://blogs.adobe.com/conversations/2011/11/flash-focus.html>

In fact, Flash applications, which formerly could not run on iOS devices, now can if they're packaged as native iOS applications using Adobe AIR and the iOS Packager.

<http://helpx.adobe.com/flash/using/packaging-applications-air-ios.html>

1.22 Summary

We have discussed

- What is HTML5
- HTML5 Goals
- HTML Specifications
- Differences from HTML4
- HTML5 is Not XML (not based on SGML)
- Required Processing for Invalid Markup
- The DOCTYPE Declaration
- Semantic Elements
- Browser Support

Chapter 2 - Forms

Objectives

Key objectives of this chapter

- Associate form input elements anywhere on the page with a form using the form attribute
- Use the new placeholder attribute
- Use the new input types
- Specify client-side validation without custom JavaScript
- Use the new autofocus and oninput attributes
- Describe HTML5 CSS Pseudo-Classes

2.1 The form Attribute

- Associate form input elements anywhere on the page with a form using the form attribute
- Example

```
<form id="example-form">
    <input name="wk" type="week"/>
    <input type="submit"/>
</form>
...
<textarea name="prose"
    form="example-form">
</textarea>
```

- When the form is submitted, the value of the "prose" textarea is included.

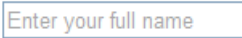
2.2 The placeholder Attribute

- Defines contents of text field before the user has entered a value
- After the user enters a value inside the field, the text is removed
- If focus is lost again without user entry, the placeholder value is restored

■ Example

```
<input type="text"
      placeholder="Enter your full name"/>
```

■ Result



The placeholder Attribute

Originally in HTML5, if a placeholder was specified for an input field, then as soon as you clicked inside the field, the placeholder text was removed by the browser. However, the specification was changed, such that the browser now only removes the placeholder text if the user types something inside the field. The field having focus isn't sufficient for the placeholder text to be removed. As of the time of the writing of this, Chrome 25 and Firefox 19 reflect the new behavior, whereas Opera 12.14 and Safari 5.1.7 still reflect the old behavior.

2.3 New Form Field Types

- | | |
|-----------------------|-----------------------------|
| ◇ input type=search | ◇ input type=datetime |
| ◇ input type=tel | ◇ input type=date |
| ◇ input type=url | ◇ input type=month |
| ◇ input type=email | ◇ input type=week |
| ◇ input type=number | ◇ input type=time |
| ◇ input type=range | ◇ input type=datetime-local |
| ◇ input type=checkbox | ◇ input type=color |
| | ◇ input type=image |

2.4 New Form Field Types

- ◇ textarea
- ◇ select
- ◇ fieldset
- ◇ datalist
- ◇ keygen
- ◇ output
- ◇ progress
- ◇ meter

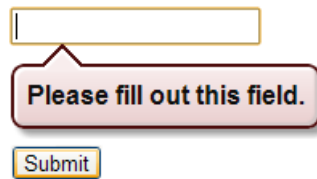
2.5 Forms and Validation

- Validation is needed on both the client and the server side
- Up until now, client-side validation required JavaScript
- Still need JavaScript to support browsers that don't support HTML5
- New input types provide validation, including
 - ◇ email
 - ◇ date
 - ◇ number
 - ◇ range
- Validation is also available through new attributes
 - ◇ pattern
 - ◇ required
- If the browser doesn't support HTML5 the new input types will be ignored and be treated as type="text"

2.6 The required Attribute

- Example

```
<input type="text" name="name" required="required"/>
```
- Result (on form submission if the field is empty)



NOTES

There are actually several valid options for indicating required fields:

```
<input type="text" id="car" required />
<input type="text" id="car" required="" />
<input type="text" id="car" required=" " />
<input type="text" id="car" required=required />
<input type="text" id="car" required="required" />
<input type="text" id="car" required='required' />
```

2.7 The number input type

- Example

```
<label for="ageId">Age: </label>
<input id="ageId" name="age" type="number"
      min="18" max="120" step="1"/>
```

2.8 The pattern Attribute

- User input may be validated against a regular expression

- Example

```
<label for="phoneId">Phone: </label>
<input id="phoneId" type="text"
      pattern="\d{3}\-\d{3}\-\d{4}"
      placeholder="XXX-XXX-XXXX"/>
```

2.9 The range and date input types

- Example


```
<input type="range" min="1" max="11" value="3"
      name="volume"/>
```

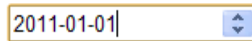
- Result



- Example

```
<input type="date" name="dob"/>
```

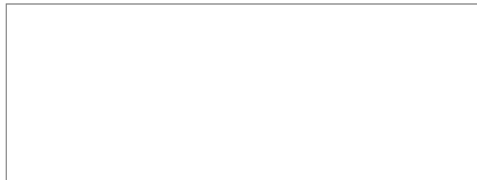
- Result



2.10 The <datalist> Element

- Provides auto-complete for text fields to provide valid choices

```
<label for="country_name">Country : </label>
<input id="country_name"
      name="country_name"
      type="text"
      list="country" />
<datalist id="country">
  <option value="Afghanistan">
  <option value="Albania">
  <option value="Algeria">
  <option value="Andorra">
  <option value="Angola">
  <!-- full list removed for brevity -->
</datalist>
```



2.11 The autofocus and oninput Attributes

- **autofocus**

- ◇ Should only be one per page
- ◇ Browser will give focus to this element when the page is loaded

- **output**

- ◇ Displays the results of a calculation directly in the browser without the need for JavaScript
- ◇ Data can be collected from other fields, the calculation performed, and the results dynamically displayed in the browser

2.12 The autofocus and oninput Attributes

- **oninput**

- ◇ Example
 - When the value of rating changes, that new value will be displayed
 - Can be used to display the numerical value of a slider

```
<form name="f1" >
  Darkness Level:
  <input type="range" name="darkness"
    value="0" min="0" max="100"
    oninput="output.value =
      parseFloat(darkness.value);">
  <output name="output">0</output><br/>
</form>
```

2.13 HTML5 CSS Pseudo-Classes

■ Examples

```
input:invalid {  
    border-color: red;  
}  
input:valid {  
    border-color: green;  
}
```

- These styles will be applied as the input elements change state based on validation.

NOTES

A **pseudo-class** is information about an element that's in the document tree but not available through any specified attributes. *For example, the `:first-child` pseudo-class applies to elements which are the first child of their parents. You can get that information from the DOM tree, but there's no `first-child` attribute.*

In CSS2.1, there were a handful of pseudo-classes available, notably the link states (`:link`, `:visited`) and those of user actions (`:active`, `:hover`).

In CSS3, the list of pseudo classes is dramatically expanded. The primary motivation is to tie-in with the validation scheme and to support conditional formatting of page content (especially as it relates to form validation).

- Required / Optional
- Valid / Invalid
- Out-of-range / In-range (*for numbers*)
- Read-only / read-write

2.14 Summary

- We have discussed the following HTML5 features relating to forms
 - ◇ Form input elements anywhere on the page can now be associated

with a form with the form attribute

- ◇ The new placeholder Attribute
- ◇ The new input types
- ◇ Client-side validation without custom JavaScript
- ◇ The new autofocus and oninput Attributes
- ◇ HTML5 CSS Pseudo-Classes

Chapter 3 - Canvas

Objectives

Key objectives of this chapter

- Contrast <canvas> with <svg>
- Describe browser support for <canvas>
- Use the <canvas> element
- Use the JavaScript context object to draw
 - ◇ Text
 - ◇ Rectangles
 - ◇ Color
 - ◇ Gradients
 - ◇ Paths
- Create patterns
- Use transformers

3.1 The <canvas> Element

- Provides a 2D drawing context
 - ◇ Lines
 - ◇ Fills
 - ◇ Images
 - ◇ Text
- One of the most exciting new features of HTML5
- Example uses:
 - ◇ Animation
 - ◇ Charts and Graphs
 - ◇ Rich user interaction
 - ◇ Games

- ◇ Drawing tools (functionality similar to Microsoft Paint) can be--and have been--implemented using `<canvas>`

3.2 `<canvas>` vs. `<svg>`

- HTML5 provides two drawing APIs
 - ◇ `<canvas>`
 - Pixel-oriented
 - A lower-level API than SVG
 - Ideal for situations where no mouse interaction is required since it does not maintain a tree of individual elements
 - Can still handle events at the document level
 - Supports animation and other JavaScript-intensive interaction
 - ◇ `<svg>`
 - Vector-based (thus supports scaling)
 - The tree of SVG elements is available while the page is displayed, allowing
 - Binding to specific elements
 - Listening for click or touch events
 - Can easily import and export from tools such as Adobe Illustrator and Inkscape

3.3 `<canvas>` vs. `<svg>`

- Our topic in this unit is `<canvas>` but here is one `<svg>` example:

```
<svg width="100" height="100"
  xmlns="http://www.w3.org/2000/svg">
  <circle cx="20" cy="20" r="50" fill="green"/>
</svg>
```
- `<canvas>` issue: To the browser, a canvas is just an array of pixels
 - ◇ Whatever is written there, even text, is not available to assistive

technology (e.g., screen readers)

3.4 Browser Support for <canvas>

- <canvas> is supported in the following browsers and devices:
 - ◇ The Big Five
 - IE 9
 - Firefox 3.0+
 - Safari 3.0+
 - Chrome 3.0+
 - Opera 10.0+
 - ◇ Mobile Platforms
 - iPhone 1.0+
 - Android 1.0+
- What about IE 8 and below?
 - ◇ Scripts are available, including ExplorerCanvas, that allow you to use the canvas element and API in your code without modifying your code

```
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
```

3.5 Creating the Canvas

- A <canvas> without JavaScript doesn't display anything
- It just defines a rectangular region on the page

```
<canvas id="canvas1">
  This browser does not support canvas.
</canvas>
```
- The default size is 300px x 150px
- Use the width and height attributes to change the size

```
<canvas id="canvas1" width="30" height="30">
</canvas>
```

- The values are in pixels

3.6 Using the Context

- You can draw on a canvas with JavaScript
- The first thing you need is the context; think of it as the pen you are drawing with.
- The default color is black.

```
<script>
  var ctx = document.
    querySelector("#canvas1").getContext("2d");
  ctx.fillText("Hello, world", 10, 20);
  ctx.fillRect(10, 30, 50, 50);
</script>
```

- Result



- Note: The `<canvas>` element is only required to support a '2d' context
 - ◇ WebGL: a JavaScript library
 - Not developed by the W3C
 - Extends the canvas element with a 3d context

3.7 Using the Context

- A more robust alternative:

```
var canvas = document.querySelector("#canvas1");
if (canvas && canvas.getContext) {
  var context = canvas.getContext('2d');
  if (context) {
    // Your code here!
```



```
}  
}
```

3.8 Using Color

- To color the rectangle and draw a box around it

```
var ctx = document.  
    .querySelector('#canvas1').getContext('2d');  
ctx.fillStyle = 'rgb(100, 100, 0)';  
ctx.fillRect(10, 30, 50, 50);  
ctx.strokeStyle = 'rgb(80, 80, 0)';  
ctx.lineWidth = 5;  
ctx.strokeRect(10-1, 30-1, 50+2, 50+2);
```



3.9 Painting Gradients

- Example

```
var canvas2 = document.querySelector('#canvas2');  
var ctx2 = canvas2.getContext('2d');  
var gradient2 = ctx2.  
    .createLinearGradient(0, 0, 0, canvas2.height);  
  
gradient2.addColorStop(0, '#00ff00');  
gradient2.addColorStop(1, '#000000');  
ctx2.fillStyle = gradient2;  
ctx2.fillRect(0, 0, canvas2.width, canvas2.height);
```



- A createRadialGradient(...) method is also available

3.10 Drawing Paths

- Example

```
var ctx3 =  
document.querySelector('#canvas3').getContext('2d');  
ctx3.beginPath();  
ctx3.arc(40, 50, 30, Math.PI-1, Math.PI+1, true);  
ctx3.strokeStyle = '#ff0000';  
ctx3.lineWidth = 3;  
ctx3.stroke();
```



- Angles are measured in radians with zero to the right.

```
var radians = degrees * Math.PI / 180;
```

- 'true' means counterclockwise

3.11 Drawing Paths

- Additional methods available

- ◇ quadraticCurveTo(...)
- ◇ bezierCurveTo(...)
- ◇ arcTo(...)
- ◇ rect(...)
- ◇ clip(...)
- ◇ isPointInPath(...)

3.12 Painting Patterns

■ Example

```
var canvas4 = document.querySelector('#canvas4');
var ctx4 = canvas4.getContext('2d');
var img = document.createElement('img');
img.onload = function () {
    ctx4.fillStyle = ctx4.createPattern(this, 'repeat');
    ctx4.fillRect(0, 0, canvas4.width, canvas4.height);
};
img.src='HTML5_Badge_32.png';
```



3.13 Transformers

- `ctx.rotate(...)` rotates the canvas itself, affecting the results of subsequent drawing.

```
var ctx =
document.querySelector('#canvas1').getContext('2d');
ctx.fillStyle = 'rgb(100, 100, 0)';
ctx.fillRect(10, 20, 50, 50);
ctx.rotate(0.1); // Added !!!
ctx.strokeStyle = 'rgb(80, 80, 0)';
ctx.lineWidth = 5;
ctx.strokeRect(10-1, 20-1, 50+2, 50+2);
```



3.14 Summary

- We have discussed
 - ◇ <canvas> Overview
 - Contrasted with <svg>
 - Browser Support
 - ◇ The <canvas> Element
 - ◇ The JavaScript Context Object
 - ◇ Using Color
 - ◇ Creating Gradient
 - ◇ Drawing Paths
 - ◇ Patterns
 - ◇ Transformers

Chapter 4 - Video and Audio

Objectives

Key objectives of this chapter

- Use the <audio> and <video> Elements
- Specify More Than One Source
- Use the poster Attribute
- Write Custom Controls With JavaScript

4.1 HTML5 Video/Audio Overview

- HTML5 ensures native support for video and audio
 - ◇ Video is now a standardized part of the web
- Users no longer need to download a separate browser plugin for audio/video playback
- Benefits to the user
 - ◇ Improved browser stability
 - ◇ Better user experience
- Benefits to the page developer
 - ◇ No longer need to use <object> and <embed>
 - ◇ Can style audio and video elements

4.2 New Elements for Video/Audio

- <video>
- <audio>
- The HTML5 spec does not require browsers to support any particular video codecs
- Typical Examples
 - ◇ MPEG-4

- ◇ H.264
- ◇ Ogg Theora
- ◇ WebM
- Similarly for Audio Codecs

4.3 Using the <audio> Element

- Example

```
<audio src="guitar.wav" controls="controls">  
  This browser does not support the audio element.  
</audio>
```

- Possible result



- If you don't specify the controls attribute you must specify autoplay

```
<audio src="guitar.wav" autoplay="autoplay"/>
```

4.4 The <video> Element

- Example

```
<video src="scene01.ogv"></video>
```

- Can specify width and height attributes to control the size of the video
 - ◇ Regardless of what you set, aspect ratio is always preserved
 - ◇ **Option 1:** Specify only one of width OR height
 - *The browser calculates the other to satisfy the aspect ratio*
 - ◇ **Option 2:** Specify both attributes
 - *If aspect ratios match, no problem*
 - *Otherwise, the browser uses blank space as necessary to preserve the aspect ratio (i.e. Letterbox or Pillarbox)*
 - ◇ **Option 3:** No height or width are specified

- *If a poster image is specified, the browser uses its dimensions*
- *Otherwise 300px x 300px*

4.5 Specifying More Than One Audio or Video File

- Example

```
<audio controls="controls">
  <source src="guitar.wav" type="audio/wav"/>
  <source src="guitar.ogg" type="audio/ogg"/>
</audio>
```

- Note: specify the MIME type -- and thus the codec
- The browser attempts the files in order until it finds one with a codec it supports
- In HTML5 the media attribute can be used to download a different file based on the size of the device
 - ◇ This attribute originated as part of the CSS Media Queries specification

```
<video controls>
  <source src="scene01-hi-res.ogv"
          media="(min-device-width:800px)">
  <source src="scene01-lo-res.ogv" ...
  ...
</video>
```

4.6 The poster Attribute

- By default the video playback area is empty during download
 - ◇ Nothing is shown until the user clicks play
- Can specify an image for the browser to display while the video loads

```
<video src="scene1.ogv" controls="controls"
        poster="cover.jpg">
  ...
</video>
```

4.7 Other <audio> and <video> Attributes

- loop
- preload
 - ◇ preload="auto"
 - A suggestion to the browser to preload
 - ◇ preload="none"
 - ◇ preload="metadata"
 - A suggestion that the browser should prefetch metadata such as dimensions, first frame, track list, duration, etc.

4.8 JavaScript and Media Elements

- The <audio> and <video> JavaScript elements have these methods
 - ◇ play()
 - ◇ pause()
 - ◇ load()
 - ◇ canPlayType(type)
 - ◇ addTextTrack(label, kind, language)
- The elements also share
 - ◇ Numerous attributes
 - ◇ Numerous events
- It is thus possible to create custom controls

4.9 Summary

- We have discussed
 - ◇ The <audio> Element
 - ◇ The <video> Element

- ◇ Specifying More Than One Source
- ◇ The poster Attribute
- ◇ Other <audio> and <video> Attributes
- ◇ JavaScript and Custom Controls

Chapter 5 - Introduction to CSS3

Objectives

Key objectives of this chapter

- Define Cascading Style Sheets
- Relation of CSS to HTML and web development
- Basic syntax of CSS rules
- Summary of new features in CSS 3

5.1 What is a Style?

- Styles allow for the control of how text is displayed
 - ◇ This can generally be applied to word processor programs or web pages in a browser
- A style can combine multiple properties that can be applied to similar types of text
 - ◇ This could perhaps make all titles of sections in a document larger, bold and a different font
- A style combines the properties that it will modify under a single name so this can easily be applied to multiple elements of text that should be displayed in a similar fashion
- Styles also allow for easily changing the appearance of a document since you only need to change the style definition to affect all of the text that has the style applied to it

5.2 What are Cascading Style Sheets?

- Cascading Style Sheets, or CSS, bring the same type of text styling to web pages
- The syntax of CSS is not HTML but it can modify the properties of how standard HTML tags are displayed
 - ◇ For example, you could establish a style rule that will modify how all HTML <h1> tags would appear

- CSS can also establish standalone styles, that are not linked directly to HTML tags and allow you to apply a style to elements of a web page using the name of the style
- CSS style rules are defined using a “style sheet language” that is separate from HTML
- Web pages will often have links to CSS files from a web server that contain the style rules referenced in the web page
 - ◇ The browser will load the HTML source of the page along with the CSS file and apply the styles to the page when displaying it

5.3 CSS and the Evolution of Web Development

- CSS, along with many other Web standards, is guided by the 'World Wide Web Consortium' (www.w3.org) and is comprised of hundreds of member organizations with interest in the development of the Web
 - ◇ It is also often referred to as the 'W3C'
 - ◇ The W3C defines a number of standards with HTML being the most important for CSS
- CSS Level 1 was released in 1996
 - ◇ It contained many of the “core” elements still present in CSS today including text formatting, fonts, and margin properties
 - ◇ CSS 1 did not include ability to set the exact position of page elements so an intermediate CSS-Positioning was released
- CSS Level 2 was released in 1998
 - ◇ This focused on international accessibility and defining styles specific to particular media types
- CSS Level 3 is not a single specification but a group of multiple “modules” being developed and released concurrently
 - ◇ Currently the core modules have a final level 3 release with most of the other major modules being in at least the “Candidate Recommendation” stage

CSS and the Evolution of Web Development

CSS Level 2.1 was released in 2006 which had some fixes and clarifications. This means CSS 3 is some of the first major changes to CSS in quite a while.

5.4 The CSS Standardization Process

- There is no such thing as the CSS₃ standard. Each CSS module is now being standardized independently
- "CSS3" consists of CSS2.1 modules that are being evolved and extended and that may have different level numbers



Source: <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3>

5.5 CSS and HTML

- Although CSS and HTML are separate specifications, CSS certainly is intended to compliment HTML
- The release of CSS and HTML specifications is independent although often changes in one will start to drive changes in the other
- HTML5 and CSS 3 are currently being developed at the same time with both already having portions finalized
 - ◇ This is more of a happenstance than the result of coordination between the specifications though

- ◇ These versions are quickly becoming the foundation for modern web sites as more features of both are finalized and supported by browsers

5.6 CSS Compatibility

- Although the newest versions of CSS (and HTML) will provide an expanded range of features they are not supported by all versions of browsers that are in use
 - ◇ Support varies not only by the type of browser (Internet Explorer, Firefox, Chrome, etc) but also based on the released versions of these browsers
- Like most Web technologies, you must determine how important backwards compatibility with older browsers is and balance that against using some of the newest features
 - ◇ This also depends on if you can set minimum browser versions and types that users of your web site can use
- The good news with CSS is that often if a browser doesn't recognize a style property or element it will simply ignore it and just display unstylized text
 - ◇ This is different than with HTML where if you use an unrecognized HTML tag it won't display on the page at all
- It is always good to plan for (and often test) how a page would be displayed if some or all of the styles defined are not applied

CSS Compatibility

There are some web sites available that list what features of CSS (and HTML) are supported by which browsers and what version that support may have been added. Combine this with some reporting on what types and versions of browsers your web site visitors might be using and this can help you decide on what elements of the newest specifications can be used.

http://www.w3schools.com/cssref/css3_browsersupport.asp

<http://caniuse.com/>

If you are creating a web site for use by a small and/or internal group of users you may be able to set more aggressive standards for what browsers and versions are “supported”. Web sites available to the general public usually require support for a wider range of browsers although usage of most of the

oldest browsers that would have the most compatibility problems is almost non-existent.

5.7 CSS Rules

- A CSS “Rule” defines what the HTML should look like and how it should behave in the browser window
- A CSS rule has three main elements
 - ◇ Selector – Indicates what the rule will apply to
 - ◇ Properties – identifies what is being defined
 - ◇ Values – Assigned to a property and can be words, numbers, or a percent depending on the property

selector { property: value; }

- Properties and values are separated by a colon
- A property and its value are a “declaration” and a CSS rule may have multiple declarations separated by semicolons within the curly brackets that define the rule
 - ◇ Even if there is a single declaration it is a best practice to end it with a semicolon to avoid mistakes if additional declarations are added later

5.8 New in CSS3

- A sampling of some of the new things in CSS3
 - ◇ Animation – moving objects without JavaScript
 - ◇ Borders – multiple colors, border images, and rounded corners
 - ◇ Backgrounds – multiple backgrounds and precise control
 - ◇ Color – color opacity, gradients in backgrounds
 - ◇ Text – text shadows, overflow and word wrapping
 - ◇ Transformations – scale, move and rotate in 2D and 3D
 - ◇ Transitions – simple dynamic style transitions
 - ◇ Box – drop shadows, horizontal and vertical overflow

- ◇ Content – styles can add content to elements
- ◇ Media queries – styles can be applied based on things like viewing size, aspect ratio and resolution (important for mobile/tablet)
- ◇ Web Fonts – link to downloadable font files for uniform fonts on pages

5.9 Summary

- Cascading Style Sheets, or CSS, allow web page authors to define how they want to alter the display of the page
- Styles are defined using CSS rules
- CSS is an independent specification from HTML although they are used together
- Much of CSS 3 is finalized with many major modules going through the last stages of release

Chapter 6 - Applying CSS Styles

Objectives

Key objectives of this chapter

- Define different ways to apply styles
- Understand the role of "selectors" in style rules
- Define reusable style classes

6.1 Inline Styles

- An "inline" style sets the style only of the tag it is applied to
- The '**style**' attribute of the tag is used along with style declarations
 - ◇ You can have multiple declarations separated by semicolons

```
<h1 style="color:red;font-style:bold;">...</h1>
```

- This will override styles applied in other ways if the same properties are set in the declarations
- It is best to use double quotes (") around the entire declaration list and single quotes (') around any value in the declaration list that might need them (eg. font names with spaces)
- Ideally this should be avoided as much as possible because this would make it so that styles defined in more reusable (and maintainable) ways do not have an effect on the particular element

6.2 Embedded Styles

- Style rules can be embedded within **<style>** tags within a web page
 - ◇ This will apply those style rules to the entire page
 - ◇ Multiple style rules can be used with the curly brackets defining the content of each
 - ◇ You can use indentation and multiple lines to make the style rules more readable without impacting how they apply

```
<style type="text/css">
  h1 {
```

```
        color: orange;
        font-weight: normal; }
    strong {
        font-weight: italic; }
</style>
```

- Although the `<style>` tag can be placed anywhere in the web page, it is best to put it in the `<head>` section so the styles are defined before the page itself is rendered

```
<head>
    <style type="text/css">
        ...
    </style>
</head>
```

6.3 External Styles

- Style rules can also be defined in external files called "style sheets"
 - ◇ This is the best way to reuse styles across multiple pages and provide consistency
- Using an external style sheet involves two steps
 - ◇ Save a text file with a '.css' file extension that contains only CSS style rules (no `<style>` tag)

```
(saved as global.css)
body { padding: 100px; }
h1 { color: red; }
```

- ◇ Link to the style sheet with a `<link>` tag in the `<head>` section of a web page with the 'href' attribute providing the URI to the file

```
<head>
<link rel="stylesheet" href="global.css" type="text/css"/>
</head>
```

- You can also use the '@import' rule in a `<style>` tag or another external style sheet to import a style sheet as long as it is before any other CSS syntax

```
<style type="text/css">
@import url(global.css);
```

```
h1 { color: red; }
</style>
```

External Styles

The 'rel="stylesheet"' attribute is important since the <link> tag can be used to link to different types of files. This makes sure the browser loads it as a style sheet.

You can split style rules into multiple files and include them with multiple <link> tags or @import rules. Having multiple files might make it more difficult to load all of them but might also be easier to maintain than one large file with all styles. You will need to adjust your approach based on how many pages are having styles applied and the number and type of styles being defined.

You could use a single <link> tag to point to a "master" CSS file that used the @import rule to import other style sheets. This may not load as quickly in the browser though as using multiple <link> tags on the page to directly link to the multiple style sheets that are needed.

6.4 Selectors

- CSS rules are applied to web page elements using 'selectors'
- There are 4 basic types of selectors
 - ◇ HTML – Uses the name of the HTML tag

```
h1 { color: red; }
```

- ◇ Class – Starts with a '.' to define a uniquely named style "class" that can be applied to any HTML element

```
.myClass { color: blue; }
```

```
<h1 class="myClass">..</h1>
```

- ◇ ID – Starts with a '#' to apply the style to the element with the given 'id' attribute

```
#myObject { font-style: italic; }
```

```
<h1 id="myObject">..</h1>
```

- ◇ Universal – A "wildcard" selector using an '*', often used in combination with other selectors or to set global properties

```
* { margin: 0; }
```

6.5 Combinator Selectors

- Multiple selectors can be combined within a style rule in various ways
 - ◇ These "combinator selectors" allow for complex selection of page elements
 - ◇ Any type of selector may be used in the combination
- A comma between them applies the declarations to each within the group
 - ◇ Selects all <div> elements and all <p> elements

```
div, p { .... }
```

- A space between selects the second element if it is a descendant of (within) the first element
 - ◇ Selects all <p> elements that are inside <div> elements

```
div p { ... }
```

<div><p>..**p**</p></div> is selected

<div><table>..**p**..**p**</p>..**p**</p></table></div> is ALSO selected

- A '>' between selects the second only if it is a direct child of the first
 - ◇ Selects all <p> elements where the parent is a <div> element

```
div > p { ... }
```

<div>pp</p></div> is selected

<div><table>..**p**..**p**</p>..**p**</p></table></div> is NOT selected

Combinator Selectors

Although the examples above used HTML selectors, the other types of selectors could be used as well. For example, the following would select all of the <p> tags which are descendents of the tag with the 'myObject' ID.

```
#myObject p { ... }
```

6.6 Combinator Selectors

- Some combinator selectors apply to "sibling" tags that are not embedded one inside the other but share the same parent
- A '+' selects the second element that is immediately after (or adjacent) the first element if both share the same parent

- ◊ Selects only sibling `` tags that immediately follow `` tags

```
strong + em { ... }
```

`..and so it was hard IS selected
...... IS NOT selected`

- A `'~'` selects any of the second element that is a sibling of the first element with the same parent

- ◊ This was introduced in CSS 3

- ◊ Selects all `` tags that are siblings of a `` tag

```
strong ~ em { ... }
```

`..............
BOTH 'em' tags selected`

6.7 Universal Selector

- The universal selector `'*'` represents a wildcard placeholder for selecting any tag
- Using the universal selector in combination with other selectors still has the same effect
 - ◊ The following would select any `` tag that is a descendent of any other tag that is a descendent of the `<p>` tag (rules out direct child elements)

```
p * em { ... }
```

`<p><code>..</code>..
....</p>`

Both 'em' tags would be selected

`<p>..</p>` would NOT be selected

`p em { ... }` would have selected ALL 'em' tags above

- The universal selector at the end of a combination of selectors indicates selection of the descendents of the parent tag listed before it
 - ◊ Selects ALL of the tags that are descendents of `<div>` but not the parent `<div>` tag itself

```
div * { ... }
```

6.8 Style Classes

- Rather than linking style rules to a particular HTML tag only, you can define reusable style classes
 - ◇ The name of the style class comes after a period (.) in the style rule selectors and defines an "independent" class

```
.chaptertitle { font-style: italic; }
```

- ◇ The class name can contain numbers, underscores, or hyphens but must start with a letter
- ◇ The style can be applied to any HTML tag using the 'class' attribute of the tag

```
<h1 class="chaptertitle">...</h1>
```

```
<a class="chaptertitle" href="..">...</a>
```

- Multiple style classes can be applied to a single tag by providing a space-separated list of class names

```
<h1 class="style1 style2 style3">..</h1>
```

6.9 Style Classes

- A "dependent" class can also be defined by providing an HTML tag name directly before the period and class name with no space between
 - ◇ This style rule will apply only when an HTML element of the correct type is linked to the class name
 - ◇ This helps redefine a class with a given name for a particular HTML tag to have slightly different declarations

```
.chaptertitle { font-style: italic; }
```

```
a.chaptertitle { font-size: 2em; }
```

6.10 Pseudo-Classes

- Pseudo-classes are a predefined state or use of an element that can be styled independently of the default state of the element
 - ◇ One example is an anchor link (<a>) tag that has styles for the link, followed link, hover, and active (clicked)

- ◇ The name of a pseudo-class is preceded by a colon which can optionally follow the name of an HTML tag

```
a:link { color: darkblue; }
a:visited { color: blue; }
a:hover { color: gray; }
a:active { color: red; }
```

- There are also several pseudo-classes that apply to the relation between elements

- ◇ These were greatly expanded in CSS 3

```
p:first-line { font-weight: bold; }
h1:first-letter { font-size: 1.5em; }
p:first-child { background-color: pink; }
    Selects every <p> element that IS a first child of
its parent
td:nth-of-type(2) { border: 2px; }
    Selects every <td> element that is the 2nd <td>
element of its parent
```

Pseudo-Classes

Oddly enough, the ':first-child' pseudo-class was part of CSS 2 but ':last-child' was added in CSS 3.

6.11 Inheriting From a Parent

- A style rule defined for a particular element will be in effect until the end of that element
 - ◇ This means that descendent tags will "inherit" the style properties of the parent
 - ◇ For example, a 'font-family' property set for a <p> element would still be in effect for child tags like , <a>, etc
- If a different value of a style property is set for a child element it will use that value and not inherit the value from parent elements
- Not all style properties are inherited by default since inheriting properties like margin, width and borders would apply those properties more than once

- If you want to explicitly indicate that a style property should inherit from the parent, many properties support a value of 'inherit'

Inheriting From a Parent

Inheriting style properties from parent tags makes sense. It would be very difficult if you needed to set all of the font properties every time a new tag was encountered.

6.12 Declaring !important Styles

- The presence of '**!important**' within the style declaration (before the semicolon) can give a style priority

```
<style type="text/css">
    h2 { color: green !important; }
    h2 { color: red; }
</style>
```

- Using '**!important**' can be used to help troubleshoot CSS code
 - ◇ If adding this to a style rule causes the rule to apply to an element that it had not been affecting before, the cascade order of CSS had been preventing the rule from applying
- Ideally styles for production code should avoid the '**!important**' declaration as it makes it hard or impossible to override a style rule when needed

Declaring !important Styles

In the above example for '**!important**', the second 'h2' style with the color red would have normally taken precedence because it was declared after the first 'h2' style. With the addition of the '**!important**' declaration though the first style takes precedence and the text would be green.

6.13 CSS Cascade Order

- A Web page can have multiple sources of styles through combination of embedded `<style>` tags, links to external style sheets and inline styles
 - ◇ This can lead to conflicts when multiple sources define a style for one or more properties of a page element and provide different values
- The "cascade order", or the factors impacting which style properties get

applied in conflicts, is shown below from high priority to low priority

- ◇ User – user styles generally take highest precedence
- ◇ !important – styles with the '!important' declaration
- ◇ Inline – styles declared inline in the HTML
- ◇ Specificity – style rules that are more specific in the selector for the rule declaration have higher precedence
- ◇ Order – style rules declared LATER will take precedence
- ◇ Inherited – style properties inherited from parent tags
- ◇ Default – the default styles of the browser

CSS Cascade Order

Most browsers will let the user set styles to override a web pages styles for accessibility reasons. This would let a user that has trouble reading always have larger text, for instance. In theory the page author would override user styles in a conflict unless the user style is declared with the '!important' declaration. In reality, most browsers honor user styles above page author styles to preserve the user experience for accessibility reasons.

Style sheets that are meant to provide the "defaults" for an entire site would be placed generally as the first style sheet referenced on each page. This way any other style sheets referenced for that specific page after the "defaults" would be able to override these default styles.

6.14 Summary

- There are several different ways to define style rules including inline, embedded, and in external style sheets
- A "selector" indicates which element(s) of a page will have a style rule applied
- Combining selectors in various ways can allow complex ways to select desired elements
- Defining reusable style classes allows applying that style to many different HTML elements by use of the 'class' attribute

Chapter 7 - Styling Text

Objectives

Key objectives of this chapter

- Understand font settings for web pages
- Use web fonts to provide greater font control
- Modify various aspects of text display with CSS properties

7.1 Web Typography

- Since most web pages will have text, typography is one of the most basic ways to control the expression of a page
 - ◇ Modifying the font, weight, size, letter spacing, etc can have a big impact on what a page conveys to the reader
- Typography on the web is based on "font families"
 - ◇ These are categories of typefaces that have similar characteristics
- In general a web page is providing information to a browser about how to display text with fonts available locally on the machine
 - ◇ CSS 3 added a way to avoid this reliance on local fonts

7.2 Generic Font Families

- CSS defines 5 generic "font families" that most fonts can be categorized into
 - ◇ Serif – small ornamentation at the end of a letter
 - eg. 'Times New Roman'
 - ◇ Sans-serif – fonts without serif ornamentation
 - eg. 'Helvetica' and 'Arial'
 - ◇ Monospace – each letter occupies the same amount of space with more space around "thinner" letters like 'i'
 - eg. 'Courier New'

- ◇ Cursive – mimic cursive or calligraphic handwriting
 - eg. 'Brush Script M7'
- ◇ Fantasy – decorative fonts not fitting into the other categories
 - eg. 'Papyrus'

7.3 Font-Stack and Understudy Fonts

- In a CSS style rule, the typeface is set as the "font stack" and the **'font-family'** property
 - ◇ The value for this property is the name of the font with quotes around if the name contains spaces

```
body { font-family: "Times New Roman"; }
```

- You can add multiple font names to the list, called "understudy fonts" and end the list with the generic font family name
 - ◇ Separate all with commas

```
body { font-family: Georgia, "Times New Roman", serif; }
```

- Specifying multiple fonts allows the browser to match the first one in the list that it can
 - ◇ Adding the generic font family at the end ensures that at the very least the browser would use the default font of that family to match the general style you might want
- You could override the font for certain HTML elements so they use something different than what is set for the body of the page

```
body { font-family: Georgia, "Times New Roman", serif; }
```

```
h1, h2, h3 { font-family: Arial, sans-serif; }
```

- There are various lists of "Web Safe Fonts" that should be available to most users

7.4 Web Fonts

- Relying on the browser to find a local font to display the page can be troublesome

- ◊ You must either use a font that is available on most browsers or those that will view the page or accept that some readers may see a different font
- CSS 3 adds the "web font" feature which allows you to define fonts in a CSS and link them to downloadable font files that will be accessible to any reader
- One issue with web fonts is that there are multiple file formats in use with different support by browsers but there are ways to overcome this
 - ◊ WOFF – Web Open Font Format is supported by the latest versions of all major desktop browsers but is relatively new and wouldn't be supported by older browsers
 - ◊ EOT – Embedded OpenType was developed by Microsoft and is the only other format supported by Internet Explorer
 - ◊ TTF/OTF – TrueType and OpenType are the most common for computer fonts and is the only other format supported by Firefox
 - ◊ SVG – Scalable Vector Graphics is the only other format supported by Chrome
- Various web sites can provide support for providing the different file formats and CSS code that would be supported by a majority of browsers
 - ◊ Verify the license of the font allows using it as a web font

Web Fonts

One web site that can help in getting fonts and creating "Webfont kits" is Font Squirrel:

<http://www.fontsquirrel.com/>

7.5 Using Web Fonts

- The first step in using web fonts is obtaining the various files for the different formats of the font and uploading them somewhere they will be available to your pages
- Within CSS code you use the '**@font-face**' rule, define a unique '**font-family**' name, and provide the URL to the source files of the font

```
@font-face {  
    font-family: BodyCopy;  
    src: url('../fonts/MyFont.eot');  
    src: url('../fonts/MyFont.eot?#iefix')  
format('embedded-opentype'),  
    url('../fonts/MyFont.woff') format('woff'),  
    url('../fonts/MyFont.ttf') format('truetype'); }
```

- You would then list the unique name you defined as the '**font-family**' for the web font in the normal '**font-family**' property of a style rule
 - ◇ You would normally list this first and also list understudy fonts and the generic font family if there is a problem

```
body { font-family: BodyCopy, Arial, sans-serif; }
```

Using Web Fonts

Remember the URL values will depend on what the relative link to the font files is from the location of where the CSS code is. This is easiest if you have external style sheets and font files that are in common locations.

The EOT format has two URLs because of a bug in versions of Internet Explorer before IE 9. Providing both helps make sure one can be used no matter which version of IE the reader is using.

When using web fonts you would normally list them first in the font-family property because that would help achieve the goal of letting all readers see the page with the same font.

7.6 Font Size

- There are two units most commonly used for setting font size in web pages:
 - ◇ px – pixels, relates to the resolution of the users screen
 - ◇ em – 1em is equal to the font size of the parent element so 1.5em is 1 ½ times the current font size
- The relative nature of the 'em' unit makes it very adaptable which is useful for mobile browsers and user settings for the sight-impaired
 - ◇ Using a percentage is also good
- Font size can be set in a style rule with the '**font-size**' property which takes one of the following values:

- ◊ <absolute> or <relative> value (eg. 20px or 1.5em or 75%)
- ◊ 'smaller' or 'larger' keywords relative to parent element
- ◊ 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', or 'xx-large' absolute size keywords

```
body { font-size: 100%; }  
h1 { font-size: 2em; }  
.caption { font-size: smaller; }
```

Font Size

The 'em' unit is not limited to font size and can be used for margin size, table padding, etc.

There are other units that are used for font size but not as common with web pages:

- in – inch
- cm – centimeter
- mm – millimeter
- pt – point, one point is 1/72 inch, this is most familiar from word processors
- pc – pica, 1pc = 12pt

7.7 Font Weight

- The '**font-weight**' property controls if the font is displayed bolder or lighter than normal
- There can be various values and depending on the support from the font they will have different effect
 - ◊ 'normal' – normal text
 - ◊ 'bold' – bold text
 - ◊ 'bolder' or 'lighter' – bolder or lighter than the font weight of the parent element
 - ◊ 100-900 – a value of 100-900 in increments of 100 can be used for relative boldness

```
strong { font-weight: bold; }  
.megaBold { font-weight: 900; }
```

- In reality most fonts may only have support for "regular" bold so the relative values will not provide the range they imply

7.8 Italics and Underlining

- Although it sounds more wide-ranging, the '**font-style**' property is just for italic and oblique text
 - ◇ It can have values of 'normal', 'italic' and 'oblique'
 - ◇ Italic text must have explicit support from the font usually with a lightly different design while "oblique" text is simply the normal font angled to the right

```
h1 { font-style: italic; }
```

- The '**text-decoration**' property allows underlining and a few other effects
 - ◇ 'underline' – Underlines text although this may look like a link to users
 - ◇ 'overline' – Places a line over the top of the text
 - ◇ 'line-through' – Also called "strikethrough" this places a single line through the middle of text
 - ◇ 'none' – this will clear any inherited value and display text normally

```
em { text-decoration: underline; }  
.strike { text-decoration: line-through; }
```

Italics and Underlining

If a font doesn't have support for italic then the 'italic' value of the '**font-style**' property would cause oblique to be used instead.

Many of the font properties have a 'normal' value so that you could revert text back to a "normal" display if the property of the parent element was setting a property to a non-default value.

You could use the 'none' value of the '**text-decoration**' property on the anchor (<a>) tag to display links without underlines normally added by browsers but this may confuse users.

If you do underline text it is probably a good idea to make sure links also show up with a different color so users can distinguish between links and text underlined for emphasis.

7.9 Capitalization

- The **'font-variant'** property can be used to create "small capitals" where the lowercase letters in the text are transformed into smaller capital letters
 - ◇ The property can have the values 'normal' and 'small-caps'

```
h1 { font-variant: small-caps; }
```

THIS HEADER IS IN SMALL CAPS

- The **'text-transform'** property has other capitalization options:
 - ◇ 'capitalize' – capitalizes the first letter of each word
 - ◇ 'uppercase' and 'lowercase' – changes all letters to the indicated case
 - ◇ 'none' – this will clear any inherited value and display text normally

```
h1 { text-transform: capitalize; }
```

This Header Is Capitalized

7.10 Line Height

- Spacing between lines can be controlled by the **'line-height'** property with the following values
 - ◇ 'normal' - reset the line spacing inherited from a parent element
 - ◇ <number> or <percentage> - relative decimal number, value in 'em', or percentage
 - ◇ <length> - an absolute spacing in something like 'px'

```
p { line-height: 2.0; }
```

```
h2 + p:first-letter { line-height: 24px; }
```

```
body { line-height: 1.5em; }
```

7.11 Multiple Font Values

- Rather than setting each font property as individual properties, you can use the **'font'** property to provide a space-separated list of many common font properties
- The property values are interpreted in a particular order:

```
font: <font-style> <font-variant> <font-weight>
```

<font-size>/<line-height> <font-family>;

- The **'font-size'** and **'font-family'** parts of the sequence are required, others are optional
 - ◇ If the **'line-height'** property is present, it is preceded by a forward slash (/) and without space after the **'font-size'**

```
body { font: normal 100%/1.5 Georgia, serif; }
h2 { font: small-caps 1.5em Arial, sans-serif; }
strong { font: italic bold 1em Georgia, serif; }
```

Multiple Font Values

Using the **'font'** property can be a useful shortcut for multiple font properties. The fact that it would always require a value for 'font-size' and 'font-family' means that the individual properties may still be the best way to set values when you do not want to specify font size or family. Often the **'font'** property would be used to setup some basic font combinations for a document and other style rules can modify the individual properties as needed.

7.12 Text Spacing

- The space between individual letters or words can be adjusted with the **'letter-spacing'** and **'word-spacing'** properties
 - ◇ Both can take a value of 'normal' or a <length> of absolute or relative units (eg. px or em)
 - ◇ The value supplied is the amount of additional space added compared to the normal for the font
 - ◇ The value can also be negative to condense the space between letters or words although this should be used sparingly

```
body { letter-spacing: 0.05em;
       word-spacing: 0.1em; }
h1 { letter-spacing: -1px;
      word-spacing: 3px; }
```

7.13 Aligning Text

- The **'text-align'** property sets the text alignment
 - ◇ The value can be 'auto', 'inherit', 'left', 'right', 'center', or 'justify'

```
body { text-align: left; }  
p { text-align: justify; }
```

- There is also a '**text-justify**' property that could control the various ways to justify the text that is defined as part of CSS 3
 - ◇ This may only be recognized by Internet Explorer although that could certainly change in the future

7.14 Summary

- The "font family" is the central concept behind specifying fonts to be used with web pages using CSS
- Web Fonts can be provided as downloadable files so that a browser can use the font you want even if it isn't available locally to the browser
- You have control over a great range of text display properties like italic, bold or underlining, font size, spacing, alignment and capitalization
- The '**font**' property can let you set a number of font properties all at once but the individual properties are still useful

Chapter 8 - Box Model and Effects

Objectives

Key objectives of this chapter

- Understand various parts of the CSS "Box Model"
- Controlling flow and display of elements
- Floating elements on the left or right
- Setting margin, border, and padding values
- Advanced CSS 3 options

8.1 Element Box Model

- All HTML elements can be considered as boxes
 - ◇ In CSS, the term "box model" is used when talking about design and layout
- The CSS box model is a box that wraps around HTML elements, and it consists of: margins, borders, padding, and the actual content



8.2 Parts of the Box Model

- The different parts of the box model are:
 - ◇ Margin - Clears an area around the border. The margin does not have a background color, it is completely transparent
 - ◇ Border - A border that goes around the padding and content. The border is inherited from the color property of the box
 - ◇ Padding - Clears an area around the content. The padding is affected by the background color of the box
 - ◇ Content - The content of the box, where text and images appear

8.3 Setting Width and Height

- The **'width'** and **'height'** properties can set the dimensions of the content box of an element
 - ◇ This is useful to set the size of an image instead of using an unknown "natural" size
 - ◇ The value can be 'auto', numeric <length> value, or <percentage>

```
header { height: 135px; }  
h1 { width: 90%; }
```

- You can also use the **'min-width'**, **'max-width'**, **'min-height'** and **'max-height'** properties to set a range for the width and height depending on how the viewing window is resized
 - ◇ These are not supported in IE6 and earlier though

```
h1 { width: 90%;  
      min-width: 660px;  
      max-width: 980px; }
```

Setting Width and Height

The 'auto' value for height or width uses the value calculated by the browser to display the content. This is the default behavior.

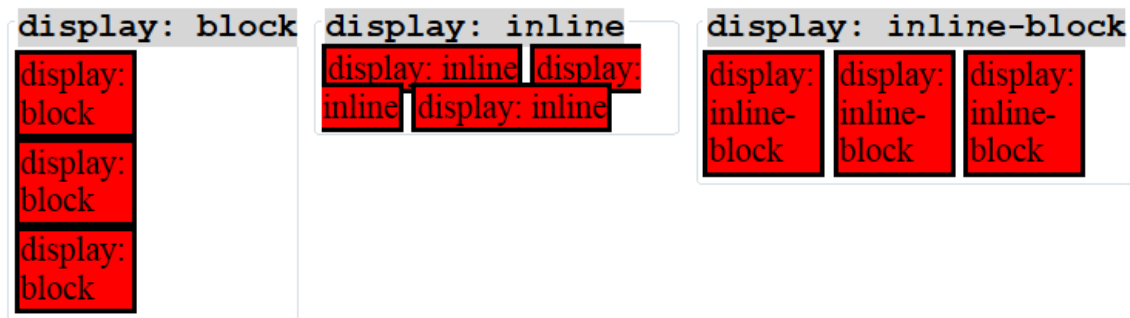
8.4 IE Box Size Bug

- In IE8 and earlier, the value for the **'width'** and **'height'** properties included the padding and the border which would lead to the content area being smaller if the border or padding were non-zero
 - ◇ Other browsers have followed the CSS standard more closely and use the property only for the size of the content box
 - ◇ Some have argued this made more sense because it would set the size of everything within the border and you won't have to adjust if padding and border values change
- CSS 3 introduced the **'box-sizing'** property so you can control how the browser handles this
 - ◇ IE support requires version 8+ and Firefox requires a **'-moz-box-sizing'** browser extension
 - ◇ A value of **'content-box'** will size only the content area as originally in CSS
 - ◇ A value of **'border-box'** will use the approach of including the border and padding in the size

```
div { box-sizing: border-box;  
      -moz-box-sizing: border-box; /* Firefox */ }
```

8.5 Controlling Flow in Position

- Values of the **'display'** property can set if an element flows horizontally with other elements or is stacked vertically
 - ◇ **'block'** has some whitespace above and below it and does not tolerate any HTML elements next to it
 - ◇ **'inline'** has no line break before or after it, and it tolerates HTML elements next to it. Doesn't recognize width or height properties
 - Inline elements can't contain block elements as children
 - ◇ **'inline-block'** is placed as an inline element (on the same line as adjacent content), but it behaves as a block element. Recognize width and height properties but flow inline



- Different HTML elements have different default values for how they display
 - ◇ `<div>` is a block element while `` is an inline element by default

Controlling Flow in Position

There are also other values for '**display**' defined by CSS 3 that would allow you to style any element as table elements or list items. It is generally better to avoid these though since there are already HTML elements for this that are built-in.

8.6 Hiding Content

- There are two different ways to remove content with the difference being if space is reserved for the hidden element(s)
 - ◇ Setting the '**display**' property to 'none' completely removes the element and does not reserve space for it
 - This is often used by JavaScript to dynamically hide or display parts of a page (like form elements depending on user choices)

```
.remove { display: none; }
```

- ◇ Setting the '**visibility**' property to 'hidden' reserves the usual space for the element but just doesn't display it

```
.hidden { visibility: hidden; }
```

- The '**visibility**' property can also have the following values:
 - ◇ 'visible'
 - ◇ 'collapse' which can only be used on table elements to hide a row or column without impacting layout

8.7 Overflowing Content

- By default, if content would require more space to be displayed, the 'height' value is overridden and renders the content outside the elements box
 - ◇ The content may "overlap" other content though as the box retains the specified height
- You can change the behavior of overflow with the '**overflow**' property
 - ◇ 'visible' is the default and overflow is not clipped but renders outside the element box
 - ◇ 'scroll' will always add scroll-bars even if they are not needed
 - ◇ 'auto' will add a scroll-bar only if needed and is generally the best value
 - ◇ 'hidden' will hide the content that overflows with no way to access it
- Overflow of the width and height can be controlled independently with the '**overflow-x**' and '**overflow-y**' properties added in CSS 3 which have some additional values available
 - ◇ 'no-display' would remove the entire box if the content doesn't fit (similar to display:none)
 - ◇ 'no-content' would leave the box but hide the content if it doesn't fit (similar to visibility:hidden)

Overflowing Content

Remember that since the 'overflow-x' and 'overflow-y' properties were added in CSS 3 they wouldn't be supported on older browsers.

8.8 Floating Elements

- CSS allows you to set how an element interacts with other elements by "floating" it
 - ◇ This will align the element to the left or right and cause subsequent elements to wrap horizontally around the element
 - ◇ The property can have values of 'left', 'right' and 'none' which is the

default and would display the element where it is without floating

```
body { float: left; }  
.figure { float: right; }  
    . This is some text. This is  
    ; some text. This is some  
    ; text. This is some text.  
    . This is some text. This is  
    ; some text. This is some  
    ; text. This is some text. This is some text.
```



- The **'clear'** property can be used to indicate if floating elements are "cleared" (or not allowed) on the left, right, or both directions
 - ◇ The value can be 'left', 'right', 'both', or the default 'none'

```
p { clear: left; }  
hr { clear: both; }
```

Floating Elements

When combined with the 'text-align: justify;' property, floating can give some sharp design to a page.

8.9 Using Float for Multiple Columns

- Although CSS3 adds the **'column-count'** property to define multiple columns of text, it is only supported on the most recent browsers
 - ◇ A more universal solution is needed
- You can use the **'float'** property on multiple elements to lay out elements horizontally and create multiple columns
 - ◇ Make sure to also set a width (fixed or relative) of the individual elements so they do not stretch the entire width of the parent element

```
<section>  
<nav style="float:left;width:250px;">...</nav>  
<article style="float:left;width:450px;">...</article>  
<aside style="float:left;width:200px;">...</aside>  
</section>
```

- One issue that can occur when using this technique is when block-level elements are floated within a parent block-level element

- ◊ In this circumstance the bottom of the parent element is not anchored and the browser can actually collapse the entire box of the parent as if no content were present
- ◊ One fix for this is to add a '**clear:both**' property to the last child element within the parent or on a **
** element after the last child that will anchor the bottom of the parent element and display with the desired height

Using Float for Multiple Columns

Designing something that is intended to be displayed with multiple columns and then not having it displayed that way can be a major issue as the layout could be unintelligible.

The CSS3 '**column-count**' property is also only recognized by some browsers if you use a browser-specific prefix like '**-moz-column-count**' and '**-webkit-column-count**' which makes this even a less attractive solution.

Another approach to solving the float issue is to set the '**overflow**' property of the parent element to any value. Using '**overflow:auto**' is generally the most common. Apparently setting this property reminds the browser to accommodate the space needed for the child elements.

8.10 Margins

- The "margin" is the space between the outside of the box border and other elements
- You can set the margin for all four sides with just the '**margin**' property with 1-4 values of <length>, <percentage> or 'auto' which lets the browser set the margin
 - ◊ 1 value - All four margins use the same value
 - ◊ 2 values - <top/bottom> <left/right>
 - ◊ 3 values - <top> <left/right> <bottom>
 - ◊ 4 values - <top> <right> <bottom> <left> (clockwise from top)
 - ◊ This "shorthand rule" of using 1 – 4 values is used in many other places
- You could also set each margin with the individual properties '**margin-top**', '**margin-bottom**', '**margin-left**', and '**margin-right**'

```
.figure { margin: 5px 5% 10px; }
```

```
.otherFigure {      margin-top: 5px;
                    margin-bottom: 10px;
                    margin-left: 5%;
                    margin-right: 5%; }
```

Margins

When using multiple values of the 'margin' property, the first value will always be the top margin, the second value (if present) will always be the right, the third value (if present) will be the bottom, and the fourth value (if present) will always be the left. If the left value is not present it will match the right and the bottom value will match the top if not present.

You could use the 'TRouBLE' mnemonic to remember that the order is Top, Right, Bottom, Left.

8.11 Padding

- The "padding" is the space inside the element box between the border and the content
- The '**padding**' property takes the same values as the '**margin**' property
 - ◇ This includes being able to use 1 – 4 values for the top, bottom, left, and right padding
- You could also set each the padding in each direction with the individual properties '**padding-top**', '**padding-bottom**', '**padding-left**', and '**padding-right**'

```
.figure { padding: 4px 2% 5px; }
.otherFigure { padding-top: 4px;
                padding-bottom: 5px;
                padding-left: 2%;
                padding-right: 2%; }
```

8.12 Border

- The border is the edge of the box of the element
- The easiest way to set this is with the '**border**' property which takes values of <border-width> <border-style> <border-color>
 - ◇ <border-width> can have values of <length>, 'thin', 'medium', or 'thick'
 - ◇ <border-style> can be 'dotted', 'dashed', 'solid', 'double', 'groove', 'ridge',

'inset', 'outset', 'inherit' or 'none'

- ◇ <border-color> can have a value of 'transparent', 'inherit' or any legal CSS color

```
.figure { border: 6px double rgb(143, 125, 132); }
```

- There are also individual properties for '**border-width**', '**border-style**', and '**border-color**' that can take 1 - 4 values that can be used for the top, bottom, left, and right borders just like the margin property

```
.complexFigure { border-width: 6px 3px 3px;  
                  border-style: double dashed;  
                  border-color: gray; }
```

Border

Remember that depending on the browser type (IE, Firefox, etc) and the value of the '**box-sizing**' property (if present), the width of the border may be part of the width of the element box.

There is also another option for specifying border which has separate width, style, and color properties for each side. This gives properties like '**border-top-width**', '**border-left-style**', '**border-bottom-color**', etc. This are mainly useful for overriding a particular value set using one of the other methods.

8.13 Outline

- An element can also have an "outline" that is displayed outside of the border but does not take space like the border and instead is displayed as part of the margin
 - ◇ The outline also doesn't have the ability to have different values on each side like the border and is a simple box
- The '**outline**' property will have values of <outline-color> <outline-style> <outline-width>
 - ◇ <outline-color> can have values of legal CSS colors, 'inherit' or 'invert' which performs a color inversion compared to the background to make sure the outline is always visible
 - ◇ <outline-style> has the same values available as '**border-style**'
 - ◇ <outline-width> has the same values available as '**border-width**'
- There are also separate properties of '**outline-color**', '**outline-style**', and

'outline-width' but these don't support multiple values like borders

```
.figure { outline: invert dashed 3px; }  
.otherFigure { outline-color: gray;  
               outline-style: solid;  
               outline-width: 6px; }
```

8.14 CSS 3 - Rounding Border Corners

- By default the corners of the border are square
 - ◇ CSS 3 adds a **'border-radius'** property that can round the corners of the border
- You can set the border corners individually with **'border-top-left-radius'**, **'border-top-right-radius'**, **'border-bottom-left-radius'** and **'border-bottom-right-radius'**
 - ◇ These can take 1 or 2 values of a <length> or <percentage>
 - ◇ Using two values can let you set the rounded corner to be a quarter ellipse with the first value being the horizontal radius and the second value being the vertical radius

```
.oddFig { border-top-left-radius: 20px 10px;  
          border-top-right-radius: 10px 25px;  
          border-bottom-right-radius: 5px 10px;  
          border-bottom-left-radius: 15px 25px; }
```

- The **'border-radius'** property can take 1 – 4 values as a shorthand way to set each corner
 - ◇ The order is <top-left> <top-right> <bottom-right> <bottom-left> with the value matching the "opposite" corner if there is no explicit value
 - ◇ A second 1 – 4 values after a forward slash (/) can be used to set a different horizontal radius

```
.oddFig { border-radius: 20px 10px 5px 15px / 10px 25px; }
```

CSS 3 - Rounding Border Corners

The two examples in the slide are equivalent. The shorthand in the **'border-radius'** example gives the same combinations in the example which has separate properties.

Even though it is part of CSS 3, **'border-radius'** has fairly good support across browsers.

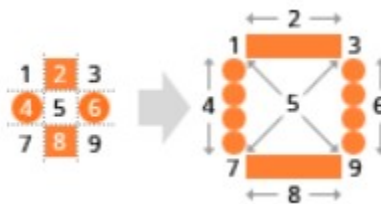
<http://caniuse.com/#feat=border-radius>


The behavior of the 1 – 4 values possible with '**border-radius**' is as follows:

- 1 value - All four corners use the same value
- 2 values - <top-left/bottom-right> <top-right/bottom-left>
- 3 values - <top-left> <top-right/bottom-left> <bottom-right>
- 4 values - <top-left> <top-right> <bottom-right> <bottom-left> (clockwise from top-left)

8.15 CSS 3 - Using a Border Image

- CSS 3 adds the ability to take an image and "slice" it into nine pieces to use as the corners and sides of a border



- The '**border-image**' property has values of <source> <slice> [/ <width>] [<repeat>]
 - ◇ The <slice> value is the distance from the edge the "slices" are made and can be 1 – 4 numbers in number of pixels or percent of image
 - ◇ If you want to override the width of the border you can do so by providing legal <border-width> values after a forward slash (/)
 - ◇ The <repeat> value can be 1 – 4 values of the following
 - 'stretch' (default) - will take one copy of the sliced image and stretch it out
 - 'repeat' - uses the original image size and repeats it which may leave partial images
- 
- 'round' - repeats the image but alters the width if needed to avoid partial images

CSS 3 - Using a Border Image

Using 1 – 4 numbers for the slice provides the distance of the <top> <right> <bottom> <left> slices. If you specify the number of pixels you do not need to add the 'px' for the unit.

You could also add 'fill' to the values for the 'slice' portion and the center of the sliced image will remain as the background.

Using 1 – 4 numbers for the repeat provides the value of the <top> <right> <bottom> <left> of the image border.

Border images do not have as much support from CSS 3 browsers as other features but with IE 11 just adding support this should improve. There is also some support on mobile browsers by using the '-webkit-' browser extension.

<http://caniuse.com/#feat=border-image>

8.16 Border Image Example

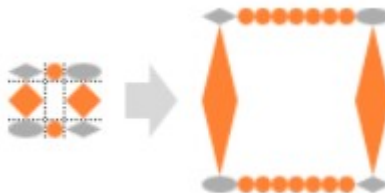
- The following example uses 9 pixels sliced in all four directions and tiles the image with the 'round' repeat value

```
border-image: url(image.png) 9 round;
```



- The following example slices 7 pixels from the top and bottom but 15 from left and right. The top and bottom border is still the 'round' repeat but the left and right use 'stretch'. This also shows overriding the 'border-width'

```
border-image: url(image.png) 7 15 / 7px 15px round stretch;
```



8.17 Summary

- The CSS box model controls many aspects of how an element is sized and positioned relative to other elements
- There are many different CSS properties related to the box model
- Using these various properties can make a page more readable by adding space where needed or providing a more professional flow and presentation of the page

Chapter 9 - Introduction to JavaScript

Objectives

Key objectives of this chapter

- JavaScript in context
- Major elements of JavaScript
- Adding JavaScript code to a web page

9.1 What JavaScript Is

- JavaScript is a programming language supported by web browsers that can be used to add interactivity to web pages
- It is often called a “scripting language” as opposed to a “programming language” but this is more because you do not need to do any kind of compilation or building of your JavaScript source code before it can execute
- Code for JavaScript is either provided directly in the HTML code of web pages or pages link to external JavaScript files with the code that are also downloaded from the server
 - ◇ This is done with a **<script>** tag that either has embedded JavaScript source code or links to another file that has the code

Notes:

Who owns the JavaScript name?

The "JavaScript" name is a trademark now owned by Oracle Corporation (US Patent and Trademark Office, copyright entry #75026640). It used be owned by Sun Microsystems.

Mark: JAVASCRIPT	
US Serial Number: 75026640	Application Filing Date: Dec. 01, 1995
US Registration Number: 2416017	Registration Date: Dec. 26, 2000
Register: Principal	
Mark Type: Trademark, Service Mark	
Status: The registration has been renewed.	
Status Date: Jan. 10, 2011	
Publication Date: May 06, 1997	Notice of Allowance Date: Jul. 29, 1997
▲ Mark Information	
▲ Goods and Services	
▲ Basis Information (Case Level)	
▼ Current Owner(s) Information	
Owner Name: ORACLE AMERICA, INC.	
Owner Address: 500 ORACLE PARKWAY REDWOOD SHORES, CALIFORNIA 94065 UNITED STATES	
Legal Entity Type: CORPORATION	State or Country Where Organized: DELAWARE

9.2 What JavaScript Is

- JavaScript executes as a “client-side” language which means it runs inside the browser after the relevant HTML and JavaScript code has been downloaded from the server
- JavaScript is consistently one of the most “popular” programming languages in use
- JavaScript was created by Brendan Eich for Netscape over a very short period of time (time, as usual, was money) with certain design flaws that are still being fixed

Notes:

Douglas Crockford, an authority on JavaScript, said that “[JavaScript] is more of a “Lisp in C's Clothing” with elements of functional programming.”

More broadly, JavaScript was influenced by a number of languages, like Java (for its syntax), Perl and Python (for strings, arrays, and regular expressions), Self (for prototypal inheritance), etc.

9.3 What JavaScript Is Not

- JavaScript is NOT related to Java
 - ◇ JavaScript was originally designed by Netscape to add scripting to their

browser. It was called 'LiveScript'.

- ◇ The Java language was getting a ton of press coverage at the time so Netscape wanted some of that attention to rub off on its scripting language, so they renamed it JavaScript

9.4 Not All JavaScripts are Created Equal ...

- JavaScript is not exactly the same in all browsers
 - ◇ Besides the original JavaScript from Netscape, Microsoft came up with a scripting language called JScript for Internet Explorer
 - ◇ With much older browsers this was more of an issue to write code that would function no matter what browser a user was using
 - ◇ In 1997 the ECMAScript specification was released which looked to standardize JavaScript code so writing ECMAScript-compatible code will help ensure compatibility with even some fairly old browsers
 - ◇ Sometimes advanced JavaScript frameworks will provide support for ensuring your code is compatible with various browsers

9.5 ECMAScript Language and Specification

- ECMAScript is the scripting language and specification standardized by Ecma International in the ECMA-262 specification and ISO/IEC 16262
- The ECMAScript spec has been implemented in such languages (as we know them) as JavaScript, JScript and ActionScript
 - ◇ You can see JavaScript, JScript, etc., as ECMAScript dialects
- Any references to the versions of JavaScript are, in fact, made to the version of the ECMAScript spec that JavaScript (more specifically, the hosting browser) implements
- Most browsers implement the ECMAScript 5 in full.
- The more recent ECMAScript 6 specification is implemented to varying degrees by each browser.

Notes:

The ECMAScript language scripts use **.es** for file extension, and are identified with the Internet media type of **application/ecmascript**

There are comments in the JavaScript community that "*ECMAScript was always an unwanted trade name that sounds like a skin disease*" [<https://mail.mozilla.org/pipermail/es-discuss/2006-October/000133.html>]

9.6 What JavaScript Can Do

- JavaScript can draw the attention of the user to certain elements of a page to highlight how the user can interact with the page
 - ◇ This could include changing the appearance of links or buttons when the mouse rolls over them to indicate they could be clicked
- JavaScript code can provide a client-side validation function to ensure that data submitted by forms on web pages is valid BEFORE the data is sent to the server
 - ◇ This can include immediate feedback of highlighting a field in red, for example, if the user moves away from that field with invalid data entered
- JavaScript could dynamically modify the elements of a page (like a form) based on data or actions the user has taken on the page
 - ◇ This could be something like changing the boxes and labels of a form to collect address or telephone information based on the user selection in a country field
- JavaScript can open new windows or popup dialogs to interact with the user instead of functioning only within a single browser window (or tab)

9.7 What JavaScript Can't Do

- JavaScript can't write data directly on a server machine
 - ◇ JavaScript can send a web request that might cause the server to store or update data but you would need a "server-side" technology to implement handling that request and modifying data on a server and this would not involve JavaScript

- The JavaScript code from a particular web site can't close a browser window or tab it hasn't opened or read data from a page that comes from another site
 - ◇ Otherwise malicious JavaScript code could close a legitimate site and open a fake site hoping you would enter sensitive information or directly read data from the page of the other site

9.8 JavaScript on the Server-side

- JavaScript is used primarily as a client-side scripting language to add interactivity to Web pages rendered in Web browsers, and is, increasingly, used on the server-side
- For example, **Node.js** is a platform for building fast, event-driven, scalable and data-intensive network applications written in JavaScript; it uses Chrome's JavaScript runtime called V8
- JavaScript can also run in the Java VM (via the Rhino technology)

Notes:

A major milestone in making JavaScript engines fast was the introduction of the Chrome browser by Google in 2008 that boasted the fastest JavaScript VM called V8. Later, V8 was open sourced and now it can be used as a stand-alone VM (not running inside a browser). In 2009, the Node.js project moved V8 to the server side.

Jeff Atwood, a co-founder of the Stack Overflow, stated that "Any application that can be written in JavaScript, will eventually be written in JavaScript" [http://en.wikipedia.org/wiki/Jeff_Atwood]

9.9 Elements of JavaScript

- JavaScript is an "object-oriented" language which includes various elements
 - ◇ **Objects** – Unique instances in memory (of the browser) that contain state and behavior
 - ◇ **Properties** – Contain values for the state of an object and can refer to simple data types or other objects
 - ◇ **Methods** – The behavior an object can perform when the method is

called on the object

- Another important aspect of JavaScript is the "Document Object Model" or "DOM"
 - ◇ This represents the elements of a web page in a hierarchy of objects and allows the JavaScript code to interact with the elements of the page

9.10 Values, Variables and Functions

- A **value** is a piece of information which can be of different types:
 - ◇ Number – any numeric value
 - ◇ String – characters inside quotation marks
 - ◇ Boolean – true or false
 - ◇ Null – empty value
 - ◇ Object – reference to an object
- A variable is a named reference to a stored value
 - ◇ Unlike some other programming languages, the type of a variable is not declared and it can reference values of different types within the script without errors
- A **function** is a set of JavaScript statements that performs a task
 - ◇ The function can be invoked, or called from other parts of the script
 - ◇ The function can declare parameters that are passed to it and return a value

Values, Variables and Functions

In other programming languages, the type of data a variable stores is declared along with the variable name. Attempting to assign a value of a different type to the variable can cause an error if the value can't be converted to the declared type of the variable. For example, the following code in JavaScript is legal but might cause errors (or unintended behavior) in other languages.

```
var answer = 42;  
answer = "Thanks for all the fish...";
```


9.11 Embedded Scripts

- JavaScript code can be embedded directly on a web page by using the **<script>** HTML tag
 - ◇ This tag can be used within the **<head>** section or **<body>** section

```
<body>
  <script>
    document.write("Hello, world!");
  </script>
</body>
```

- You can also have a **<noscript>** tag for what the browser should display if JavaScript is disabled

```
<noscript>
  <h2>This page requires JavaScript.</h2>
</noscript>
```

Embedded Scripts

In older JavaScript code you might see a 'type' attribute on the **<script>** tag indicating the type is JavaScript. This is no longer required as JavaScript is the default language in modern browsers. You might want to include this for compatibility with very old browsers.

```
<script type="text/javascript">
```

9.12 External Scripts

- Embedding lots of JavaScript code directly in a page can make it tough to read and is not reusable on multiple pages
- Linking to external files that contain JavaScript code can be a more efficient way to separate HTML and JavaScript code and allow reuse from multiple pages
- First you create a text file with a **'.js'** file extension that contains the JavaScript code
 - ◇ You do not include any **<script>** tags within the external file

```
(Saved in file 'external.js')
alert("Hello, world from external script!");
```

- Then you link to the source of the external JavaScript file using the **'src'**

attribute of the `<script>` tag on the page and the path to the location of the external file

```
<script src="external.js" />
```

- Although you can place the reference to the external script source in the `<head>` or `<body>` sections of the page, it is generally suggested to use the `<head>` section to ensure the external file is loaded before the page begins to display

External Scripts

Since the `<script>` tag in the HTML page does not have any content when linking to an external source file, you can use the `<script />` format for the tag which does not have a separate ending tag.

9.13 Browser Dialog Boxes

- There are some basic functions built-in to JavaScript that allow you to display pop-up dialog boxes from the browser

- ◇ **'alert'** allows you to display a message with an 'OK' button

```
alert("Hello from JavaScript!");
```

- ◇ **'confirm'** displays a message with 'OK' and 'Cancel' buttons and returns a boolean depending on which button the user clicks ('OK' is true, 'Cancel' is false)

```
var continue = confirm("Do you want to continue?");  
if (continue) { ...
```

- ◇ **'prompt'** displays a message along with a text box for the user to fill in a value

- The user's value will be returned by the function if the user clicks the 'OK' button
- A 'null' is returned if the user clicks the 'Cancel' button
- This function takes two parameters for the message and the default value

```
var yourName = prompt("What is your name?", "");  
if (yourName != null) { alert("Hello " + yourName); }
```

Browser Dialog Boxes

The appearance of these dialog boxes will vary somewhat by browser but their function is the same.

When using the 'prompt' function you should always check if a 'null' was returned. You could also check if the user filled in anything or if they simply clicked 'OK' and something like an empty String was returned.

9.14 What is AJAX?

- AJAX stands for Asynchronous JavaScript and XML (JSON)
- AJAX is often implemented with a combination of technologies including HTML, CSS, and JavaScript
- AJAX is an approach to developing web applications that differs from the traditional way of developing applications
 - ◇ In a traditional application, the user submits a form or clicks on a link, the browser sends a HTTP request to the server, the server replies with a fresh new HTML document that the browser renders as a new page
- AJAX differs in two main ways:
 - ◇ The browser makes a HTTP request that may or may not be due to a user action. For example, a clock application may automatically make a HTTP request every second
 - ◇ The reply from such a request does not contain a full new page. It contains information that is used to update portions of the existing page. For example, the clock application receives the current time at the server as a part of the HTTP reply and updates the time displayed in the page. The rest of the page remains as it is

9.15 Summary

- JavaScript is a very popular programming language
- Although the name implies some relationship with Java the two are actually quite different
- JavaScript contains objects with properties and methods, functions and variables

- JavaScript code can be embedded directly in a web page or linked from external source files

Chapter 10 - JavaScript Fundamentals

Objectives

Key objectives of this chapter

- JavaScript variables
- Operators and flow control statements
- Useful built-in JavaScript objects

10.1 Variables

- JavaScript variables are "containers" for storing information that will be used later in the code
 - ◇ The value stored by the variable can be used by using the variable name in an expression
- Variables have a name, are case-sensitive, and must follow certain rules
 - ◇ Variable names must begin with a letter
 - ◇ Variable names can also begin with \$ and _ (although not as common)
 - ◇ Variable names can contain numbers but can't start with them
 - ◇ Variable names can't be reserved JavaScript words
- You declare a variable with the **'var'** keyword

```
var firstName;
```

- You assign a variable a value by using the name of the variable and an equals sign

```
firstName = "Susan";  
var lastName = "Bishop";
```

Variables

You can declare and initialize a variable in one statement as in the last example above.

10.2 JavaScript Reserved Words

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

- You are also not allowed to use these key words: *Infinity*, *NaN* (not a number), and *undefined*

10.3 Dynamic Types

- JavaScript has dynamic types which means the same variable can be used as different types
 - ◇ Since the declaration of the variable doesn't tie it to a specific type, you can assign a value of a different type to a variable later without error

```
var answer = 42;  
answer = "So long and thanks for all the fish...";
```

- ◇ Although this might let you reuse variables for other data later, you must also be careful in case a variable is storing data of a type different than what you expect
- "Literals" are types of expressions in JavaScript that can be evaluated
 - ◇ String literals have quotation marks (single or double) and numeric literals have no quotes
 - ◇ Boolean literal values are 'true' or 'false' with no quotes

```
var keepGoing = true;
```

- ◇ A literal value of 'null' means no value and is often used to clear a variable of a value

```
answer = null;
```

Dynamic Types

Other languages are more strict about the data type of variables. In these languages, you must declare the type of data a variable will store with the variable declaration. Assigning a value to a variable that doesn't match this type, or could be converted to this type, would cause an error, often preventing you from even compiling and running the code.

Numeric literals in JavaScript can also be **'octal'** values that start with a zero and contain the digits 0-7, or **'hexadecimal'** that start with '0x' or '0X' and contain the digits 0-9 or the characters a-f.

Octal:	025	042
Hexadecimal:	0x2F	0xa3

10.4 JavaScript Strings

- Strings in JavaScript are created directly using string literals
- Literals can be delimited either by single (') or double (") quotes

```
var str1 = 'The value of str1'; var str2 = 'The value of str2';
```

- You concatenate string using the "+" operator
- You can use the backslash (\) character to escape some control characters as well as allowing single or double quotes inside so delimited string, e.g.

```
var str1 = "The value of str1 is \"The value of str1 is ...\" ";
```

10.5 Escaping Control Characters

- JavaScript follows the C / Java special character escaping notation:

\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace

10.6 What is False in JavaScript?

- JavaScript treats the following values as the **false** boolean value:
 - ◇ Numbers: *0*, *NaN*
 - Any non-zero numbers evaluate to neither *true* nor *false*
 - ◇ Strings: Empty strings and strings containing whitespace characters, e.g. `' '`, or `" "`, or `'\t'`, etc.
 - Any non-empty strings evaluate to neither *true* nor *false*
 - ◇ Boolean literal: *false*
 - ◇ Keywords: *undefined* and *null* (more on those a bit later ...)

10.7 Numbers

- JavaScript has only one type of numbers which are double precision floating-point numbers (which contain IEEE 64 bit values)
 - ◇ JavaScript stores numbers in 64 bits as follows (as per the IEEE 754 Floating Point Standard): the fraction (mantissa) is stored in bits 0 to 51, and the exponent is stored in bits 52 to 62; bit 63 is used for the sign
 - ◇ This way you can store values in a range of about $\pm 10^{-308} \dots 10^{308}$
- In addition to regular notation for numbers, like `var a = 123; var b = 45.99;` you can also use scientific (exponent) notation, e.g. `var e = 1.23e2;` (will evaluate to 123)

Notes:

The following table summarizes the bit allocation in JavaScript numbers

Bit No	Size	Field Name
63	1 bit	Sign (S)
52-62	11 bits	Exponent (E)
0-51	52 bits	Mantissa (M)

10.8 The Number Object

- JavaScript uses a system object called *Number* as a holder of some useful constants, e.g.
 - ◇ **Number.MAX_VALUE**
 - 1.7976931348623157e+308
 - ◇ **Number.POSITIVE_INFINITY**
 - it represents infinity (*Infinity*); returned on math operation overflow
 - E.g. `var x = 1/0;` will result in x getting the "Infinity" reserved numeric value, but not a run-time exception
 - ◇ **Number.NEGATIVE_INFINITY**
 - It represents negative infinity (*-Infinity*); returned on math operation overflow
- The '**Number**' object is also a wrapper for numeric values with some methods to convert between various forms (decimal and exponential)

```
var x = new Number(123.45);  
alert(x.toExponential());
```

10.9 Not A Number (NaN) Reserved Keyword

- *NaN* is a special keyword in JavaScript referring to a value or evaluation result which is "Not a Number"
- JavaScript offers a system function *isNaN* for checking if the value passed to it as a parameter gets evaluated to the *NaN* value
 - ◇ `isNaN("Sun Microsystems");` // returns **true**
 - ◇ `isNaN(123);` // returns **false**

10.10 JavaScript Objects

- JavaScript variables can also refer to objects
- There are two ways to create a direct object instance:
 - ◇ Using the Object constructor and then adding individual properties

```
person = new Object();  
person.firstname = "John";  
person.lastname = "Doe";  
person.age = 50;  
person.eyecolor = "blue";
```

- ◇ Using an object literal with curly braces

```
person = {firstname:"John", lastname:"Doe", age:50,  
eyecolor:"blue"};
```

- Object property values can be accessed using a 'dot' notation

```
alert(person.firstName);
```

10.11 Operators

- JavaScript uses many of the same operators as other programming languages

- ◇ Arithmetic: +, -, *, /, % (modulus, division remainder)

- ◇ Increment/Decrement: ++, --

- *Before* an operand, modifies a value before being used in the rest of the expression

- *After* an operand, the original value is used before being modified

- ◇ Assignment: =, +=, -=, *=, /=, %=

- ◇ String concatenation with '+'

- ◇ Logical: && (and), || (or), ! (not)

- ◇ Comparison: <, <=, >, >=, ==, !=

- ◇ "Conditional" or "Ternary" operator: (condition)?(true value):(false value)

- JavaScript also contains a unique '===' operator (triple equals sign) that tests if two operands are identical (equal value and equal type)

- ◇ And also '!==' for "not identical" (different value or different type)

```
x = 5;
```

```
x == "5";    // true
x === "5";   // false
```

Operators

Since JavaScript variables are not declared with a type, the '===' identity operator is useful when checking that the type of two values also matches. Values can be considered "equal" with the '==' operator but not "identical" with the '===' operator.

There are also operators for various bitwise operations but these are not as common.

Shift bits: <<, >>, >>>

Bitwise comparison: & (bitwise AND), | (bitwise OR), ^ (bitwise XOR)

10.12 Primitive Values vs Objects

- JavaScript categorizes values into two groups (this is, to some extent, Java language influence):
 - ◇ The primitive values:
 - *booleans, strings, numbers, null, and undefined*
 - ◇ All other values are treated as objects (including arrays, dates, regex and functions)

10.13 Primitive Values vs Objects

- Objects differ from primitives in one substantial aspect: objects have their own identity, primitives have their identity expressed in their value

```
objA = { f : 99.99};
```

```
objB = { f : 99.99};
```

```
objA == objB; // false, same as with ===
```

```
objA.f == objB.f // true, the property f in both objects have the same value
```

```
objA.f === objB.f // also true (same value and same type)
```

10.14 Flow Control

- Similar to other programming languages, JavaScript has many control statements that are used to alter the flow of a program based on certain conditions
 - ◇ Decision statements
 - if
 - if...else
 - switch
 - ◇ Iteration statements
 - for
 - for / in
 - while
 - do...while

10.15 'if' Statement

- | | |
|---|--|
| <ul style="list-style-type: none">■ Most basic control flow statement■ Executes a statement if a condition is met■ The condition must be a boolean expression■ May execute a single statement or a statement block■ It is possible to reassign variable values within the statement block | <pre>if (condition) statement; if (condition) { statement1; statement2; statement3; } var number = -4; if (number < 0) { alert("negative number"); number = 0; }</pre> |
|---|--|

'if' Statement

It is legal to write an if statement on a single line:

```
if (number % 2 != 0) alert("odd number.");
```

Although this is legal, it is often best to put the action statement on a separate line from the condition being tested, as in the first example on the slide.

The condition of an if statement may be a more complex expression, as long as the entire expression evaluates to a single boolean (true/false) value. For example, this statement:

```
if (numOrders != 0 && daysInMonth / numOrders > 4)
    alert("Sales are up from last month.");
```

is legal and desirable to avoid division by zero.

10.16 'if...else' Statement

- Extends basic if statement
 - Executes **else** clause if condition of if statement is false
 - May be a single statement or a statement block
- ```
if (condition)
 statement;
else
 statement;

if (condition) {
 statement1;
 statement2;
} else {
 statement1;
 statement2;
}
```

## 'if...else' Statement

Several if...else statements can be chained together by using another if statement as the body of the else clause. This structure evaluates conditions until one is true, executes the corresponding action statement, and then skips the rest of the if...else statements in the chain. The following statements assign a letter grade based on an integer score:

```
var score = 73;
var grade;
if (score >= 90) {
 grade = 'A';
} else if (score >= 80) {
 grade = 'B';
}
```

```
} else if (score >= 70) {
 grade = 'C';
} else if (score >= 60) {
 grade = 'D';
} else {
 grade = 'F';
}
```

This code assigns a letter grade of 'C' even though the score would also pass the last condition. This is because the "score >= 60" condition is part of the else clause of the first condition to pass and is never even evaluated.

Care must be taken with this type of expression that logical errors are not introduced. For example, using the following for the beginning of the if...else if statement would assign a letter grade of 'D' even to those who deserved an 'A', 'B' or 'C' because those scores would satisfy the first condition:

```
if (score >= 60) {
 grade = 'D';
```

### 10.17 'switch' Statement

- Selects from multiple choices based on an expression
- The value of the expression determines which case is executed
- The **default** case handles values not matched by other cases
- Use **break** to prevent the code from running into the next case automatically

```
switch (expression) {
 case n1:
 // case 1 statements
 break;
 case n2:
 // case 2 statements
 break;
 ...
 case n:
 // case n statements
 break;
 default:
 // default statements
 break;
}
```

#### 'switch' Statement

The statements which execute during the course of a particular case do not have to be enclosed in a separate pair of brackets.

The functionality of a switch statement could also be matched by using an if statement. Using a switch statement is often easier because the expression only needs to be written out once and testing against each case is implied.

The following prints out what grade someone received:

```
var grade = 'D';
switch (grade) {
 case 'A':
 alert("You received an A"); break;
 case 'B':
 alert("You received a B"); break;
 case 'C':
 alert("You received a C"); break;
 case 'D':
 alert("You received a D"); break;
 case 'F':
 alert("You received an F"); break;
}
```

### 10.18 'for' Loop

- Repeats a set of statements a certain number of times

```
for (var i = 0; i < 10; i++)
{
 // executed 10 times
 ...
}
```
- The control of the loop is defined within parenthesis after the for keyword
- The initialization, condition, and iteration parts of the control are separated by semi-colons

```
for (var i = 0, j = 10;
 i < j; i++) {
 // executed 10 times
 ...
}
```
- More than one statement per control segment can be included separated by commas

### 'for' Loop

The initialization part of the control is executed once before any other parts of the for loop are executed. This segment is often used to declare and initialize a variable which will be used for counting the loop iterations. The condition part of the control is a boolean expression which must be true to execute the body of the for loop. This condition is checked before every iteration of the loop including the first iteration. The iteration part of the control is executed after every successful iteration of the loop. It is executed before the condition expression is evaluated again. The iteration segment is often used to increment or decrement variables.

Sometimes one for loop is nested within another. This is often done with multidimensional arrays. Each loop has a unique index and a complete set of iterations of the inner loop occurs for each iteration of the outer loop:

```
for (var i = 0; i < 10; i++) {
 for (var j = 0; j < 10; j++) {
 // These statements executed 100 times total
 ...
 }
}
```

An infinite for loop is possible but is generally avoided and must contain a break statement to exit:

```
for (;;) {
 ...
 if (timeToBreak) {
 break;
 }
}
```

### 10.19 'for / in' Loop

- The 'for/in' loop is a unique loop type for JavaScript
- This loops through the properties of an object, assigning each property name to a temporary variable
  - ◇ This could then be used to access the value of that property

```
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};

for (var x in person)
{
 txt = txt + person[x];
}
// txt is 'JohnDoe25'
```



## 10.20 'while' Loop

- Repeats a set of statements while a condition is **true**

```
while (expression) {
 // These statements repeated
 ...
}
```
- The condition expression must return a **boolean** value
- If the condition is initially **false**, the loop block will not be executed at all

```
var number = 1.2;
while (number < 5.0) {
 number *= 2;
 ...
}
```
- Make sure something changes in the body of the loop to eventually let the condition be false

### 'while' Loop

The while loop is used to repeat a set of statements when the number of iterations is not as well known as when using a for loop. Unlike a for loop, some variable or condition must change during the course of execution of the loop for the loop to eventually terminate.

An infinite while loop is possible but generally avoided. If declared, a break statement must be used to terminate the loop:

```
while (true) {
 ...
 if (timeToBreak) {
 break;
 }
}
```

## 10.21 'do...while' Loop

- Similar to **while** loop
  - Condition is tested at the end of the loop statement
  - Semi-colon at the end of the loop
  - Loop statement is always executed at least once
- ```
do {  
    // These statements  
    repeated  
    // at least once  
    ...  
} while (expression);  
  
var number = 6;  
do {  
    // this loop stills  
    // executes once  
    ...  
} while (number < 5);
```

'do...while' Loop

One common problem encountered with loops is termed the "fencepost" problem. If you imagine constructing a fence, you always need one more fencepost than sections of a fence. For instance, 5 fence sections requires 6 posts and 10 fence sections require 11 posts. Often this means that some part of a process must be performed outside of the loop to ensure the extra "post" is placed. Each type of loop can be used in some way to solve the fencepost problem. The following code uses a do...while loop to print sections and posts of a fence:

```
var count = 0;  
var numSections = 5;  
var display = "";  
do {  
    display = display + "|-";  
    count++;  
} while (count < numSections);  
display = display + "|";  
alert(display);
```

Notice inside the loop that there is a statement to add a post and a section of fence. Outside the loop there is a single statement to add the final post. This loop always prints out at least one fence section (|-|). Use of a do...while loop makes sense here because other loops might print out only the single final post (|) as a minimum.

10.22 Break and Continue

- The **'break'** and **'continue'** statements can be used to alter the execution of loops

- ◊ The **break** statement breaks the loop and continues executing the code after the loop (if any)

```
for (i = 0; i < 10; i++)
{
    if (i == 3) {
        break; // stops the entire loop after 0, 1, 2
    }
    x = x + "The number is " + i + "<br>";
}
```

- ◊ The **continue** statement breaks one iteration (in the loop) and continues with the next iteration in the loop

```
for (i = 0; i < 10; i++)
{
    if (i == 3) {
        continue; // skips over 3 for 0, 1, 2, 4, ...
    }
    x = x + "The number is " + i + "<br>";
}
```

Break and Continue

Although certainly not required, often you will use 'break' or 'continue' in a loop with a conditional 'if' so that they only affect the execution of the loop under certain conditions.

10.23 Labeled Statements

- Statements may be labeled with an identifier
- Most often used with control statements to create a more complex structure
- Defined by placing an identifier and colon before the statement to be labeled

```
OuterLoop:
for (var i = 0; i < 10; i++) {
    // other statements
}
```

- Referred to by other statements

```
break OuterLoop;
```

Labeled Statements

When used without a label, the `break` statement and a few similar control statements, we'll see later, only affect the innermost switch statement or loop. To affect something outside of this, they must refer to a pre-defined label. For example, the following code uses an 'OuterLoop' statement label to break out of both nested loops at once:

```
var foundIt = false;           // set to true when we wish to exit
OuterLoop:
for (var i = 0; i < 10; i++) {
    for (var j = 0; j < 10; j++) {
        // do some searching to find if we found something
        if (foundIt) {
            alert("We found it!");
            break OuterLoop;
        }
    }
}
```

This code would stop both loops once the item was found so the inner block of statements may not be executed for a full 100 iterations.

10.24 The undefined and null Keywords

- JavaScript distinguishes between two situations when some information is missing
 - ◇ A variable has been declared but not initialized (it does not have value), for example:
 - `var myVar; alert (myVar);` // will show *undefined*
 - `var myObj = {}; alert (myObj.myVar);` // will show *undefined* (the *myVar* property has been defined by way of attaching it to the object variable via the dot notation, but not properly initialized)
 - Your function gets fewer arguments than it declares
 - ◇ A "no object" situation, is treated as a *null*
 - It is used as the last element in the prototype chain and in some other arcane situations

- **Note:** JavaScript uses *undefined* in most cases of missing information, even in situations where you would expect a *null*

10.25 Checking for undefined and null

- You can use either JavaScript idiom when checking for missing information:
 - ◇ `if (myVar === undefined || myVar === null) { ... }`
 - ◇ `if (!x) { ... }`
- You can use the boolean not operator as both *undefined* and *null* are treated by JavaScript as **false**

10.26 Checking Types with typeof Operator

- The *typeof* operator is used in JavaScript to find out the type of primitives and objects
- Here are some of the examples:

```
typeof false; // "boolean"

typeof 123; // "number"

typeof "Yo JavaScript!"; // "string"

typeof new Object(); // "object"

typeof {}; // "object"
```

10.27 Date Object

- JavaScript has a built-in '**Date**' object to work with date and time values
- Constructing a new object without any parameters is the current date and time

```
var now = new Date();
```

- There are three other ways to initialize a Date object:
 - ◇ With the last approach, not specifying a value causes a '0' to be used

```
new Date(milliseconds) // milliseconds since 1970/01/01
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds,
        milliseconds)
```

```
var birthday = new Date(1970, 3, 15);
```

- There are various methods like 'getFullYear', 'getDate', 'setHours', etc.

```
var age = now.getFullYear() - birthday.getFullYear();
```

- A later version of JavaScript also added methods for UTC (Coordinated Universal Time)

Date Object

There is a 'getYear' method on the Date object. This returns a 2 digit value for years 1900-1999 and a 4 digit value for 2000 and later. The method 'getFullYear' always returns 4 digits.

10.28 Document Object

- The '**Document**' object is part of the Document Object Model (DOM) API and provides access to the elements of the page
 - ◇ The DOM API is very extensive and is only introduced at a high level here
- There is a '**document**' variable that is automatically initialized as the reference to the current page
- One of the most common ways to access elements in the page is by id using the 'getElementById' method
- You can also change the content of an element with the 'innerHTML' property

```
document.getElementById("date").innerHTML = new Date();
```

- You can also alter the structure of the page with various methods to add and delete elements like 'createElement', 'appendChild', 'removeChild', and 'replaceChild'
- There is also a 'write' method on the Document object that could be used to directly write raw HTML, but access through the DOM API is preferred

```
document.write("It is: " + new Date());
```

Document Object

DOM is a very complete API used to access the elements on the page. Although this is just a brief introduction, there is certainly much more that can be used with DOM and makes possible some of the more modern rich internet applications that rely on JavaScript code making changes to the page.

10.29 Other Useful Objects

- The **'String'** object has useful methods for accessing and manipulating the characters in a String
 - ◇ All String literals (declared with just quotes) are objects even without the 'new' keyword

```
var name = "Susan"; // same as new String("Susan")
var secondChar = name.charAt(1); // position starts at 0
var upper = name.toUpperCase();
```

- The **'Math'** object has various useful math functions like 'random', 'sin', 'cos', 'tan', 'ceil', 'floor', 'sqrt', 'abs' (absolute value), etc.

```
var random = Math.floor(Math.random() * 11);
```

10.30 Browser Object Detection

- One issue with writing JavaScript is dealing with the potential that an older browser will not recognize the code you use
 - ◇ This is becoming less of an issue but still may be a concern in some situations
- One way to deal with this is to use "object detection" which tests to see if an object, or a method on that object, exists before calling code that will use the object
 - ◇ This allows you to provide an alternate path in the code if the browser doesn't understand the object or method you are trying to use
- To do this, you use an 'if' statement where the condition is the name of the object or method you want to detect
 - ◇ If a browser doesn't understand an object or method, it will return **false**
 - ◇ You only use a method name and don't pass parameters to a method, as you are not calling the method in the conditional test

```
if (now.toLocaleDateString) {  
    // This could cause errors in older browsers  
    var date = now.toLocaleDateString();  
} else {  
    alert("Your browser does not give a locale date/time");  
}
```

Browser Object Detection

Although it is possible to try and detect the type of browser directly, this is very tedious and error-prone. Object detection is a very easy way to accomplish the same thing, avoiding calling code that a browser might not understand.

You can use the Modernizr JavaScript library (<http://modernizr.com>) for browser feature detection (HTML, CSS, JavaScript)

10.31 The eval Function

- The *eval()* function compiles and executes an argument which can be a JavaScript expression, variable, statement, or sequence of statements
- May pose some security threats due to possible harmful code injection attacks
- Example:

```
eval ("1 + 999"); // will return 1000
```

10.32 Enforcing Strict Mode

- By default, JavaScript runs in the so-called "sloppy programming mode" where JavaScript interpreter is very lenient to user code ambiguities and makes a lot of assumptions (sometimes very wrong) about the programmer's intent
- You can enable Strict Mode to make JavaScript more stringent and enabling more warnings
- To enable Strict Mode, type the following:

```
'use strict';  
as the first line in your JavaScript file or a <script> tag
```


10.33 Summary

- JavaScript variables are not declared with a type and can hold a value of any type
- JavaScript uses operators that are similar to other programming languages
- The flow control statements in JavaScript are also similar to other languages, although the 'for/in' statement is unique
- There are several built-in object types in JavaScript that can provide useful utility functions
- Using "browser object detection" to determine if a browser recognizes an object, before trying to use it in the code, makes it easier to write cross-browser compatible code than trying to detect the exact type of browser

Chapter 11 - JavaScript Functions

Objectives

Key objectives of this chapter

- Declaring JavaScript functions
- Function arguments and return values
- Local and global variables

11.1 Functions Defined

- A JavaScript function is a named block of code that can be called from other JavaScript code by using the function name
- Functions can optionally be defined to take arguments with values passed in by the code calling the function
- Functions can optionally return a value from the function that can be used by the code that called the function
- In JavaScript, functions are "first-class objects" that are treated as any other JavaScript objects
 - ◇ Functions can be passed to and returned from other functions (as an anonymous or *lambda* function); more on that in a later module

11.2 Declaring Functions

- Function declarations have specific syntax in JavaScript although it is somewhat simpler than other programming languages
- Functions are declared by using the '**function**' keyword, the name of the function, a set of parenthesis for arguments (possibly empty), and a set of curly brackets '{ }' around the statements that make up the body of the function

```
function myFunction() {  
    // function body  
    alert("Hello from inside the function!");  
} // end of function declaration
```

- ◇ Note: there

- To call the function you simply use the function name along with any needed arguments in parenthesis
 - ◇ The parenthesis still need to be there even if empty

```
// call the function  
myFunction();
```

Declaring Functions

The **'function'** keyword for the function declaration is case-sensitive and must always be lowercase.

11.3 Function Arguments

- If a function is declared to accept arguments, the names of the arguments are given in the parenthesis of the function declaration and separated by commas
 - ◇ These become variables available in the function whose value is initialized by whatever value is passed in by the code calling the function
 - ◇ Just like regular JavaScript variables, function arguments do not have a type declared for them and can accept any type

```
function sayHello(firstName, lastName) {  
    alert("Hello " + firstName + " " + lastName + "!");  
}
```

11.4 More on Function Arguments

- Missing arguments are treated in the body of the function as undefined
- Objects (covered later in the course) are passed to functions by reference
- When calling a function and passing in argument values, be careful the order the arguments are passed in to the function and the type that might be used

```
sayHello("Bob", "Smith");  
sayHello(4, 12); // still legal, probably not as intended
```

Function Arguments

Since the declaration of a function does not have a type for the function arguments, it may be easy to pass in arguments of a type that might cause unintended behavior of the function.

11.5 Return Values

- Functions can return a single value back to where the function was called
 - ◇ This is done by using the '**return**' statement along with the value to be returned in the body of the function
 - ◇ There is nothing in the function declaration that indicates if a function returns a value, unlike some other programming languages
 - ◇ When a '**return**' statement is executed, the body of the function stops executing and returns the value

```
function multiply(a, b) {  
    return a * b;  
}
```

- The returned value is often stored in a variable where the function was called from

```
var result = multiply(2, 14);  
// value of result variable is 28 after calling function
```

Return Values

Since nothing in the declaration of a function indicates if a function returns a value or what type it returns, you must make sure you understand what value is returned from a function.

It might be common to have 'return' statements inside conditional 'if..else' blocks to return different values under different conditions. You could also have a 'return' statement inside an 'if' block early in the code to return a value before later code in the function body executes.

11.6 Multiple Return Values in ECMAScript 6

- The upcoming release of ECMAScript 6 (ES6 "Harmony") plans to add multiple function return values
- The proposed syntax is:
 - ◇ `return {var1, var2};` for a two-value return

- ◊ You can then process the multi-value returns as objects
- Don't confuse this proposed syntax with returning an object in ECMAScript 5-compliant JavaScript engines, e.g.:

```
function foo () {  
    return {i : 123, j: "OK"};  
}
```

would return an Object {i: 123, j: "OK"}

11.7 Optional Default Parameter Values

- Current version of JavaScript (ECMAScript 5) does not support optional function arguments or parameters
- **Note:** ECMAScript 6 spec includes provisions for default values
 - ◊ The proposed syntax is:

```
function f (x1, x2="Some optional value") {...}
```

11.8 Emulating Optional Default Parameter Values

- You can use the following pattern to emulate default values for function's arguments using the logical or operator: ||

```
function foo(x) {  
    x = x || -1;  
// if x is missing or is undefined, it will be assigned the default value of -1;  
// Now you can use x's real value, of its default value of -1
```

11.9 Anonymous Function Expressions

- JavaScript has another way to define a function called **anonymous function expressions**
- An anonymous function expression produces a function object that is assigned to a variable, which becomes the name of the function
 - ◊ For example:

```
var add = function (x, y) { return x + y };  
add(6, 4); //will return 10
```

11.10 Functions as a Way to Create Private Scope

- A JavaScript function, when called, creates a new scope or execution context for your code running inside the function
- Variables defined within a function are only accessible from inside it and are not visible to the outside context
- So, in JavaScript, functions offer you a mechanism to establish a private scope

11.11 Linking Functions to Page Elements

- Functions can be called when certain events happen on page elements like clicking a button
 - ◇ This is done by having JavaScript code to call the function tied to an HTML element event attribute

```
<html><head><script>
function myFunction()
{
    alert("Hello World!");
}
</script></head>
<body>
<button onclick="myFunction()">Try it</button>
</body></html>
```

- Some of the most common events JavaScript functions are linked to are:
 - ◇ The 'onload' event on the 'window' object when the page is loaded
 - ◇ The 'onmousedown', 'onmouseup', or 'onclick' events of a button
 - ◇ The 'onchange' event of form input elements to trigger validation code
 - ◇ The 'onmouseover' and 'onmouseout' events of various elements

Linking Functions to Page Elements

This is not a full description of the various events that page elements can raise but simply some context on one popular use of functions.

The `onmousedown`, `onmouseup`, and `onclick` events are all parts of a mouse-click. First when a mouse-button is clicked, the `onmousedown` event is triggered, then, when the mouse-button is released, the `onmouseup` event is triggered, finally, when the mouse-click is completed, the `onclick` event is triggered.

11.12 Local and Global Variables

- Any variable that is "local" to the function will go out of scope when the function is finished executing
 - ◇ This means the value will no longer be accessible
- Function arguments are automatically local variables
- Variables declared with the **'var'** keyword inside the body of the function are also local variables
- Any variables that are declared within a function and initialized without the **'var'** keyword are GLOBAL
 - ◇ These can be modified from within the function and still accessed once the function is finished executing
- Variables declared outside of a function are always global

Local and Global Variables

In general variables should be declared with the **'var'** keyword so they are local unless you require the behavior of having a global variable. Simply being lazy and leaving off the **'var'** keyword on all variable declarations would cause the browser to retain more data in memory than is likely required and might cause some odd behavior.

So, if you don't use **var** to declare variables, you may encounter the hard to chase bugs. For, example, let's say you have two functions: *one* and *two*, that both reference variable *i*.

```
function one () {  
    i = 10;  
    alert ("i =" + i);  
}  
function two () {  
    alert ("i =" + i);  
}
```

Now if you invoke function *one* before *two*, everything works as expected and you will have two alert dialogs popping up displaying 10 as the value *i* (the *two* function will see *i* as *i* has global scope by not being declared with the **var** keyword). But if you change the order of function calls: you call *two* before *one*, you will encounter the *Uncaught ReferenceError: i is not defined*, as *i* has not been brought

into existence yet and properly initialized.

11.13 Local and Global Variables

- The example code below shows global and local variables
 - ◇ '**localVariable**' is not accessible outside of the function

```
<script>
var globalVar = 45;    // global variable

function myFunction( localParameter ) {
    var localVariable = 12;    // local to function
    alsoGlobal = 15;    // global variable within function
    var answer = localParameter + localVariable +
        alsoGlobal;
    alert("Answer: " + answer);
}

myFunction(34)    // shows alert with '61'
alert(alsoGlobal);    // legal because the variable is global
alert(localVariable);    // doesn't work outside function

</script>
```

11.14 Declaring Object Methods

- In JavaScript a "method" is just the terminology for a function that is declared on a specific object and is not a "global" function declared at the root of the script
 - ◇ The '**function**' keyword is still used and the difference is the way it is declared
 - ◇ The method parameters (if any) are defined immediately after the '**function**' keyword
- A method is a named property of an object that is initialized as a function
 - ◇ Within a method for an object, other properties of the object can be accessed by using the '**this**' keyword

```
person = new Object();
person.firstName = "John";
person.setFirstName = function(newName) {
    this.firstName = newName;
};
```

- You must call the method by using the 'dot' notation between the name of the object and the method name along with any arguments

```
person.setFirstName("Jonathan");
```

11.15 The arguments Parameter

- When a function is invoked, it is supplied a special parameter called **arguments** accessible inside the function
- It is an array-like structure which has the **length** parameter containing the number of parameters actually passed to the function
- *arguments* can be used to access parameters using indexes
- Missing parameters or any extra parameters are treated as *undefined*

11.16 Example of Using arguments Parameter

- In code below we are using the *console* object supported by most modern JavaScript engines to output content to the JavaScript browser console

```
function foo (x, y, z) {
    console.log ("The number of arguments actually passed in : "
+ arguments.length);
    console.log (arguments[0]);
    console.log (arguments[1]);
    console.log (arguments[2]);
    console.log (arguments[3]); // extra parameter !
}
```

```
foo (777, 'a');
```

OUTPUT:

```
The number of arguments actually passed in : 2
777           // arguments[0]
a             // arguments[1]
undefined     // arguments[2]
undefined     // arguments[3]
```

11.17 Summary

- JavaScript functions are declared with the '**function**' keyword
- Function arguments are declared within parenthesis but do not have a declared type similar to "normal" JavaScript variables
- Functions can return values but nothing about the function declaration indicates if it does
- Function arguments and variables declared within the function with the '**var**' keyword are local to the function and not accessible outside the function
- Objects can have "methods" which are functions declared as object properties

Chapter 12 - JavaScript Arrays

Objectives

Key objectives of this chapter

- Define JavaScript arrays
- How to perform various actions with arrays
- Useful methods for arrays

12.1 Arrays Defined

- An array is an object that is implemented as a map (dictionary / hash table) where indexes are numbers converted into strings which are used as keys to retrieve the values
 - ◇ In contrast, in many programming languages, arrays hold a linear sequence of data of the same type
- An array can hold many values under a single name, and you can access the values by referring to an index number
 - ◇ JavaScript arrays use a '0' as the index of the first element in the array
- Just like JavaScript variables that store a single value, JavaScript arrays are not declared with a single data type
 - ◇ This means that different elements of the array can hold values of different types
- Arrays are efficient for holding sparse data (where some elements are missing)

12.2 Creating an Array

- There are three different ways to create and initialize an array
 - ◇ Create empty array and initialize each element

```
var myCars = new Array();  
myCars[0] = "Saab";  
myCars[1] = "Volvo";  
myCars[2] = "BMW";
```

- ◇ Supply the initial values of the array elements as parameters to the constructor

```
var myCars = new Array("Saab", "Volvo", "BMW");
```

- ◇ Use square brackets with an array literal

```
var myCars = ["Saab", "Volvo", "BMW"];
```

- You can also pass a single number to the Array constructor to indicate how many elements to create the array with
 - ◇ If you do this, the value of the elements will start as 'undefined'
 - ◇ Unlike other languages, you can add elements to an array at any time and do not need to work with a "fixed" number of elements

```
var myCars = new Array(3);
```

Creating an Array

All of the first three examples above create the same array.

The Array() constructor function behaves as follows. If there is only one parameter and it is a number, the constructor creates an array of that many elements. The elements do not point to any objects. If there are multiple parameters present or the only parameter is not a number, the constructor creates an array and adds the parameters as elements.

Since you can add array elements any time you need, the constructor that takes a parameter for the number of elements to create is not required. It is also not often as useful since it will leave array elements uninitialized and it might be better to simply add and initialize elements as needed.

12.3 The length Array Member

- An array object has a '**length**' property that (surprise !) does not return the number of elements in the array (as is the case in many other programming languages)
- The *length* property contains a number which is the highest integer subscript in the array + 1
- Example of using the *length* property:

```
var a = [];  
a[4] = 4;  
a[1001] = 'Whatever ..';  
a.length; // 1002 = 1001 + 1
```

12.4 Traversing an Array

- You can use the usual **for...in** syntax:

```
var array1 = ["John Smith", "Jimmy Dean"];
for (i in array1) {
    log("Item " + i + ": " + array1[i]);
}
```

- Or use the **length** property:

```
for (var i = 0; i < array1.length; ++i) {
    log("Item " + i + ": " + array1[i]);
}
```

Traversing an Array

The variable that is used as the array index is often compared with the '<' operator to the array length. This will stop traversing the array without trying to use the value of the array length as an array index which would cause problems.

12.5 Appending to an Array

- The correct way to append an element is to use the **push()** method

```
var a2 = ["John Smith", "Jimmy Dean", "Meg Ryan"];
a2.push("Iron Man", "Jackie Chan");
//Adds two elements to a2 and returns the current length.
```

- Alternatively, just add a new property to the object with the right index value

```
a2[3] = "Iron Man";
```

◊ Generally, the formula is: `myArray[myArray.length] = ...;`

- If you add a new element with an index greater than what the next index would normally be, any elements "between" the current elements and the element being added will be uninitialized

```
a2[5] = "Spider Man";
// a2.length will return 6. However,
// a2[4] will return undefined.
```

Appending to an Array

Using the 'push' method is recommended because any change in the length of the array could cause code that uses literal indices to add an element to behave unexpectedly.

12.6 Deleting Elements

- You can use the **delete** keyword to remove an element, but that doesn't change the length of the array
 - ◇ This simply removes the value stored by that element

```
var a2 = ["John Smith", "Jimmy Dean", "Meg Ryan"];
delete a2[2];
// a2.length still returns 3 but a2[2] returns undefined
```

- The correct way to delete elements is to use the **splice()** method of the array object
 - ◇ splice(index to start deleting from, number of elements to delete)

```
var a2 = ["John Smith", "Jimmy Dean", "Meg Ryan",
         "Spider Man"];
a2.splice(1, 2);
// This will delete "Jimmy Dean" and "Meg Ryan"
// a2.length will now return 2
```

12.7 Inserting Elements

- The **splice()** method is also used to insert elements
 - ◇ splice(index to start, number of elements to delete, list of new elements to insert)

```
var a2 = ["John Smith", "Jimmy Dean", "Meg Ryan",
         "Spider Man"];
a2.splice(1, 2, "Iron Man", "Hulk", "Jackie Chan");
// This will delete "Jimmy Dean" and "Meg Ryan" and replace
// them with "Iron Man", "Hulk" and "Jackie Chan".
```

- ◇ To insert new elements without deleting existing elements of the array, use **0** for the number of elements to delete (the second splice's argument)


```
a2.splice(1, 0, "Iron Man", "Hulk", "Jackie Chan");

// the above command will produce a 6-element array
// (inserted elements are shown in bold):
["John Smith", "Iron Man", "Hulk", "Jackie Chan", "Jimmy
Dean", "Meg Ryan", "Spider Man"]
```

12.8 Other Array Methods

```
var a1=["Jackie Chan", "Iron Man"];
var a2=["John Smith","Jimmy Dean","Meg Ryan","Spider Man"];
  ■ pop() – Removes the last element and returns it
a2.pop() // Returns "Spider Man" and deletes that element
  ■ shift() – Removes the first element and returns it
a2.shift() // Removes "John Smith" and returns it
  ■ unshift() - Adds new elements to the beginning of the array and returns
    the new length
newLength = a1.unshift("Meg Ryan", "Spider Man");
// Adds new elements to the start of the array and returns
4
  ■ sort() - Sorts the elements of an array, taking a sorting function as an
    optional parameter
    ◇ Default sort order is ascending alphabetical
a4 = [45, 12, 26, -34];
a4.sort(function(a,b) {return a-b}); // [-34,12,26,45]
  ■ reverse() - Reverses the elements of an array
  ■ indexOf() & lastIndexOf() - Searches from the start or end of the array for
    the given item and returns the position
    ◇ An optional second parameter can give the position to start from
```

12.9 Other Array Methods

```
var a1=["Jackie Chan", "Iron Man"];
var a2=["John Smith","Jimmy Dean","Meg Ryan","Spider Man"];
  ■ concat(array or elements) – Creates a new array by adding elements
```

from another array

```
//Contents of a1 and a2 are combined and returned
a3 = a1.concat(a2);
//Elements of a1 and two new elements are combined and
returned
a3 = a1.concat("Meg Ryan", "Spider Man");
```

- **slice(begin, stop)** – Returns a subset of the array starting with begin index and up to, but not including the stop index

```
a3 = a2.slice(1, 3)
// Returns an array containing "Jimmy Dean" and "Meg Ryan"
```

- **toString()** - Converts an array to a comma-separated String of the elements and returns the result
- **join()** - Joins the elements of an array into a String with an optional parameter for the separator between elements

12.10 Accessing Objects as Arrays

- You can access the properties of a JavaScript object with array syntax instead of the 'dot' syntax by using the property name as the array index

```
person = {firstname:"John", lastname:"Doe", age:50,
eyecolor:"blue"};
alert( person["firstname"] );
```

- This is used in the 'for..in' loop where the temporary variable in the loop actually loops through the names of the object properties

```
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};

for (var x in person)
{
    txt = txt + person[x];
}
// txt is 'JohnDoe25'
```

12.11 Summary

- Arrays in JavaScript do not have a declared type or size and are very

dynamic

- An array has a 'length' property that can be important when working with the array
- There are various array methods that perform various actions on the array

Chapter 13 - Advanced Objects and Functionality in JavaScript

Objectives

Key objectives of this chapter

- Using JavaScript as an Object Oriented language
- Object Constructor functions
- JavaScript closures
- Prototype property
- Techniques for establishing inheritance in objects

13.1 Basic Objects

- A basic object is a collection of properties. Each property has a name and a value. In a way, an object is like a dictionary or a hash map
- A property can be of any type (a string, date, time, number or a function)
- There are three ways to create an object in JavaScript:
 - ◇ Using an object literal
 - ◇ Using a constructor object
 - ◇ Using the `Object.create()` method (reviewed after we cover inheritance)
- Object literal:

```
var person = {firstName: "John", lastName: "Smith"};
var employee = {name: "John Smith",
  address: {street: "...", city: "..."}};
```

- Here we use the Object constructor to create an object:

```
var person = new Object();
person.firstName = "John";
person.lastname = "Smith";
```

Basic Objects

The object literal is the foundation of JSON, or JavaScript Object Notation. Properties are defined within `{}`. Properties are separated by a comma. The name and value of a property are separate by a colon.

13.2 Constructor Function

- The problem with a basic object is that it has no constructor function that can initialize the object. In addition, a literal based object can not have multiple instances created using the "new" keyword
- A constructor function removes these restrictions and makes JavaScript more object oriented. Example constructor function:

```
function Person(fname, lname) {  
    //Access object property using "this" keyword  
    this.firstName = fname;  
    this.lastName = lname;  
    this.display = function () {  
        alert(this.firstName + " " + this.lastName);  
    }  
}
```

- All properties for a function object must be defined with the "this." prefix
- Although not required, the naming convention is to capitalize the first letter of any function which defines and initializes a new object

Constructor Function

The constructor function (Person) itself is an object. There is only one instance of that object. Whereas p1 is a different object. This object was initialized by executing the constructor function (Person). Many such objects can be created using the new keyword.

Although it is technically possible to invoke the 'Person' function without using the 'new' keyword, this will not have the desired effect. Instead of modifying a new object with the properties and functions of 'Person' this would be done to an unintended object, most likely the browser window.

```
var notAPerson = Person("Mike", "Clueless"); // doesn't create a new object
```

13.3 More on the Constructor Function

- New instances (objects) of the constructor function are created using the "new" keyword

```
var p1 = new Person("Jimmy", "Dean");  
p1.display();
```

- The constructor functions (constructors) can be viewed as factories for objects created when invoked via the *new* operator

- Semantically, JavaScript constructor functions are similar to classes in other languages

13.4 Object Properties

- A property of an object can be accessed in one of two ways:
 - ◇ `obj.property` – The dot notation. This is similar to C++ or Java
 - ◇ `obj["property"]` – The associative array notation (a.k.a. *subscript notation*)

```
var p1 = new Person("Jimmy", "Dean");  
alert(p1.firstName);  
alert(p1["lastName"]);  
p1["lastName"] = "Latimer";  
p1.display();
```

- Arbitrary properties can be added to an object at any time

```
p1.salary = 40000.00; //A new property is added  
alert("Salary is: " + p1.salary);
```

- Accessing an undefined property returns "undefined"

```
if (p1.department !== undefined) { // or (p1.department)  
    alert("Department is: " + p1.department);  
} else {  
    alert("Department is not found"); }  
}
```

- Trying to use an undefined property throws error

```
p1.department.name = "Engineering"; //Throws TypeError
```

Object Properties

Often the conditional test for `(obj.property !== undefined)` is simply done as `(obj.property)` as this will also return true if the property is defined.

13.5 Deleting a Property

- The true dynamic nature of JavaScript can be illustrated by the fact that you can delete a property of an object!
- To delete an existing property of an object, use the *delete* keyword

```
delete obj.propName
```

- ◊ Both dot and subscript notations are supported
- The *delete* keyword deletes both the value of the property and the property itself; the associated memory will be automatically reclaimed by the garbage collector

13.6 Object Properties

- You can enumerate through all object properties using the for...in syntax:

```
var p1 = new Person("Jimmy", "Dean");  
for (propertyName in p1) {  
    alert("Property: " + propertyName);  
    alert(p1[propertyName]);  
}
```

- Every property has a data type that can be retrieved using the typeof operator. Possible values are: "number", "string", "object", "function" and undefined

```
for (propertyName in p1) {  
    if (typeof p1[propertyName] != "function") {  
        alert("Property: " + propertyName);  
        alert(p1[propertyName]);  
    }  
}
```

Object Properties

The first example will show all properties including the "display" function in the Person function object. The value of the "display" property is the text of the function.

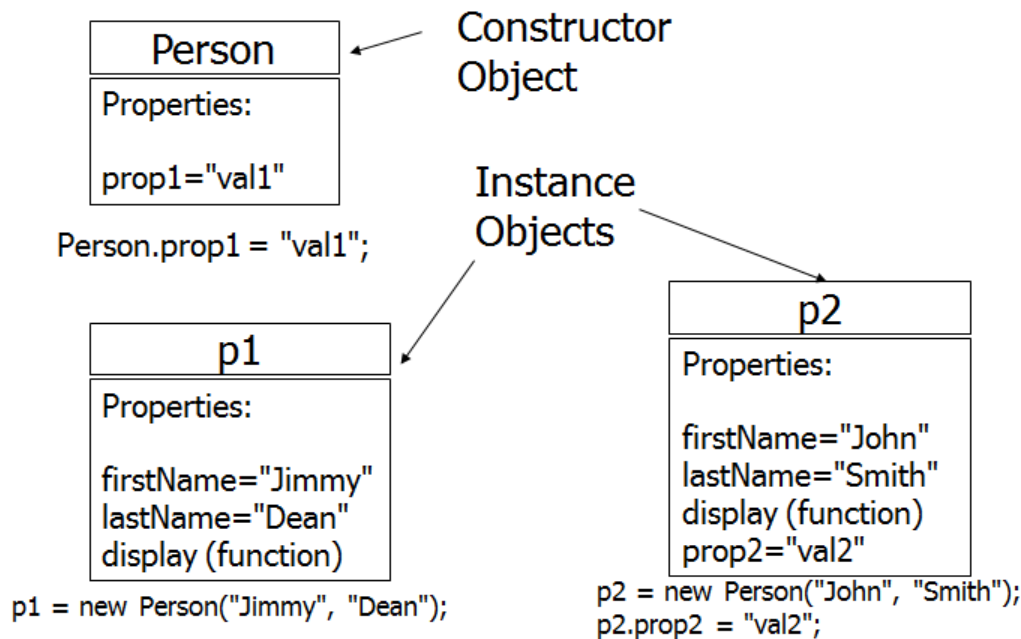
The second example will skip the "display" property because it is a function.

There is no guarantee of the order of the properties in the for...in syntax.

13.7 Constructor and Instance Objects

- It is important to note that the constructor function, like any other JavaScript function, is an object.
- Only one instance of the constructor object will exist. With the use of the new keyword, you can create many instance objects

- Properties that are added outside of the constructor function are not shared by the instances



Constructor and Instance Objects

In this example, we have added a property called "prop1" to the Person constructor object. This property will not be shared by the instance objects. Within the Person() function definition, we had added two instance level properties – firstName and lastName. Such properties are defined within the constructor using the "this" keyword. There is also the "display" function which counts as a property. These properties will be present in all instance objects as soon as they are created and the constructor function is executed. Individual instance objects can have additional properties. For example, we have added a property called "prop2" to p2.

13.8 Constructor Level Properties

- In Java or C++ you can have class level methods and member variables. They are called 'static'. You can do the same in JavaScript using properties of the constructor function
- Constructor level properties can be data or function. Just like for static methods in Java, a function property does not have visibility into the instance level properties (such as firstName)
- Use constructor level properties to:

- ◊ Define constant values
- ◊ Define functions that do not operate on instance level data
- ◊ Define global variables and functions in a unique namespace

```
function MyDate () {           // constructor function
...
}
MyDate.JANUARY=1;           //Constant
MyDate.FEBRUARY=2;
...
MyDate.isGreater = function(date1, date2){ ... }
//Calling the function
if (MyDate.isGreater(date1, date2)) { ... };
```

13.9 Namespace

- Use namespace to avoid any conflict of names for global variables and functions
 - ◊ A namespace is a prefix for variable or function names that make them virtually unique
- It is recommended that you use Java or C# style dot separated namespace, such as "com.webage.util"
- In JavaScript namespaces are created using properties of a constructor
- The namespace properties can also be added to the window object. The advantage of this option is that you can check for the existence of a property to make sure you don't overwrite it. Example:

```
function createNamespace(ns) {
    var nsParts = ns.split(".");
    var root = window;
    for(var i = 0; i < nsParts.length; i++) {
        if(root[nsParts[i]] == undefined) {
            root[nsParts[i]] = new Object();
        }
        root = root[nsParts[i]];
    }
}
createNamespace("com.webage.util");
com.webage.util.doTest = function doTest() {
```

```
alert("Test worked"); }
```

Namespace

Many times you need to define global variables or functions in a script. An HTML page may use many script files. There is possibility of variable or function name collision (multiple scripts using the same name). It is better to do define these global objects using the constructor level property. This way, the variable name is always prefixed with the name of the constructor. As long as the name of the constructor is fairly unique, there is very little chance that your function or variable names will collide with another script included by the HTML page.

Another way to define the namespace is as follows:

```
//Define com.webage.util namespace
function com(){};
com.webage = new Object();
com.webage.util = new Object();
//Define functions or variables in the namespace
com.webage.util.doTest = function() {alert("Test worked");}
//Use the function or variable
<button onclick="com.webage.util.doTest();">Test</button>
```

The downside of this approach is that another script may have also defined the "com" constructor. A better way is to add properties to the "window" object which forms the root of all names by default

13.10 Functions are First-Class Objects

- In JavaScript, functions are "first-class objects"
 - ◇ They co-exist with, and can be treated like, any other JavaScript object
- In particular, JavaScript functions enjoy the same capabilities as objects:
 - ◇ They can be created via literals
 - ◇ They can be assigned to variables, array entries, or properties of other objects
 - ◇ They can be passed as function arguments
 - ◇ They can be returned as values from functions
 - ◇ They can possess properties that can be dynamically created and assigned
- One example of this is passing a function that will be used to compare two values as a parameter to the 'sort' function of an Array

```
var values = new Array(....);
```

```
values.sort( function(value1, value2){
    return value2 - value1; } );
```

Functions are First-Class Objects

Compare the JavaScript code above which simply passes the function used to compare values directly to the function to an implementation in Java. The Java implementation must construct a new anonymous object that implements the Comparator interface and pass this to the function.

```
Arrays.sort(values, new Comparator<Integer>() {
    public int compare(Integer value1, Integer value2) {
        return value2 - value1;
    }
});
```

13.11 Closures

- A closure is the scope created when a function is declared that allows the function to access and manipulate variables that are external to that function
 - ◇ Closures allow a function to access all the variables and other functions that are in scope when the function is declared
- The closure for a function creates a protected scope that can preserve local variable and function declarations that would normally have gone out of scope and would have been deallocated
 - ◇ The closure example below has an anonymous function that is assigned to the 'sayAlert' variable and then returned and executed from outside the scope of the 'sayHello2' function which creates a closure and protects access to the local variable 'text'

```
function sayHello2(name) {
    var text = 'Hello ' + name; // local variable
    var sayAlert = function() { alert(text); }
    return sayAlert; }
var say2 = sayHello2('Jane');
say2();
```

- Declaring one function inside another function creates a closure that allows the function to access variables that would normally have gone out of scope
 - ◇ A reference to the function also references the closure the function was

created in

13.12 Closure Examples

- Local variables within a closure are not copied, they are kept by reference and can change value after the function that will use them is declared
 - ◇ In the following example the value of the 'num' variable changes after the inner function 'sayAlert' is declared and shows a '6' in the alert when the function is called again as 'sayNumba'

```
function say6() {  
    // Local variable that ends up within closure  
    var num = 5;  
    var sayAlert = function() { alert(num); }  
    num++; // changes after the function is declared  
    return sayAlert;  
}  
var sayNumba = say6();  
sayNumba(); // shows alert for 6
```

13.13 Closure Examples

- The following code

```
function increment () {  
    var i = setUpIncrementor(0); // we start with 0  
    i(1);  
    i(10);  
    i(100);}  
function setUpIncrementor(initValue) {  
    return function (inc) {  
        initValue += inc; //first call: 0+1; then 1+10, etc.  
        console.log("initValue in closure: " + initValue);  
        return initValue;  
    }  
}
```

- will produce the following results (we use *console.log* to print results in the JavaScript console in most popular browsers):

```
initValue in closure: 1
```

```
initValue in closure: 11  
initValue in closure: 111
```

13.14 Closure Examples

- If several functions are declared within the same scope they have the same closure and would see changes to the variable values in the closure
 - ◇ In the example below the variable 'num' is within the one closure that contains the three declared functions and all functions will see the current value in the closure when executed

```
function setupSomeGlobals() {  
    // Local variable that ends up within closure  
    var num = 6;  
    // Store some function references as global variables  
    gAlertNumber = function() { alert(num); }  
    gIncreaseNumber = function() { num++; }  
    gSetNumber = function(x) { num = x; }  
}  
gAlertNumber(); // alerts 6  
gIncreaseNumber();  
gAlertNumber(); // alerts 7
```

13.15 Private Variables with Closures

- One use of closures can be to encapsulate some information as a private variable and limit the scope of such variables
 - ◇ Properties that are declared "normally" in JavaScript do not have this kind of protection
- In this example the 'balance' variable declared within the 'Account' constructor function is part of the closure of the other functions that are defined but is not directly accessible from outside the Account constructor function

```
function Account() {  
    var balance = 0;  
    this.getBalance = function() { return balance; };  
    this.deposit = function(amount) {
```

```
        balance += amount;    };
    }
    var myAccount = new Account();
    myAccount.getBalance();    // returns 0
    myAccount.deposit(100);
    myAccount.getBalance();    // returns 100
    myAccount.balance    // this is undefined (inaccessible)
```

13.16 Immediately Invoked Function Expression (IIFE)

- As you see in the above closure examples, they keep connections to the outer variables, which sometimes has negative side-effects
- Immediately Invoked Function Expression (pronounced "iffy") allow you to introduce a new variable scope to prevent it from becoming global using the following JavaScript idiom:

```
(function () {
    var x = ...; // x is not a global variable
})();
```

- An IIFE is called immediately after you define it

Notes:

IIFE idiom helps solve the following common problem you encounter when you are trying to create an array of event handlers. For example, the following code which is intended to create event listeners for an array of hyperlinks defined on the page:

```
var elems = document.getElementsByTagName( 'a' );
for ( var i = 0; i < elems.length; i++ ) {
    elems[ i ].addEventListener( 'click', function(e){
        e.preventDefault();
        console.log( 'This is link #' + i );
    }, 'false' );
}
```

doesn't work as intended as you will be getting handlers printing the same value: This is link # (total number of hyperlinks on the page).

The IIFE would look as follows:

```
var elems = document.getElementsByTagName( 'a' );
for ( var i = 0; i < elems.length; i++ ) {
    (function( locali ){
```

```
elems[ i ].addEventListener( 'click', function(e){
    e.preventDefault();
    console.log( 'This is link #' + locali);
}, 'false' );
})( i );
}
```

The **locali** local scope variable will be assigned the current value of the *i* loop variable on every iteration (0, 1, 2, ...)

13.17 Prototype

- Every constructor function object implicitly has a property called "prototype". It is an object that is used as a template to create new instance objects from the constructor
- Any property defined in the prototype object is automatically added to an instance object

```
function Person(...) { ...
} // end of constructor function
Person.prototype.prop1 = 'vall';
// This is different than Person.prop1

var p1 = new Person();
var p2 = new Person();

p1.prop1 = 'new vall';
// Now p1.prop1 is 'new vall', but p2.prop1 is still 'vall'
```

Prototype

Every constructor object automatically gets the prototype property which is an object. The prototype object is like a template. Any property added to the object will automatically be added to any instance object created from the constructor.

In the example above, the line that adds the 'prop1' property to 'Person.prototype' adds this property to every instance created from the 'Person' constructor function even though the code occurs outside of the constructor function.

13.18 Inheritance in JavaScript

- Inheritance is object-oriented code (behavior) reuse that, among many other benefits, helps reduce the cost and time of development

- There are two schools of thought on how best to do inheritance (two inheritance styles / models):
 - ◇ The classical one (Java, C#, C++, etc.)
 - It is a class-based inheritance (a class inherits from another class); objects are instances of classes
 - ◇ Prototype-based (JavaScript; Self, which actually influenced inheritance design of JavaScript)
 - This style of inheritance is a.k.a. prototypal, or instance-based; normally supported by the *delegation* feature in a language, where supported
 - There are no classes at present in JavaScript (although they will be added with ECMAScript 6 "Harmony")
 - Individual objects are cloned from the generic (template) object

Notes:

Note 1: There is a joke in the programming community that inheritance is an object-oriented way to become rich.

Note 2:

Wikipedia offers this good example of Prototypal inheritance:

A "fruit" object would represent the properties and functionality of fruit in general. A "banana" object would be cloned from the "fruit" object, and would also be extended to include general properties specific to bananas. Each individual "banana" object would be cloned from the generic "banana" object.

Behavior reuse in the "banana" case is performed by way of cloning the "fruit" object that serves as a prototype.

13.19 The Prototype Chain

- Although, we have said all properties of the prototype are automatically added to an instance, in reality, it is achieved through a mechanism called **prototype chain**
 - ◇ Instances may not physically contain these properties; they inherit them from prototypes, if any

- ◇ Prototype chaining is fundamental to the way inheritance is implemented in JavaScript
 - The *prototype chain* is somewhat similar to *multiple inheritance* in other languages (e.g. C++)
- ◇ JavaScript does not impose any limits on how long the prototype chain can be (subject to memory availability in the JavaScript engine)
- ◇ Prototype-based inheritance helps conserve memory as new objects don't need to carry the baggage of state and methods of the prototype(s)

Notes:

According to Douglas Crockford, an authority on JavaScript topics, *"So instead of creating classes [like in Java, C++ using the classical inheritance], you make prototype objects, and then ... make new instances. Objects are mutable in JavaScript, so we can augment the new instances, giving them new fields and methods. These can then act as prototypes for even newer objects. We don't need classes to make lots of similar objects....Objects inherit from objects. What could be more object oriented than that?"* [<http://javascript.crockford.com/prototypal.html>]

The above statement can always be trumped by performance considerations; but this is for another class.

13.20 Traversing Prototype Property Hierarchy

- When you read the property of an instance object, the system first tries to locate the property in that instance. If it can not be found, the system tries to locate it in the prototype object. If it is not there, the system looks for it in the prototype of the prototype object, and so on
 - ◇ In the previous example, the reason `p2.prop1` returns 'val1' is because there is no property called `prop1` in `p2`. The system is actually locating the property in the prototype. `p1`, on the other hand, has a property called `prop1` with value 'new val1'
- This also means, new properties can be added to a prototype at any time, and it will show up for any instances that have been already created

```
var p3 = new Person('John', 'Smith');  
Person.prototype.yell = function(){alert('what up?');}  
p3.yell();    //Due to chaining, the system will
```

```
p1.yell();    // find the yell property in the prototype.
```

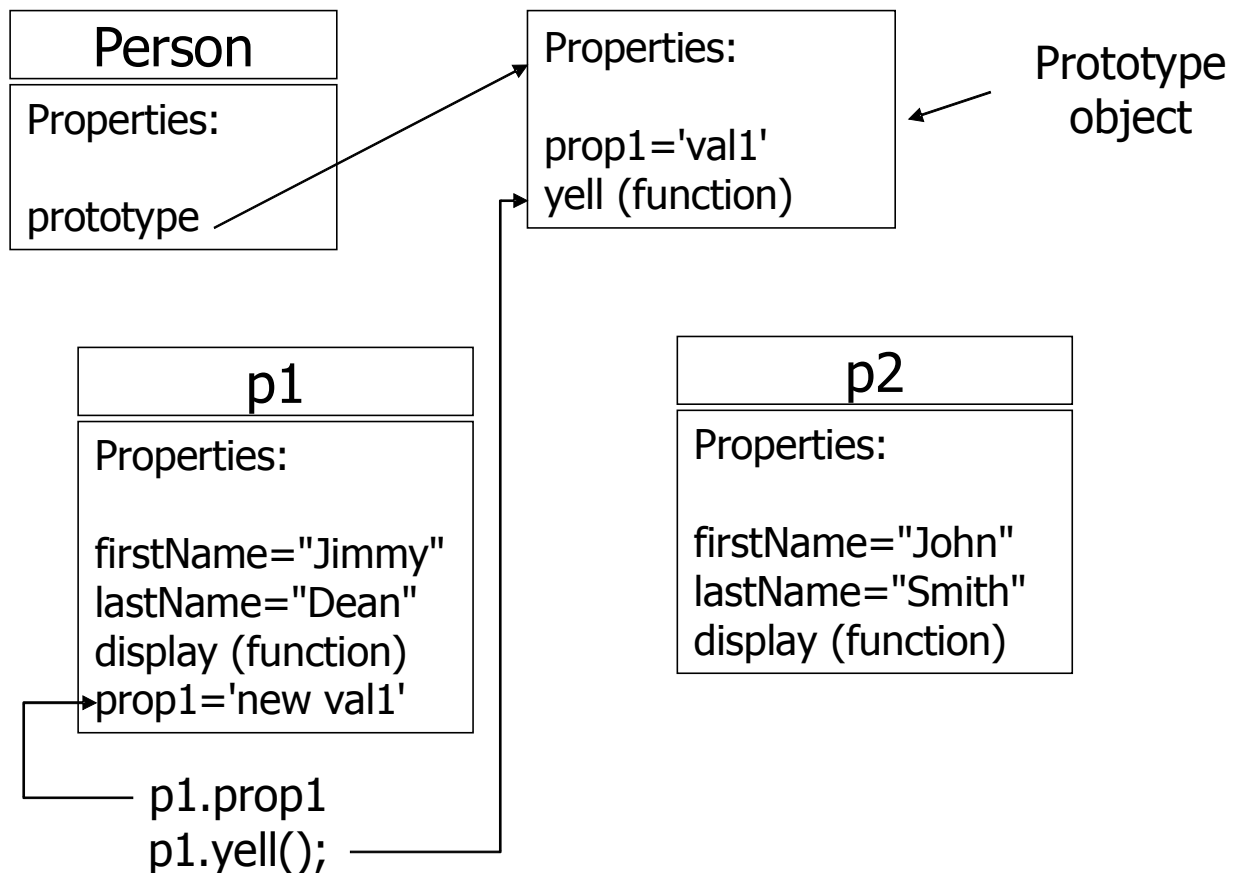
Prototype Property Hierarchy

In the example on the previous slide, after objects p1 and p2 are created, they don't really contain the "prop1" property. Yet, if you refer to it using p1.prop1 or p2.prop1 you get the value 'val1'. This is because of prototype chaining. The system actually fetching the value from the prototype. After we execute the line:

```
p1.prop1 = 'new val1';
```

The p1 object gets a new property called prop1. The expression p1.prop1 will now return that property and will not go up the chain to look at the prototype. The expression p2.prop1 will continue to go up the chain and get the value from the prototype.

13.21 Prototype Chain



Prototype Chain

When you access (read) a property of an instance, the system tries to locate it first in the instance. If not found, it looks for it in the prototype. If not found there, the system looks for it in the prototype of the prototype and so on. In this example, the "prop1" property can be found in p1 instance. The yell property is not found there. As a result, when you call p1.yell(), the system locates the function property in the prototype object.

13.22 Summary

- Even though JavaScript is a "functional" language, it can be used in an object oriented way
- Constructor functions can define properties and functions on objects when called with the 'new' keyword
- Closures in JavaScript can extend the scope of variables local to a function declaration
- The 'prototype' property on an object can establish a relationship similar to inheritance in other object oriented languages
- When accessing a property of an object, if it is not found JavaScript will look to the object referenced by 'prototype' to keep searching for the property

Chapter 14 - jQuery Overview

Objectives

After completing this unit, you should be able to:

- Explain how jQuery is used in web site development
- Make the jQuery framework available in your web development environment
- Associate JavaScript using jQuery with the proper DOM event
- Properly use the jQuery \$ function in your JavaScript
- Effectively use jQuery wrapper objects in your JavaScript, including the use of chaining

14.1 What Is jQuery?

- A JavaScript (ECMAScript) library
- "write less, do more" – You can achieve great effects with small amounts of code.
- Supports "Unobstrusive JavaScript"
 - ◇ "Unobstrusive JavaScript" discussed in Unit 5
- Other JavaScript libraries (also called frameworks)
 - ◇ Prototype
 - ◇ MooTools
 - ◇ YUI
 - ◇ Dojo

14.2 Benefits of Using a JavaScript Library

- Reduces browser incompatibility problems
- Speeds development time by simplifying common tasks
- The most popular and arguably the best among all JavaScript libraries.
- Tons of additional plug-ins available from **plugins.jquery.com**. They can

help you add rich user experience to your site very quickly.

14.3 jQuery Example

- To draw a box around every div with CSS style class "box" (You're not expected to understand the details of this code yet!):

```
$("#div.box").css("border-style", "solid").  
    css("border-width", "2px");
```

14.4 CSS Selectors

- Recall the jQuery example we just saw
`$("#div.box").css("border-style", "solid").css("border-width", "2px");`
- jQuery uses CSS selectors to identify lists of elements
- Examples:
 - ◇ "p" all p elements
 - ◇ "p a" all anchors inside paragraphs
 - ◇ "p > a" all anchors that are direct children of paragraphs
 - ◇ "div.box" every div with CSS style class "box"
 - ◇ ".box" every element with CSS style class "box"
 - ◇ "#testButton" element with id testButton
- More on selectors in Unit 2

14.5 How to Use jQuery

1. From Google URL

```
<script type="text/javascript"  
  
src="http://ajax.googleapis.com/ajax/libs/jquery/<jQuery  
Version>/jquery.min.js"></script>
```

2. From your own web server

```
<script type="text/javascript"
      src="/jquery/jquery-<jQuery Version>.min.js">
</script>
```

3. Embedded within your individual JEE web application

```
<script type="text/javascript"
      src="${pageContext.request.contextPath}/script/
      jquery-<jQuery Version>.min.js">
</script>
```

where `<jQuery Version>` is the version of jQuery library you want to use, e.g. 1.10.2

How to Use jQuery

Note: "min" in the filename means "minimized"; comments and whitespace are reduced and internal variables are given shorter names to reduce download time.

14.6 Practical Usage Notes

- jQuery may conflict with other JavaScript libraries such as Prototype, MooTools, YUI, or dojo in use of \$.
- Solutions
 - ◇ Avoid using multiple libraries, especially multiple libraries that use "\$"
 - ◇ Invoke `jQuery.noConflict()`
 - Cancels jQuery usage of \$ and reverts \$ to its previous definition
 - See http://docs.jquery.com/Using_jQuery_with_Other_Libraries
- Debugging:
 - ◇ Chrome built-in JavaScript debugger
 - ◇ Firefox Firebug extension
 - ◇ `console.log()`

14.7 Background – DOM

- DOM stands for **Document Object Model**. According to this model, an XML document is loaded into memory as a tree like hierarchy of nodes.
- W3C specifies the API for DOM tree navigation and manipulation. Nearly all browsers provide a JavaScript binding for this API.
- Key types of nodes in a DOM tree are:
 - ◇ **Document** – Represents the DOM document itself. The `documentElement` property points to the root element.
 - ◇ **Element** – Represents a tag, like, `<html>` and `<div>`.
 - ◇ **Text** – Represents the body text of an element. `<p>Body text</p>`.
 - ◇ **Attr** – Attributes of an element. `<p class="big">Body text</p>`.

14.8 Background - DOM Ready Events

- Two DOM events can be used to detect if a page is loaded:
 - ◇ **onload** - Fires when the user agent finishes loading all content within a document, including images and scripts.
 - ◇ **DOMContentLoaded** – Fired when the complete DOM tree is available. This happens prior to the images are loaded. You can start manipulating the DOM tree at this point.
 - This event is being standardized; IE before IE 9 calls it `onreadystatechange`
 - Often referred to simply as "the DOM ready event"
- The point is, we want to run our DOM manipulation JavaScript code as soon as the `DOMContentLoaded` event occurs. Doing that will complete page composition earlier than if we did DOM manipulation after `onload`.

14.9 Background - JavaScript Functions

- JavaScript functions are Function objects...

```
function saySomething() {
```



```
    alert('Something');    }
// same as
saySomething = function() {
    alert('Something');    }
// same as
window.saySomething = function() {
    alert('Something');    }
```

- Function objects can be passed as arguments of other functions, just like any other JavaScript object.

```
setTimeout(
    function() {
        alert("Five seconds have elapsed.");
    }, 5000);
```

- The point is, a JavaScript function is an object; it doesn't have a name but it may be assigned to a variable that has a name.

14.10 The jQuery Function Object

- At the heart of the API is the **jQuery** object. It is a JavaScript function that is also set as a property of the window. Example:
 - ◇ window.jQuery(argument)
 - ◇ jQuery(argument)
- For the sake of brevity, another variable called "\$" is declared that points to the jQuery function object. Hence, the following is equivalent to the examples above:
 - ◇ \$(argument)
- The \$() notation is preferred.

14.11 What Does the \$() Function Take as Argument?

- The function takes many different types of parameters. It's behavior changes drastically based on what is supplied.
- There are four cases:

1. The parameter is a string that looks like a CSS selector. e.g. \$('p.big').
2. The parameter is a function object
3. The parameter is a string that looks like HTML markup
4. The parameter is an Element, Document, or Window object

14.12 What Does the \$() Function do?

- Case 1: When passed a string that looks like a selector, \$() returns a collection object containing all matching DOM elements. Example:

```
var list = $('p.big');
```

- ◇ The list variable is a collection of all <p> DOM elements that have the 'big' CSS class assigned.
- ◇ Data type of each item in the collection is a DOM Element.

- Case 2: When passed a function object, \$() attaches the function as a handler for the DOM ready event. Example:

```
$(  
    function() {  
        alert('The DOM is ready!');  
    }  
);
```

- ◇ In this example, when the DOM ready event occurs, an alert will be displayed

- Cases 3 and 4 will be discussed in Unit 4

14.13 The jQuery Wrapper

- \$(someSelector) returns a collection of DOM Elements
- Since the collection has additional jQuery properties beyond those available to any collection, it is often called a wrapper
- It is also called a
 - ◇ jQuery object

- ◊ jQuery result
- ◊ jQuery set
- ◊ wrapped set
- The DOM Elements are returned in the order they appear in the document

14.14 The jQuery Wrapper as an Array-Like Object

- The following illustrates the jQuery object as an array-like object but we'll see a better way to write the code next.

```
var paragraphs = $("p");
for(var i=0; i<paragraphs.length; i++) {
    paragraphs[i].innerHTML =
        "New text for each paragraph.";
}
```

- Better way to write the same loop:

```
$("p").html("New text for each paragraph.");
```

14.15 Note: innerHTML() vs. .html()

- innerHTML() is an Element method that places the provided string between the start and end tags of the specified Element.
- .html() is a jQuery wrapper method that does the same thing (applied to each element of the jQuery collection)

14.16 jQuery Wrapper Chaining

- Apply the html function to each DOM object and return the wrapped collection

```
$("p").html("New text for each paragraph.");
```

- Since the wrapped collection is returned, we can chain!

```
$("p").html("New text for each paragraph.")
.css("color", "red");
```

14.17 API Function Notation

- jQuery documentation shows the functions supported by the wrapper object using the *.FUNCTION_NAME()* notation. Example:
 - ◇ `.html()`
 - ◇ `.append()`
 - ◇ `.insertAfter()`
- We will follow the same notation in this class.

14.18 Handling DOM Ready Event

- Most jQuery code goes into the DOM ready event handler. This can be registered as follows:

```
$(  
    function() {  
        // your JavaScript here  
    }  
);
```

- The above is equivalent to:

```
$(document).ready(  
    function() {  
        // your JavaScript here  
    }  
);
```

- You can attach multiple DOM ready handlers in a page. Example:

```
$( function(){...} ); //Handler 1  
$( function(){...} ); //Handler 2
```

14.19 xhtml Note

- JavaScript in xhtml documents cannot include characters such as '<' for less than so when working in xhtml enclose your script in a CDATA section

```
<script type="text/javascript">  
    // </pre></div><div data-bbox="875 907 916 925" data-label="Page-Footer"><hr/><p>164</p></div>
```

```
$(function() {  
    if ($("#p").length < 20) {  
        alert("Few paragraphs");  
    } else {  
        alert("Many paragraphs");  
    }  
})  
// ]]>  
</script>
```

14.20 References

- <http://blog.rebeccamurphey.com/2010/06/17/open-source-jquery-training>
- <http://jquery.com>
- jQuery in Action, 2nd Edition, by Bear Bibeault & Yehuda Katz
- JavaScript: The Definitive Guide, 6th Edition, by David Flanagan

14.21 Summary

- What is jQuery?
- How to use jQuery
- Selectors
- The DOM Ready Event
- Functions as "First-Class Objects"
- The \$ Function
- The jQuery Wrapper
- Wrapper Chaining

Chapter 15 - Selectors

Objectives

After completing this unit, you should be able to:

- Name several CSS3 pseudo-selectors and use them to select elements with jQuery
- Recognize potentially inefficient use of selectors and rewrite for faster performance
- Use jQuery wrapper relationship methods such as `.children()` to select elements
- Use jQuery wrapper filter methods to select elements and the `.end()` method to restore the original selection
- Use the `.is()` and `.hasClass()` methods to test elements
- Save and reuse jQuery wrapper objects
- Use the jQuery wrapper `.each()` method to operate on elements
- Explain the element referenced by "this" depending on the context

15.1 Background: The Sizzle Selector Engine

- Sizzle is a JavaScript library written by John Resig, the inventor of jQuery
- Finds DOM elements using CSS selector syntax
- You may find yourself in this library when debugging

15.2 Selecting Elements by Attribute

- We listed some example selectors in Unit 1
- Here's another example—specifying an attribute

```
$("input[name=firstName]");
```

- Compound selectors are also supported, for example

```
$("input[type='text'], input[type='password']")
```

- Selecting by attribute can be slow

15.3 Pseudo-Selectors

- jQuery supports most CSS3 Pseudo-selectors, for example:
 - ◇ "p:even" every other p element, starting with first (0th)
 - ◇ "p:nth-child(3)" every p element that is the 3rd child of its parent (1-based indexing)
 - ◇ "a[href\$='pdf']" anchors with an href value the ends with 'pdf'
 - ◇ "a[href^='acme']" anchors with an href value the starts with 'acme' (reg ex can be slow)
- jQuery supports additional pseudo-selectors, including
 - ◇ "p:eq(3)" the 4th p (0-based indexing)

15.4 Form Pseudo-Selectors

- Example

```
$("#registrationForm :input");  
◇ Gets all elements that accept input
```

- Form Pseudo-Selectors include
 - ◇ :input
 - <input>, <textarea>, and <select> elements
 - ◇ :button
 - <button> elements and elements with type="button"
 - ◇ :checkbox
 - input elements with type="checkbox"

15.5 Form Pseudo-Selectors

- More Form Pseudo-Selectors (there are more not listed here)

- ◇ :checked
 - checked inputs
- ◇ :password
 - input elements with type="password"
- ◇ :radio
 - input elements with type="radio"
- ◇ :selected
 - options that are selected
- ◇ :submit
- ◇ input elements with type="submit"

15.6 Faster Selection

- Beginning your selector with an ID is fast

```
$("#myContainer div.myImages");
```
- Even faster

```
$("#myContainer").find("div.myImages");
```

 - ◇ Faster because ID-only selections such as `$("#myContainer")` are handled natively in the browser via `document.getElementById()` without going through the Sizzle selector engine

15.7 Selecting Elements Using Relationships

- jQuery methods such as the following take the existing wrapped element set and return a new wrapped element set based on relationships
 - ◇ `.children()`
 - ◇ `.closest()`
 - The single nearest ancestor
 - ◇ `.next()` and `.prev()`
 - Next/previous siblings

- ◇ .parent()
 - Direct parents of each element in the original set

15.8 Selecting Elements Using Filters

- jQuery methods such as the following take the existing wrapped element set and return a new wrapped element set by filtering
 - ◇ .has(someSelector)
 - ◇ .eq(someIndex)
 - Reduces the set to the element at the specified index
 - ◇ .not(someSelector)

15.9 More on Chaining: .end()

- If we change the selection in a chain, we can use .end() to get back to the original selection

```
$("body")
  .find("p")
  .eq(2).html("New text for par 3.").end()
  .eq(0).html("New text for the first par.");
```

15.10 Testing Elements

- .is() and .hasClass() return true or false
- Examples

```
if ($("p:first").is(".myClass") { ... }
if ($("p:first").hasClass("myClass") { ... }
```

15.11 Is the Selection Empty?

- The following won't work

```
if ($("li.customer")) { ... }
```

- `$()` always returns an object and objects always evaluate to true; the code

inside the if statement will always run

- Instead, test the selection's length property. A value of 0 evaluates to false when used as a boolean value

```
if ($("#li.customer").length) { ... }
```

15.12 Saving Selections

- Evaluating a selector to build a wrapper object can be expensive. You should cache the wrapper object to avoid evaluating a selector unnecessarily. Example:

```
//Evaluate selector outside the loop
var customers = $("#li.customer");
for (...) {
    //Use the customers object here
}
```

- Beware: if you add dynamically add new elements later, you need to execute the selection again; stored selections aren't updated

15.13 Iterating Through Selected Elements Using .each()

- .each() invokes a call back function repeatedly, once for each element in the collection. Example:

```
$("#p").each(
    function (i) {
        alert("id=" + this.id + " index=" + i);
    }
);
```

- The callback function takes a 0 based index as parameter.
- The **this** keyword within the callback points to the current DOM Element object at that index. To understand this better, let's review JavaScript functions, methods, this, and function context

15.14 JavaScript Methods

- Suppose f is a function and o is an object; the following assignment

defines a method named `m` on object `o`

`o.m = f`

- A "method" is a function invoked via an object
- Example:

```
var firstParagraph = $("p")[0];  
firstParagraph.onclick = function() {alert(this.id)};
```

- `onclick()` is a method because it is invoked via `firstParagraph`

15.15 JavaScript "this"

- In OO programming languages such as Java and C++
 - ◇ "this" in a function refers to the current instance of the class within which the function was declared
- In JavaScript
 - ◇ functions are not declared within a class and the meaning of "this" is different
 - ◇ The meaning of "this" depends on how the function is invoked

15.16 Function Context

- The function context this depends on how the function is invoked
- (1) Function invoked as an object property (in other words, as a method)
 - ◇ this refers to the object
 - ◇ Very similar to Java, C++, etc.

```
var firstParagraph = $("p")[0];  
firstParagraph.onclick = function() {alert(this.id)};
```

15.17 Function Context

- Top-level functions
 - ◇ this refers to window because top-level functions are properties of

window

```
function demo() {alert(this.innerWidth + " by " +  
this.innerHeight)};
```

- Same as:

```
demo = function() {alert(this.innerWidth + " by " +  
this.innerHeight)};
```

15.18 The Function call() Method

- Every Function has a call() method; the parameter is the desired context

```
var firstParagraph = $('p')[0];  
var secondParagraph = $('p')[1];
```

```
idGetter = function() {alert(this.id)};
```

```
idGetter.call(firstParagraph);  
idGetter.call(secondParagraph);
```

15.19 .each() Revisited

- Recall our .each() example:

```
$("p").each(  
  function (i) {  
    alert("id=" + this.id + " index=" + i);  
  }  
);
```

- The .each() wrapper method takes a function with one parameter
- The function is invoked once for each element in the jQuery wrapper set
- In the function, **this** refers to the current DOM element

15.20 Summary

- Selectors and Pseudo-Selectors
- Faster Selection

- Selecting Elements Using Relationships and Filters
- Testing Elements
- Using the Results: The jQuery Wrapper `each()` Method
 - ◇ JavaScript Methods
 - ◇ "this" and Function Context, Three Cases: Methods, Top-level Functions, and the `call()` Method
 - ◇ `.each()` Revisited

Chapter 16 - Style Class Manipulation

Objectives

After completing this unit, you should be able to:

- Add styles to an element using jQuery either directly with `.css()` or indirectly with `.addClass()`
- Recognize and specify jQuery names for CSS style properties
- Use jQuery to create and add a stylesheet to a document
- Use jQuery to interrogate the value of an element style property
- Use jQuery to add an attribute to an element

16.1 Two Options

- Apply styles directly to Element

```
$("a.customer").css("color", "#FF00FF");
```

- ◇ Advantage

- Easy

- ◇ Disadvantages

- Specific style information should be kept out of the JavaScript
 - Slow if the number of elements is large

- Use classes

```
$("a.customer").addClass("customerClass");
```

- ◇ Separates style and behavior

- ◇ Faster for a large number of elements

16.2 Specifying Style Properties

- jQuery supports both
 - ◇ Hyphenated CSS property names
 - ◇ The JavaScript camel-case alternative
- Style property reading example: both of the following return a string such

as "21px"

```
$("#h1").css("font-size");  
$("#h1").css("fontSize");
```

16.3 Setting Style Properties

- In general we want to avoid setting style properties directly but here's how to do it:

```
$("#h1").css("fontSize", "100px");  
$("#h1").css({"fontSize" : "100px",  
"color" : "red" });
```

- Note that in the second example we're passing in a JavaScript object literal with two properties

16.4 .addClass() / .removeClass()

- Examples

```
var $h1 = $("#h1");  
$h1.addClass("important");  
$h1.removeClass("important");  
$h1.toggleClass("important");  
if ($h1.hasClass("important")) { ... }
```

16.5 Defining a Stylesheet

- Typically we reference existing stylesheets
- If necessary we can add our own!

```
$('#<style type="text/css">a.customer{color : #FF00FF }  
</style>').appendTo('head');
```

16.6 Setting & Getting Dimensions

- Examples

```
$("#h1").width("50px");  
$("#h1").width();
```



```
$("#h1").height("50px");  
$("#h1").height();
```

16.7 Attributes

- Getting and setting attributes with jQuery is syntactically similar to getting and setting styles

```
$("#a").attr("href", "sameForAll.html");  
$("#a").attr({  
    "title" : "Same Title",  
    "href" : "newForAll.html"  
});
```

16.8 Summary

- Two options: add styles directly or use classes
- Specifying style properties
- Setting style properties
- Using classes
 - ◇ .addClass() / .removeClass
 - ◇ Defining stylesheets
- Dimensions
- Attributes

Chapter 17 - DOM Manipulation

Objectives

After completing this unit, you should be able to:

- Create new elements from markup text using the jQuery \$ function
- Use jQuery wrapper methods on an existing DOM element by first wrapping the element using the jQuery \$ function
- Read HTML from the document as a string using the .html() method
- Read HTML minus the markup as a string using the .text() method
- Update the DOM using .append(), .prepend() and .remove()
- Rewrite loops that manipulate the DOM to improve their performance

17.1 The \$ Function Revisited

- Recall that the behavior of the \$ Function depends on the parameter passed to it. There are four cases:
 - (1) The parameter is a string that looks like a CSS selector
 - (2) The parameter is a function
 - (3) The parameter is a string that looks like HTML markup
 - (4) The parameter is an Element, Document, or Window object
- Let's look at cases 3 and 4

17.2 The \$ Function Revisited

- (3) when passed a string that looks like HTML markup

```
$ ('<p>New paragraph.</p>')
```

the \$ Function returns a jQuery object representing the HTML, for example

```
$ ('<p>New paragraph.</p>') .
```

```
insertAfter("#testParagraph1");
```

- (4) when passed an Element, Document, or Window object
- \$(document)

the \$ Function returns a jQuery object that wraps the Element, Document, or Window object

17.3 Getters and Setters

- .html() as a getter refers only to the first element in the selection

```
var p1 = $("p").html();
```

- Getters return whatever they were asked to get; you cannot continue the chain
- In this case: everything between the start and end tags as a string
- .html() as a setter effects all elements in the selection
- \$("p").html("New text for all paragraphs!");
- The original jQuery wrapped set is returned, allowing you to continue the chain

17.4 The text() Element Method

- Example

◇ Assume the following HTML

```
<ul id="asteroids"> <li>Ceres</li> <li>Vesta</li></ul>
```

◇ JavaScript

```
var answer = $("#asteroids").text();// Ceres Vesta
```

17.5 Appending DOM Elements

- The ability to append and remove DOM Elements without loading a new page allows for dynamic, interactive web pages
- Examples

```
$("p").append("Text at the <b>end</b>.");
```

```
$("#p1").append($("#p2")); // moves #p2 !!
$("p").prepend("Text at the <b>beginning</b>.");
```

- Many other related methods are available

17.6 Removing DOM Elements

- `.remove()` removes all the elements in the set
- `.empty()` removes the contents of all the elements in the set

17.7 Performance

- Manipulating the DOM is relatively slow

```
$("#img").each(function(index, image) {
    $("<tr><td>" + image.src + "</td><td>" +
        image.alt + "</td></tr>").
        appendTo("#imageTable");
});

var rowHtml = "";
$("#img").each(function(index, image) {
    rowHtml += "<tr><td>" + image.src + "</td><td>" +
        image.alt + "</td></tr>";
});

$("#imageTable").html(rowHtml);
```

17.8 Performance

- Detach Elements to work with them more quickly

```
var table = $("#imageTable");
var parent = table.parent();
table.detach();

$("#img").each(function(index, image) {
    $("<tr><td>" + image.src + "</td><td>" +
        image.alt + "</td></tr>").
```

```
        appendTo (table) ;  
    } ) ;  
  
parent.append (table) ;
```

17.9 Summary

- Using the \$ function to create wrapped elements
- jQuery wrapper methods for manipulating the DOM
 - ◇ html()
 - ◇ text()
 - ◇ append()
 - ◇ remove()
- Performance tips

Chapter 18 - Events

Objectives

After completing this unit, you should be able to:

- List several benefits of "Unobtrusive JavaScript"
- Define "Unobtrusive JavaScript" and how it differs from the traditional approach
- Bind more than one event handler to the same event using jQuery
- Bind the same event handler to more than one event
- Identify all the elements through which an event passes
- Register an event handler for elements that have not yet been created, using the `.on()` method
- Simulate user-generated events using jQuery

18.1 Event Overview

- HTML Events and Responses
 - ◇ Load a page in response to click on anchor element
 - ◇ Load a page in response to form submission
- Example jQuery DOM Events
 - ◇ `.blur()`
 - ◇ `.change()`
 - ◇ `.click()`
 - ◇ `.keyup()`
 - ◇ `.mousedown()`
 - ◇ `.mouseover()`
 - ◇ `.mouseup()`

18.2 Old School: Event Handling Using HTML Element Attributes

- Consider the following typical HTML

```

```

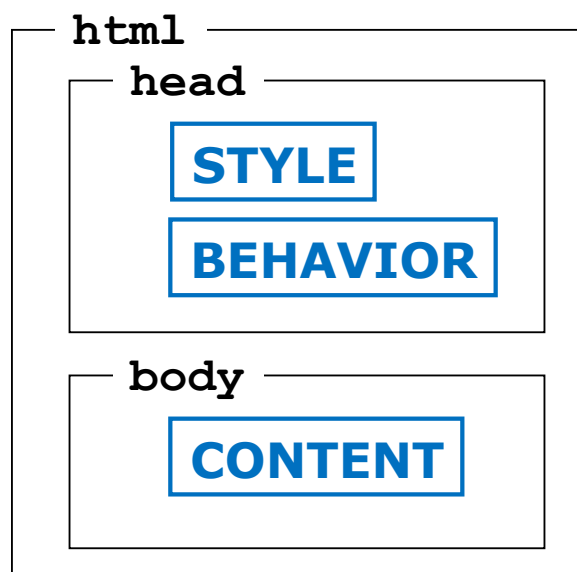
- The result is

```
myImgRef.onclick = function(event) {
    alert("Image clicked!");
};
```

- We will modify this approach
 - ◇ Separate the structured content from the behavior
 - ◇ Use jQuery

18.3 Unobtrusive JavaScript

- Separate behavior (JavaScript) from style (CSS) and structured content (the body)
- Part of the "Web Standards" movement
- Modularity benefits may increase line-count



18.4 Unobtrusive JavaScript Example

- In the STRUCTURED CONTENT portion of document (<body>)

```
<button type="button"          id="testButton">Click Me!
</button>
<p id="testParagraph">Content 1 Here</p>
```

- In the BEHAVIOR portion of document (<head> <script>)

```
$("#testButton")[0].onclick = function() {$
$("#testParagraph").css("color", "red")};
```

18.5 Multiple Handlers

- The following sets the onclick handler for the #test element

```
$("#test")[0].onclick = function() {alert("Clicked!")};
```

- But what about this?

```
$("#test")[0].onclick = function() {alert("Clicked!")};
$("#test")[0].onclick = function() {alert("Clicked
again!")};
```

- There is a better way...

Multiple Handlers

The second handler replaces the first.

18.6 Using jQuery Wrapper Event Registration Methods

- Note the use of the wrapper .click() method rather than the element onclick() method

```
$("#myButton").click(function(event) {alert("1 " +
event);})
    .click(function(event) {alert("2 " + event);});
```

- Aside: Counting clicks using a closure...

```
$("#myButton").click(
    (function(event) {
        var count=0;
        return function(event) {
```

```
                                alert("count " + +
+count);
                                }
                                }) ()
                                );
```

Note

This closure example is much more complex than it seems. The parameter passed to the `.click()` method is an *anonymous self-executing function*. The `()` parenthesis cause the anonymous function to be called. The function creates a variable called `count` and then returns another anonymous function. The second anonymous function is actually the click handler.

Notice that the second function accesses the `count` variable created in the outer function. In traditional languages like C, Pascal, C++, or Java this would not be possible. The variable `count` would no longer be in scope. In JavaScript this variable stays in scope as long as the inner function needs it. This is an example of a closure.

18.7 The `.on()` Method

- The `.on()` Method allows easy registration of the same handler for multiple events

```
$("#p1").on("click mouseover", function(evt) {
                                console.log("Got here");
                                });
```

- The earlier methods we saw were just convenience methods implemented with this method

18.8 Event Propagation

- When an event occurs (DOM Level 2 Event Model)
 - ◇ Capture phase
 - The event is first passed down the DOM tree
 - From the root to the target element
 - Not supported by jQuery because not supported by IE
 - ◇ Bubble phase

- Then the event is passed back up the DOM tree
- From the target element to the root
- Thus can register an event handler with a parent element rather than each child

18.9 Handlers for Elements Before They Exist!

- Register myHandler for every with myClass in the current DOM

```
$( "li" ).click( myHandler );
```

- Register myHandler for every in the #myList now or added later

```
$( "#myList" ).on( "li.myClass", "click", myHandler );
```

18.10 The Event Object

- The Event object provided by jQuery provides browser-independent access to the actual event object it wraps
- Properties include
 - ◇ currentTarget
 - ◇ pageX, pageY
 - ◇ screenX, screenY
- Methods include
 - ◇ preventDefault()
 - ◇ stopPropagation()

18.11 Triggering Events

- Typically the user triggers events
- jQuery provides methods for triggering events under script control

```
$( "div.tabbed a" ).first().click();
```

18.12 Summary

- Event Overview
- Unobtrusive JavaScript
- Using the jQuery Wrapper Event Handler Registration Methods
- Event Propagation and Delegation
- The Event Object
- Triggering Events

Chapter 19 - Utility Functions

Objectives

After completing this unit, you should be able to:

- List several jQuery utility functions
- Use jQuery to remove leading and trailing blanks from a string
- Use jQuery to execute the same function on each element of an array
- Use jQuery to find the index of a given element in a given array
- Use jQuery to test an object to determine whether it is empty or an array or a function or none

19.1 The jQuery Object Revisited

- jQuery is an object, a window property
- It happens to be a function
- It is aliased as \$
- The following are equivalent:
 - ◇ window.jQuery
 - ◇ jQuery
 - ◇ \$

19.2 Functions May Have Methods

- Suppose f is a function and o is an object; the following assignment defines a method named m on object o

`o.m = f`

- A "method" is a function invoked via an object
- The keyword 'this' in the function refers to o
- Now reread all that and recall that o may be a function object itself!

- Functions can have methods!
- In particular, \$ has methods

Functions May Have Methods

The first few bullets are repeated from Unit 2.

19.3 A jQuery Utility Function: \$.trim()

- Keeping in mind the previous two slides, consider the following

```
var trimmedFirstName = $.trim(firstName);
```

- trim() is a method of the jQuery object
- Even though trim() is technically a method (because it is invoked via an object), it is called a "utility function" in the jQuery documentation
- A jQuery utility function is a \$ method
- At <http://jquery.com> they are documented as, for example, jquery.trim()

19.4 \$.each()

- We have already looked at the .each() wrapper method
- There is also a \$.each() utility method
- jQuery.each(collection, callbackFunction(index, objectValue))
- Iterates over both objects and arrays; non-array-like objects are iterated via their named properties
- Example:

```
$.each(["alpha", "beta"], function(index, value) {  
    console.log("element " + index + "is " + value);  
});
```

19.5 Example jQuery Utility Functions

- jQuery.contains(container, contained)
 - ◇ Is DOM node "contained" within DOM node "container"?

- `jQuery.data(element, key, value)`
 - ◇ Associates "key"/"value" with the specified element Returns value.
- `jQuery.removeData(element, key)`
 - ◇ Remove a previously-stored piece of data
- `jQuery.parseXML()`
 - ◇ Parses a string into an XML document

19.6 Example jQuery Utility Functions

- `jQuery.inArray(value, array)`
 - ◇ Returns index of value in array or -1 if not found
- `jQuery.isArray(obj)`
 - ◇ Determine whether obj is an array
- `jQuery.isEmptyObject(obj)`
 - ◇ Check to see if obj is empty (contains no properties)
- `jQuery.isFunction(obj)`
 - ◇ Determine if the argument passed is a JavaScript function object
- `jQuery.isPlainObject(obj)`
 - ◇ Check to see if an object is a plain object (created using "{}" or "new Object")

19.7 Example jQuery Utility Functions

- `jQuery.noop()`
 - ◇ An empty function
- `jQuery.now()`
 - ◇ Returns a number representing the current time
- `jQuery.parseJSON`
 - ◇ Takes a well-formed JSON string and returns the resulting JavaScript

object

- ◇ We'll discuss in Unit 10

19.8 Summary

- What are utility functions?
 - ◇ The jQuery object revisited
 - ◇ Functions may have methods
 - ◇ Utility functions defined
- \$.trim()
- \$.each()
- Other utility functions

Chapter 20 - Ajax

Objectives

After completing this unit, you should be able to:

- Explain what Ajax is and what its benefits are
- Describe the typical contents of an Ajax response
- Use the jQuery \$.get() utility function to make an Ajax request
- Use the jQuery .load() method to make an Ajax request
- Use the .data() method to associate named data with an element

20.1 Ajax Overview

- Ajax
 - ◇ Provides the ability to make asynchronous requests to the server and update only a portion of the page in response
 - ◇ Allows the page is still active and available to the user while the request is being made
 - ◇ An essential part of modern web development
- Ajax stands for "Asynchronous JavaScript and XML"
 - ◇ In spite of the "x" in "Ajax", XML is not required and is, in fact, rarely part of contemporary Ajax usage
- jQuery provides a powerful, implementation-independent, easy-to-use Ajax API

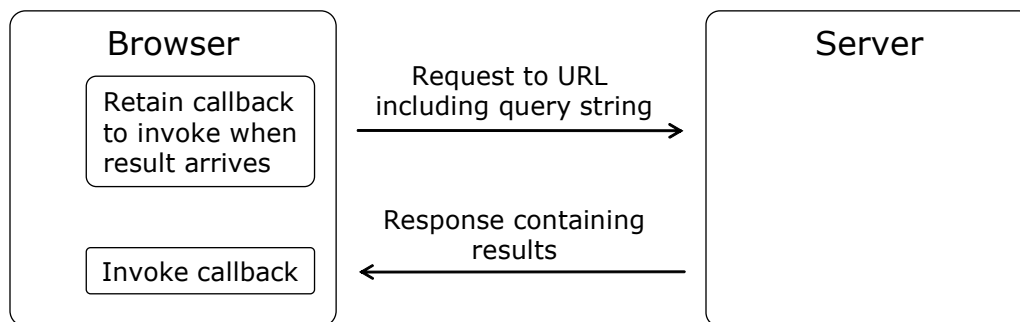
20.2 The Browser & the Server

- Ajax is implemented in the browser with
 - ◇ the XMLHttpRequest object
 - emulated in IE starting with IE7
 - Referred to as "XHR"

- ◇ Microsoft's XMLHTTP ActiveX control
- Working with Ajax requires a server to respond to requests
 - ◇ You can't experiment with Ajax using only a browser and a text editor
 - ◇ Fortunately we have Apache!

20.3 The Ajax Request

- To send a request to the server we must provide
 1. The HTTP method to use (GET or POST)
 2. The URL on the server to contact
 3. Any query string parameters
 4. A callback to invoke when the result arrives



- The Ajax request looks like any other HTTP request message.

20.4 The Ajax Response

- The response to a typical HTTP GET or POST is an entire web page.
- The response to an Ajax request contains the data needed for a partial page update
 - ◇ XML
 - ◇ JSON
 - ◇ Really could be anything; you just need to know what it will be and how to parse it in your callback
 - ◇ Often an HTML fragment

- The response may arrive at any time—or never, in case of an error
- It's the job of the callback function to use the data in the response to update the page

20.5 Sending an Ajax Request With jQuery - The General Case

- An example of the general jQuery Ajax solution:

```
$.get("/test/MyService",  
    "firstName=Bob;lastName=Smith",  
    function(responseContent) {  
        $("#customerResult").html(responseContent);  
    }, "html"  
);
```

20.6 When this code is executed...

- A GET is sent to `/test/MyService?firstName=Bob;lastName=Smith`
- At some later time, asynchronously, a response is received
- The response is interpreted as HTML (because of the "html" parameter) and passed to the callback
- The callback updates the element that has id `customerResult` with the contents of the response
- There is an easier way to implement this particular example!

20.7 Sending an Ajax Request With jQuery - Simpler, Typical Case

- Often, the result of an Ajax request is an HTML fragment that we want to load into a page element.
- The following has the same result as the previous `$.get` example:

```
$("#customerResult")  
.load("/test/MyService", "firstName=Bob;lastName=Smith");
```

20.8 Data Types

- Recall the fourth parameter in our \$.get example: "html"
- This parameter is needed to tell jQuery how to interpret the contents of the response message
- Options (the default is jQuery's best guess)
 - ◇ html
 - ◇ script
 - ◇ json
 - ◇ xml

20.9 The .data() method

- This method is used in the lab for this unit
- It is often convenient to associate data--strings, functions, arrays, objects of any kind--with a particular DOM Element
 - ◇ .data(name, value)
 - ◇ .data(name)
 - ◇ .removeData(name)
- Example: In the lab we will store the most recently selected tab in the div containing tabs

20.10 Summary

- Ajax Overview
- The Ajax Request and the Response
- Ajax With \$.get()
- Easy Ajax with .load()
- The .data() Method

Chapter 21 - Advanced Ajax

Objectives

After completing this unit, you should be able to:

- Use Ajax to implement form submission
- Use `.serialize()` to serialize form parameters
- Determine whether a give example `.load()` invocation will send a GET or a POST
- Choose the proper jQuery method to force either a GET or a POST
- Create a query string from arbitrary data
- Register a function for the `ajaxStart` event and the `ajaxError` event

21.1 A Form Example

- Consider the following

```
<form action="/test/MyService">
  Username: <input type="text" name="user"/> <br/>
  Password: <input type="password" name="password"/>
<br/>
  <input type="submit" value="Login"/>
</form>
```

- If the user enters "Whatever" for the Username and "Secret" for the Password and clicks the Login button, the browser GETs the following URL:
 - ◇ <http://myServer/test/MyService?user=Whatever&password=Secret>

21.2 An Ajax Form Example

- We would like to be able to do the following

```
$("form").submit(function() {
  $("#customerResult").load("/test/MyService", ??????? );
  return false;
});
```

}) ;

- What needs to go where we have written "???????" and how do we get it there?

21.3 Serialize()

- The following is just what we need!
 - ◇ \$("form").serialize()
- If the user enters "Whatever" for the Username and "Secret" for the Password and clicks the Login button, the value of this object is "user=Whatever&password=Secret"
- The serialize() method follows the same rules as normal form submission; the following are omitted
 - ◇ Unchecked checkboxes and radio buttons
 - ◇ Dropdowns with no selections
 - ◇ Disabled controls

21.4 Get vs. Post

- If the Ajax request may change the server state, POST should be used rather than GET
 - ◇ \$.post(...)
- With the load() method, Ajax chooses between GET and POST based on the type of the parameters object
 - ◇ String -> GET
 - ◇ Object hash or Array -> POST
- To convert the form data into an array, use \$("form").serializeArray()

21.5 More on Query Strings

- You can use \$.param() to create your own query string from an Object hash

- Similar to JavaScript encodeURIComponent()

```
var params = { "name" : "John Resig", "rating" : "<WOW>" };  
alert ($.param(params));
```

- Result

- ◇ name=John+Resig&rating=%3CWOW%3E

- Note: We'll discuss object literals in Unit 10

21.6 ajaxStart() and ajaxError()

jQuery triggers the ajaxStart event before sending an Ajax request

You can register a handler with any element

```
$("#msgs").ajaxStart(function() {  
    $(this).text("Sending Ajax request.");  
});
```

Similarly for ajaxError()

21.7 Summary

- Handling form submission with Ajax
- Using serialize() and \$.param() to create properly formatted and encoded query string
- Using Ajax POST as an alternative to GET
- ajaxStart() and ajaxError()

Chapter 22 - Parsing JSON

Objectives

After completing this unit, you should be able to:

- Distinguish between well-formed JSON and JSON that is not well-formed
- Use the \$.getJSON() utility function to read JSON from a server
- Determine the status of a \$.getJSON() request
- Make JSON requests using POST rather than GET
- Handle JSON errors

22.1 JSON

- JavaScript objects are maps containing a collection of *property:value* pairs
- We see this clearly when we create an object from an object literal

```
var imageList = [  
  {"src": "annie.jpg", "caption": "Annie the cat"},  
  {"src": "cave.jpg", "caption": "Sea cave"},  
  {"src": "desert.jpg", "caption": "Neat desert scene"},  
  {"src": "poppy.jpg", "caption": "Poppy field"}  
];
```

- This notation is called JavaScript Object Notation (JSON)
- When JSON is used as an interchange format, stricter rules apply than within a script. To be well-formed JSON, double-quotes are required around each property name as well as around each value.

22.2 Reading JSON from the Server Using Ajax

- Use \$.getJSON(), specifying a URL and a callback.

```
var imageList;  
// ...  
$.getJSON("/lab/image_list.txt", function(data) {  
  imageList = data;  
});
```

```
    alert("Found " + imageList.length + " images");
});
```

- In this example the JSON is coming from a file but in practice it would often be generated on the server.

22.3 Example file contents

```
[
  {"src": "annie.jpg", "caption": "Annie the cat"},
  {"src": "cave.jpg", "caption": "Sea cave"},
  {"src": "desert.jpg", "caption": "Neat desert scene"},
  {"src": "poppy.jpg", "caption": "Poppy field"}
]
```

22.4 Using the Results

- Updating our callback to use the JSON data in the page:

```
$.getJSON("/lab/image_list.txt", function(data) {
    $.each(data, function(index, image) {
        $("<tr><td>" + image.src + "</td><td>" +
            image.caption + "</td></tr>").
            appendTo("#imageTable");
    });
});
```

- The fact that "data" was read in from an external server as JSON is transparent to the function.

22.5 Optimized Version

```
$.getJSON("/test/image_list.json", function(data) {
    var rowHtml = "";
    $.each(data, function(index, image) {
        rowHtml += "<tr><td>" + image.src + "</td><td>" +
            image.caption + "</td></tr>";
    });
    $("#imageTable").html(rowHtml);
});
```

- Getting the JSON response is not any different, just what is done with it
 - ◇ It is processed into the complete HTML and that is appended to the table all at once

22.6 Getting More From the Response

- The callback may have up to three parameters

```
$.getJSON("/lab/image_list.txt", function(data, textStatus,
jqXHR) {
    alert(textStatus);
    alert(jqXHR.getAllResponseHeaders());
});
```

- textStatus is "success", "notmodified", "error", "timeout", "abort", or "parsererror"

22.7 jqXHR Methods

- readyState
- status
- statusText
- responseXML and/or responseText // depending on the format of the response
- setRequestHeader(name, value)
- getAllResponseHeaders()
- getResponseHeader()
- abort()

22.8 POST vs. GET

- Our current example

```
$.getJSON("/lab/image_list.txt", function(data) {
... });
```

uses GET to make the Ajax request for the JSON.

- To use POST instead, you can "roll your own" Ajax invocation; note the fourth parameter.

```
$.post(url, dataToBeSent, function(data,
    textStatus, jqXHR) {
    //...
}, "json");
```

22.9 Invalid JSON

- Often you can trust the data from the server to be well-formed JSON
- If not, read the data as a string and convert it to JSON yourself.

```
$.get("/test/image_list.json", function(data) {
    // clean up the data string as necessary here
    var parsedData = eval(data);
    $.each(parsedData, function(index, item) {
        // Do something
    });
});
```

22.10 Using \$.ajaxSetup()

- \$.ajaxSetup() can be used to configure Ajax in many ways
- For example, the following might be needed to solve a variety of problems reading JSON
 - ◇ Some browsers, depending on the MimeType associated with the file, generate an error if the file begins with [or {, even though it's legal, well-formed JSON
 - ◇ A server might not set the MimeType properly, especially if you use a file with a different extension than .txt

```
$.ajaxSetup({'beforeSend': function(xhr) {
    if (xhr.overrideMimeType)
        xhr.overrideMimeType("text/plain");
    }
});
```

22.11 Summary

- How to read JSON from a server using Ajax
- How to make a POST request rather than a GET
- How to handle invalid data
- How to solve typical configuration problems

Chapter 23 - Animations and Effects with jQuery and jQuery UI

Objectives

After completing this unit, you should be able to:

- Understand applicability of .hide(), .show(), .slideUp(), and .slideDown() operations available in jQuery
- Select an appropriate widget and interaction from the rich pallet of UI elements offered by jQuery UI

23.1 What is jQuery UI?

- jQuery UI (<http://jqueryui.com>) is a collection of of user interface (UI) elements built on top of the jQuery JavaScript Library and used to create highly interactive web applications
- The jQuery UI elements are broken into the following categories:
 - ◇ user interactions
 - ◇ visual effects
 - ◇ widgets (controls)
 - ◇ and themes

23.2 Can I do Animations and Effects using jQuery only?

- Simple (but still effective) animations and effects are possible with:
 - ◇ .hide() / .show() methods with configured animation duration (in milliseconds) passed as a method parameter and additional parameters controlling the animation process (see jQuery API catalog page: <http://api.jquery.com/>)
 - ◇ .slideUp() / .slideDown() methods
 - ◇ Effects:
 - Fading, Sliding, etc.

23.3 Hiding Elements with jQuery

- Most common type of dynamic effect
- Hidden elements are still
 - ◇ In the DOM tree
 - ◇ Returned by jQuery wrapper selectors that match the elements
- Simplest way to hide/show using jQuery:
 - ◇ .hide()
 - ◇ .show()
- We'll discuss more advanced techniques later in this unit

23.4 Using .hide() and .show() in jQuery

- These two Element methods toggle between
 - ◇ display:none
 - ◇ Either
 - display:previously-saved-value
 - If there was a previously saved value, or
 - display: block or display:inline as appropriate for the element type

23.5 Alternating an Element's Visibility in jQuery

- The following code snippet will alternate the visibility of the p HTML element with id "p1" by calling show() and hide() methods attached to it

```
<head>
<script type="text/javascript"
  src="/jquery/jquery-<jQuery version>.min.js"></script>
<script type="text/javascript">
  $(function() {
    $("#toggler")[0].onclick = function () {
      var someElement=$("#p1");
      if (someElement.is(":hidden")) {
```



```
        someElement.show();
    } else {
        someElement.hide();
    }
    }; });
</script></head>
<body>
    <input type="button" id="toggler" value="Toggler" />
    <p id="p1">Hide and Seek</p>
</body></html>
```

23.6 Adjusting the Speed in jQuery

- It's usually better to allow the user to see elements appearing and disappearing
- .hide() and .show() take an optional parameter for speed
- Values
 - ◇ "slow", "normal", or "fast"
 - ◇ Number of milliseconds

23.7 Providing a Handler in jQuery

- To further enhance the user experience we may want to change something on the page when the animation completes
- .hide() and .show() take an optional parameter for a callback
- Example: Add a plus sign after hiding an element

```
$("#id").hide("slow", function() {
    $(this).prev().find("span:first-child").text(' + ');
});
```

23.8 Using .slideUp() / .slideDown() methods in jQuery

- .show() and .hide() scale the elements horizontally and vertically and also adjust the opacity

- To adjust only the vertical dimension and the opacity
 - ◇ Use `.slideUp()` and `.slideDown()`
- To adjust only the opacity
 - ◇ Use `.fadeIn()` and `.fadeOut()`

23.9 jQuery UI Categories

- Interactions:
 - ◇ Draggable, Droppable, Resizable, Selectable, Sortable
- Widgets:
 - ◇ Autocomplete, Button, Datepicker, Dialog, Menu, Slider, Tabs, etc.
 - ◇ **Note:** jQuery UI's Widget Factory allows page authors to create stateful jQuery plugins using the same abstraction as all jQuery UI widgets
- Effects
 - ◇ Color animation, Hide/Show (with more options than their counterparts in jQuery), Toggle (a wrapper around Hide/Show), etc.
- Utilities
 - ◇ Position that helps place an HTML element relative to the window, document, another element, or the cursor/mouse

23.10 jQuery UI Interactions: Droppable and Draggable

- You can implement Drag & Drop functionality in a Web page using the same visual desktop paradigm we are familiar with as follows:
 - ◇ You create a "Droppable" area and a "Draggable" object(s) that can be dropped off when the mouse button is released
 - ◇ Droppable area and Draggable objects can be coded as simple div sections made visible on the page:

```
<div id="draggable" class="c1">  
  <p>Drag me to the droppable area </p>  
</div>
```

```
<div id="droppable" class="c2">
  <p>Drop it here!</p>
</div>
```

23.11 jQuery UI Interactions: Droppable and Draggable

- ◊ In the \$ global handler that is attached to the DOM-ready event, you initiate the drag action by calling the *draggable()* method:

```
$( "#draggable" ).draggable();
```

- ◊ Right after this, you code the droppable (or accept) callback method with any logic that you want to attach to it, for example:

```
$( "#droppable" ).droppable({
  drop: function( event, ui ) {
    $( this ).addClass( "highlighted" )});
```

- ◊ The '*drop*' parameter denotes the type of the event. Other events used in this context are: *over*, *deactivate*, *out*, etc. The *drop* event type is bound to the event handler function via the ':' notation

23.12 Droppable and Draggable More Complete Example

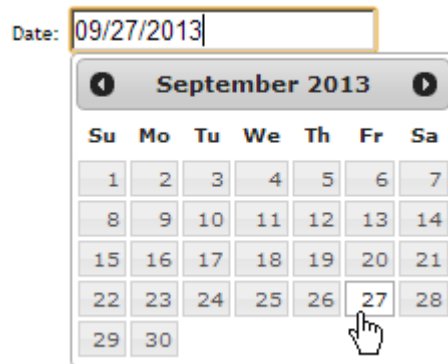
- You will have the following code to do basic drag & drop functionality on the web page using jQuery UI

```
<script>
$(function() // Global handler attached to DOM-ready event
{
$( "#draggable" ).draggable(); // init the drag action
$( "#droppable" ).droppable({
  drop: function( event, ui )
// drop of the draggable event will be handled here
{
    $( this ).addClass( "highlighted" );
// we simply change the style of the drop (this) area
}
// You can have a sequence of events:
// add other events after ',' (coma) here
}
});});
```

</script>

23.13 jQuery UI Widgets: Datepicker

- Datepicker is a very popular jQuery UI widget



- Visually appealing, easy to use and program
- When open, it supports a wide range of keyboard short-cuts, e.g.:
 - ◇ Page-Up – move to the previous month (with a matching Page-Down)
 - ◇ Ctrl-Up – move to the previous week (with a matching Ctrl-Down)
 - ◇ etc.
- Comes complete with an impressive array of customization options and supported operations

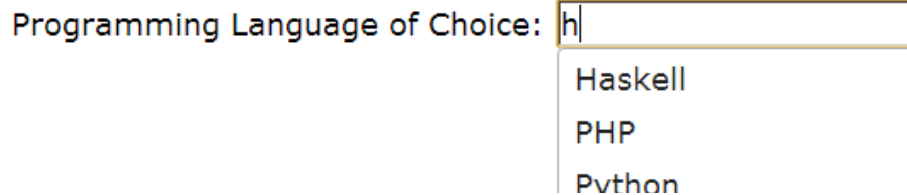
23.14 jQuery UI Widgets: Autocomplete

- There are a number of implementation options:
 - ◇ Combobox, Remote JSON data store (via Ajax calls), etc.
- In its simplest form, it requires:
 - ◇ An HTML input element (e.g. text box) where possible matches are displayed as you key in a sequence of characters
 - ◇ The data source of the available entries (e.g. a list of programming languages)
 - ◇ The jQuery UI initialization code binds the input element's selector to

the data source :

```
$( ".input_selector" ).autocomplete({  
  source: dataArray  
});
```

- ◊ When you start typing into the input element, the autocomplete function continuously performs the matching of characters you entered with entries (words) from the data source (e.g. string array)



23.15 Summary

- Using Animations in jQuery and jQuery UI
- Simple .hide() / .show() and .slideUp() / .slideDown() interactions
- Support for Drag & Drop functionality, Datepicker and Autocomplete widgets offered by the jQuery UI library

Chapter 24 - Plugins

Objectives

After completing this unit, you should be able to:

- Explain what a jQuery plugin is and name the two types
- Write jQuery that will safely execute even if \$ has been redefined by another library
- Use the jQuery prototype object for the wrapper class to add a new wrapper method
- Explain how to make a plugin function support wrapper chaining

24.1 What is a Plugin?

- JavaScript that extends jQuery; typically provided as a .js file.
- Two kinds of plugins:
 - ◇ Add additional utility functions to the \$ object
 - ◇ Add additional methods to operate on a jQuery wrapped collection
- You may write your own or obtain them from others
 - ◇ <http://plugins.jquery.com/>

24.2 Goal

- We want to
 - ◇ Use \$ in our plugin code
 - ◇ Be 100% sure our use of \$ does not conflict with other scripts our plugin user is using
- To achieve these goals, the following idiom is often used for plugin code

```
(function($) {  
    // Code using $ as alias for jQuery  
})(jQuery);
```

- Let's look at that in more detail...

24.3 Self-Executing Anonymous Functions

- First, consider the following, which declares an anonymous function
- This function can't be invoked; we're writing it as a stepping stone to our goal.

```
(function(param) {  
    alert(param);  
});
```

- Now make the anonymous function self-executing
- This function executes itself

```
(function(param) {  
    alert(param);  
})("My alert");
```

24.4 Meeting Our Goal

- An example illustrating scope

```
(function() {  
    var msg = "Local to the function";  
})();  
console.log(msg);    // undefined
```

- One final example
- This construct allows us to meet our plugin goals

```
(function($) {  
    // Code using $ as alias for jQuery  
(jQuery);
```

24.5 Prototype Objects

- JavaScript associates a special prototype object with each class.
- Declare a class

```
function MyWidget() {  
}
```

- Assign a property to the prototype object of the class

- `MyWidget.prototype.color="blue";`
- Demonstrate that each instance of the class has the property we added to the prototype object
- `var mw = new MyWidget();`
- `alert(mw.color);`

24.6 The jQuery Wrapper Class Revisited

- Recall that `$(someSelector)` returns an instance of the jQuery wrapper class
- `$.prototype` is the prototype object for the wrapper class
- Thus, we can add new properties to the wrapper class!
- `$.prototype.myNewMethod = function() { ... };`
- Note: The wrapper prototype object is aliased as "fn" so the following works also:
- `$.fn.myNewMethod = function() { ... };`

24.7 Example Plugin

- Declaring the plugin

```
(function($) {  
    $.fn.myPlugin = function() {  
        this.each(function() {  
            $(this).click(function(e) {  
                alert("Element class: " + this.className);  
            });  
        });  
        return this;    // to support jQuery object chaining  
    };  
})(jQuery);
```

- Using the Plugin

```
$('p').myPlugin();
```

24.8 Summary

- What is a plugin?
- Using a self-executing anonymous function to simplify plugin development
- Prototype objects
- The jQuery wrapper class revisited
- An example plugin
- Using the plugin

Chapter 25 - Introduction to Responsive Web Design

Objectives

Key objectives of this chapter

- Describe Responsive Design
- Review some mobile web considerations
- Review alternatives to Responsive Design

25.1 What is Responsive Web Design?

- "Responsive design" is being able to adapt the presentation based on the user's context
 - ◇ For a web site this is most often the screen size and resolution of the device and browser being used to view the site
 - ◇ It is a combination of techniques to create web pages that optimize themselves to the environment of the end user
- Having one design that is applied to all devices simply won't work
 - ◇ Something that looks fine in a desktop computer monitor will be cramped in a mobile phone or even tablet
 - ◇ Designs that look fine on a mobile phone may vastly under-utilize the space available to a regular computer
- The page needs to automatically scale or rearrange page elements following a planned set of instructions to better fit displays of different sizes and shapes
- With the proliferation of mobile phones and tablets running browsers capable of adequately viewing web pages ignoring this group of client types is not an option either

25.2 Mobile Browsers Quirks

- Mobile browsers have the following limitations that need to be accounted for in design of your web pages:
 - ◇ A smaller real estate

- ◇ Generally slower page rendering times than desktop browsers
- ◇ Varying degree of support for web technologies (HTML, CSS, and JavaScript)

25.3 Other Mobile Web Considerations

- Mobile web page UI design is different from that of the desktop world
 - ◇ More compact, various device orientation, different types of supported interactions
- Mobile devices offer a wider range of input devices
 - ◇ Stylus pens, gestures, etc.
- Mobile networks are less stable
- Data plans may be expensive
 - ◇ Users may stop visiting sites with "heavy" web pages

25.4 Primary Responsive Design Techniques

- CSS3 media queries (<http://www.w3.org/TR/css3-mediaqueries/>)
 - ◇ Apply CSS based on the client browser's system profile (windows aspect ratio, device orientation, etc.)
- Fluid grid layouts (a.k.a *proportional* layouts) lead to more flexible layouts
 - ◇ This is achieved by using proportional units (percentages) instead of fixed sizes in pixels
- Scalable images and media
 - ◇ This is achieved by placing objects in the scalable parent element and sizing them in relative units (e.g. %) preventing them from displaying outside their containing element

25.5 Elements of Responsive Design

- A responsive design often combines several different elements:

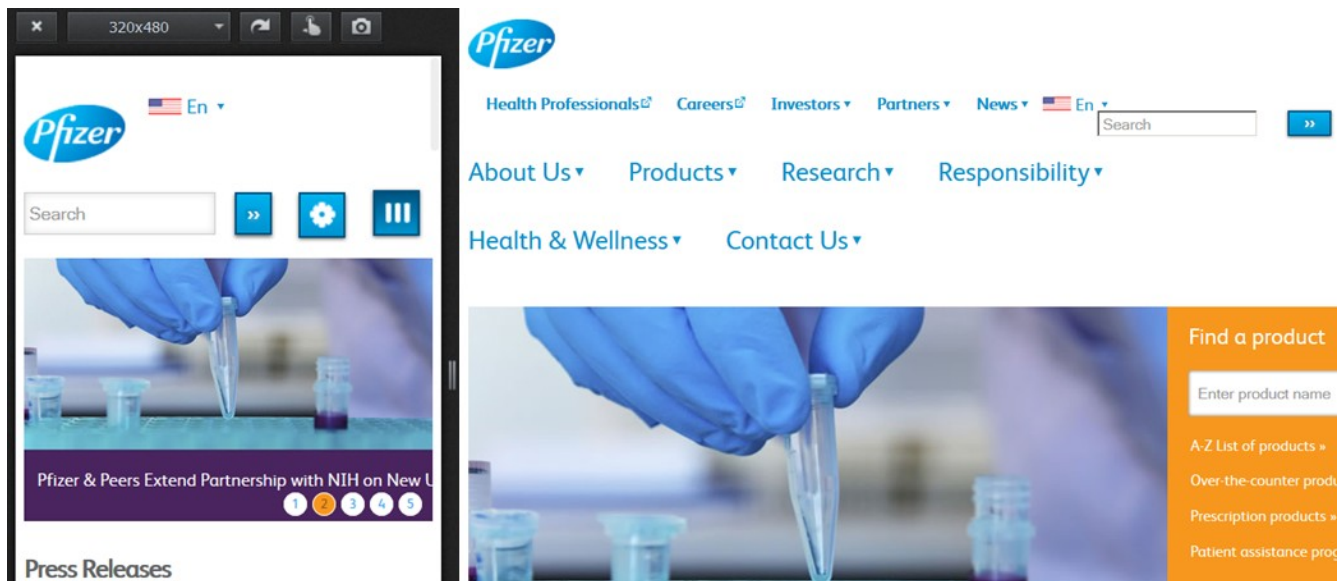
- ◇ **Progressive enhancement** – The design can scale up or down with different layers of styles being applied depending on the environment
- ◇ **Resetting styles** – Manually resetting various style properties so that default style values, which may be different between browsers, are not applied
- ◇ **Conditional styles for Internet Explorer** – Addressing the quirks of older Internet Explorer versions
- ◇ **Adaptive styles** – Avoids the "pixel perfection" of attempting to exactly fix all elements of a page and instead uses a more fluid layout that can stretch or shrink through the use of relative sizes
- ◇ **Media Queries** – Expressions that allow a Web page to conditionally apply stylesheets based on device characteristics, including the device's width, orientation and resolution
- ◇ **Break points** – Points at which the design of the page is more radically adjusted

Elements of Responsive Design

The concept of "adaptive styles" means that almost always you should avoid using a number of pixels to set style properties and use relative values. This can be percentages or the 'em' unit as that is relative to font size.

25.6 Example of Responsive Design

- Side-by-side, left to right: mobile browser and large desktop browser views displaying the home page of *Pfizer* (one of the world's largest pharmaceutical company) created with Responsive design techniques.
 - ◇ **Note:** This is the same web page that responds to the screen resolution of the user browser (no redirects to the "mobile web" site)



25.7 Responsive Page Design Schematic

- Notice the fluid page layout that allows the same page to adapt to the screen sizes of mobile devices, tablets, and desktop browsers without compromising content.



25.8 Alternatives to Responsive Design

- There are certainly alternatives to responsive design:
 - ◇ **Mobile app** – You could develop an application in the native programming language of the various mobile platforms
 - Even just focusing on two main platforms of Android and iOS can get cumbersome and require a completely different skill set from web development

- ◇ **Mobile site** – You could have a completely separate site for mobile or tablet devices and even redirect users to this if you detect they have the correct device
 - This makes it more difficult to maintain the content in two separate sites and cause user frustration if trying to move between the two (showing a friend on your phone the page you found on the laptop)
 - This still tries to apply a "one size" approach to a wide range of devices and may mean some medium-sized devices are under-utilized since the mobile site is designed for the lowest-level browser
- Both of these strategies have been widely employed before the emergence of HTML5 and CSS 3
- ◇ With the advancement of HTML and CSS, responsive design is becoming as the preferred approach

25.9 Summary

- Responsive design is the approach of allowing the design to adapt to differing presentation environments
- Responsive design has become the preferred way to handle the growing number of browsers and devices that are being used to view web sites
- There are alternatives to responsive design:
 - ◇ Mobile apps development in the native programming language of the various mobile platforms
 - ◇ A separate mobile site serving content specific for mobile devices

Chapter 26 - CSS 3 and Responsive Web Design

Objectives

Key objectives of this chapter

- Describe Progressive Enhancement Design
- Use media type and media queries
- Specify viewport properties

26.1 Progressive Enhancement

- Progressive enhancement is the acceptance that the design may look slightly different in different browsers that vary in capability
 - ◇ Trying to provide the "pixel perfect" design that looks the same in all browsers is simply not possible anymore with the range of versions of desktop browsers and the introduction of mobile browsers
 - ◇ This will use newer technologies like CSS 3 where available but "gracefully degrade" to a default design in the absence of support
- Some basic guidelines help to apply progressive enhancement:
 - ◇ Basic content is available to all browsers
 - ◇ Basic functionality is available to all browsers
 - ◇ Enhanced layout is provided by external CSS style sheets
 - ◇ The custom styles of end users should be honored

Progressive Enhancement

Browsers will let users indicate a custom style sheet that can override styles of a page. This is often for making sure a page is readable for those with poor eyesight.

26.2 Implementing Progressive Enhancement

- Progressive enhancement can be implemented by applying default styles that will be used for all browsers and then conditionally overriding those styles for browsers that support the additional features

- An example of progressive enhancement could be:
 - ◇ The page is appropriate for mobile browsers without any extra styles applied
 - ◇ More complex navigation or multiple columns can be applied when a wider browser is detected
 - ◇ The features of modern browsers are leveraged by using CSS 3 styles which are just ignored by older browsers

26.3 Media Types

- The <link> tag contains a 'media' attribute that can be used to determine which type of media the style sheet should be applied to
 - ◇ Different <link> tags with different 'media' attribute values can link to style sheets with style properties unique to that media type to provide differences in the design

```
<link rel="stylesheet" media="all" href="default.css" />
<link rel="stylesheet" media="screen" href="view.css" />
<link rel="stylesheet" media="print" href="print.css" />
```

- Some of the most common media types used are:
 - ◇ all – all devices
 - ◇ screen – computer displays
 - ◇ print – paper
 - ◇ handheld – portable phones and PDAs although many modern smartphones respond to 'screen' and not 'handheld'
- You can also use the '@media' rule syntax within a <style> tag or style sheet with the media type between the @media and the curly brackets separated by space

```
@media screen { ... }
```

Media Types

Other less common media types are:

- braille - braille tactile readers

- embossed - paged braille printers
- projection - projectors
- speech (CSS 3), aural (CSS 2) - speech synthesizers
- tty - teletypes, computer terminals and older portable devices
- tv - television displays

Although it might be nice to be able to use 'handheld' to apply styles to all mobile phones, smartphones have higher resolution and zoom capability compared to older phones and PDAs. You can use 'screen' and various media query expressions to find the size of the screen to apply different styles to these.

26.4 CSS Style "Reset"

- In the absence of explicit style property values supplied by you, browsers will supply default values to things like margin, padding, etc
 - ◇ Different browsers may provide different defaults which will show up in your design as small differences depending on the browser used
- To avoid this you can apply a style sheet to "reset" the various style attributes and override the browser defaults
 - ◇ You will put this style sheet <link> tag first in the <head> section as you will want other style sheets listed later to override these "reset" values
- Although you can come up with your own reset style sheet there are a few popular ones

<http://yuilibrary.com/yui/docs/cssreset/>

<http://meyerweb.com/eric/tools/css/reset/>

<http://html5doctor.com/html-5-reset-stylesheet/>

- Instead of modifying the "reset" style sheet for your particular site, just link additional style sheets with your styles

```
<link type="text/css" rel="stylesheet" media="all"
      href="cssreset-ericmeyer.css" />
```

```
<link rel="stylesheet" media="screen" href="default.css" />
```

26.5 Conditional Styles for Internet Explorer

- Internet Explorer, especially in early versions, was an outlier in how styles were interpreted in sometimes non-standard or buggy ways

- ◊ For a web site that may be used by a wide range of users it is still important to support Internet Explorer so some style "fixes" may need to be applied just to it
- Internet Explorer has the ability to evaluate conditional statements that are otherwise ignored as comments by other browsers
 - ◊ We can use this to define style rules or even JavaScript that will only be seen and applied by Internet Explorer
 - ◊ Conditional statements are no longer supported by IE 10 and later
- To use this you use a '[if <expression>]' within a starting comment tag and a '[endif]' within a closing comment tag

```
<!--[if IE 8]> - only IE 8
```

```
...
```

```
<![endif]-->
```

- In addition to 'IE' and an optional version the '<expression>' can have various operators (lt – less than, lte – less than or equal, gt, gte, etc)

```
lte IE 8 - Less than or equal to IE 8
```

```
!gt IE 6 - Not greater than IE 6 (also lte IE 6)
```

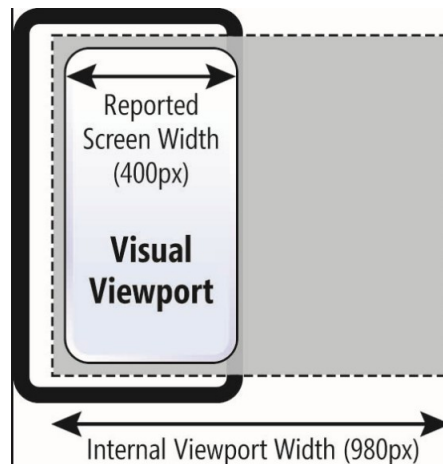
Conditional Styles for Internet Explorer

Although support for conditional statements was removed in Internet Explorer 10, this does not have much impact. Internet Explorer 10 and later have good support of HTML5, CSS3 and many modern standards so the need to adapt specifically to these later versions is much less.

26.6 What is the Viewport?

- The "viewport" is something slightly different on desktop and mobile browsers
 - ◊ On a desktop browser, it is the width of the visible window
 - ◊ On a mobile browser, it is the width used to determine the page layout and can be wider than the visible window
- The difference in the viewport between desktop and mobile browsers is because mobile browsers assume that web pages are designed for a desktop and will be "wider" (~1000 px) than the mobile device could display without zooming out

- ◇ The "scale" of the viewport will control this zooming in or out



26.7 Adapting the Viewport

- The first step in responsive design is to specify settings for the viewport so that mobile browsers in particular are aware that the page will adapt to the differences of the display
 - ◇ Generally it is enough to set the width of the viewport to the width of the device and start the scale at '1'
 - ◇ Although it is possible to disable being able to zoom in and out this is generally discouraged because mobile users will expect to have this ability
- If you do set viewport properties it is important to make sure the page adapts to the device or your page may start zoomed in to the top left corner in mobile browsers

Adapting the Viewport

Instead of disabling the ability to zoom altogether, it is possible to set minimum and maximum values for what the zoom level can be. This would let some users do some zooming and perhaps devices to auto-zoom on page elements like form fields but not be too extreme.

The default behavior of most mobile browsers in the absence of viewport properties specified by the page, is to assume a width of the page as around 980-1000 pixels as the "viewport". This is then fit within the visible window (which is smaller than 1000 pixels) by zooming out. If the user zooms in, the "viewport" remains the 1000 pixels but the user sees less of it at a time.

26.8 Specifying the Viewport

- The method currently recognized by the widest number of browsers is a viewport "meta" tag
 - ◇ Properties of the viewport can be set with the 'content' attribute
 - width – most commonly set to 'device-width'
 - initial-scale – most commonly set to '1'
 - maximum-scale and minimum-scale – less than 1 is zoomed out
 - user-scalable – default is 'yes', setting to 'no' not suggested

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

- A '@viewport' style rule is being developed as part of the CSS Device Adaptation standards and will start to be recognized by more browsers
 - ◇ Specifying viewport properties uses different syntax
 - 'zoom' instead of 'initial-scale'
 - 'max-zoom' instead of 'maximum-scale', also 'min-zoom'
 - 'user-zoom' with values of 'zoom' and 'fixed' instead of 'user-scalable'

```
@viewport { width: device-width; zoom: 1; }
```

Specifying the Viewport

Multiple viewport properties specified with the meta viewport tag are separated with spaces in the 'content' attribute.

The most restrictive viewport would be one that doesn't allow any zooming. This may be too restrictive for mobile browsers.

```
<meta name="viewport" content="width=device-width, initial-scale=1,
      maximum-scale=1, minimum-scale=1, user-scalable=no" />
```

The following meta viewport tag will allow some zooming but set limits.

```
<meta name="viewport" content="width=device-width, initial-scale=1,
      maximum-scale=3, minimum-scale=0.25" />
```

Work on the CSS Device Adaptation standard is being done here:

<http://www.w3.org/TR/css-device-adapt/>

The '@viewport' style rule is not a finalized standard but it is already recognized by some browsers. There are also browser extensions of this rule recognized by the latest Internet Explorer

(@-ms-viewport) and Opera (@-o-viewport) browsers. Specifying the viewport so it would be recognized by every browser could be something like:

```
<meta name='viewport' content='width=device-width, initial-scale=1' />
```

```
<style type='text/css'>
@-ms-viewport { width: device-width; zoom: 1; }
@-o-viewport { width: device-width; zoom: 1; }
@viewport { width: device-width; zoom: 1; }
</style>
```

A '@viewport' rule that would disable zooming would look like:

```
@viewport { width: device-width; zoom: 1; min-zoom: 1;
            max-zoom: 1, user-zoom: fixed; }
```

26.9 Media Queries

- CSS 3 added the ability to do "media queries"
 - ◇ These are expressions that are more complex than just the media type and can include various "media features" like width, height, aspect ratio, orientation, screen resolution and color
- These media queries can determine if a particular style sheet is applied for a user's device and establish the "break points" that can define more substantial changes in design
- The '**media**' attribute of the <link> tag can contain multiple "media features" combined with 'and' to create a more complex expression

```
<link rel="stylesheet" media="screen and (min-width: 980px)
                               and (min-device-width: 980px)" href="large.css" />
```

- You can also use media queries with the '@**media**' style rule syntax
 - ◇ The expression simply comes between the '@media' rule and the curly brackets for the applied style properties for the rule

```
@media screen and (min-width: 980px) and
               (min-device-width: 980px) { ... }
```

- You can separate multiple expressions with a comma and the style sheet or style rules will be applied if any of the comma-separated expressions is true

Media Queries

Browsers that do not support CSS 3 will not be able to evaluate media queries and will ignore the entire

<link> tag. It is important to include default style sheets with <link> tags that do not have media queries (and just media type) so that these browsers will still see a basic display of the site.

It is also true that any browser that supports media queries supports CSS 3 in general so you can include CSS 3 style properties in style sheets linked to by media queries. Remember that since CSS 3 was developed in modules not all browsers support all style properties but you at least know you are working with a relatively modern browser.

26.10 Media Features Used in Media Queries

- There are many different media features that can be used in a media query
 - ◇ Many of these have 'max-' or 'min-' prefixes
 - ◇ Features which include 'device' apply to the device instead of the viewport
- Some of the media features that can be used in expressions:
 - ◇ size – width (min/max also), device-width (min/max also), height (similar)
 - ◇ aspect ratio – aspect-ratio (min/max also), device-aspect-ratio (min/max also)
 - ◇ orientation – values of 'portrait' or 'landscape'
 - ◇ resolution – resolution (min/max also)
 - ◇ color - color (min/max also), color-index (min/max also), monochrome (min/max also)

```
<link rel="stylesheet" media="screen and (min-width: 740px)
    and (min-device-width: 740px),
    (max-device-width: 800px) and (min-width: 740px)
    and (orientation:landscape)" href="medium.css" />
```

Media Features Used in Media Queries

For media queries, 'width' is the size of the viewport width, while 'device-width' is the width of the screen. If you did not use a meta viewport tag to set 'width=device-width', then the current 'width' is likely larger than the 'device-width' for mobile browsers, and media queries may not behave as expected.

In the example above, there are actually two different media query expressions separated by a comma.

Only one of these expressions needs to evaluate to true for the linked style sheet to be applied to the page. Each expression is made up of several conditions combined with 'and' which would all need to be true for the entire expression to be true.

Some of the listed features above have the 'min-' and 'max-' versions combined to shorten the overall list. These are separate features that can be tested against.

The most common media features used in media queries are 'width' and 'device-width' with some number of pixels, and the 'orientation' feature.

The 'color' media feature describes the number of bits per color component of the output device. If the device is not a color device, the value is zero. The 'color-index' media feature describes the number of entries in the color lookup table of the output device. If the device does not use a color lookup table, the value is zero. The 'monochrome' media feature describes the number of bits per pixel in a monochrome frame buffer. If the device is not a monochrome device, the output device value will be 0.

26.11 Combining Responsive Design Techniques

- The assumption that you will need to support mobile devices and older Internet Explorer versions do not pose any challenges
 - ◇ Loading fewer style sheets in a mobile device will improve performance
 - ◇ Older IE versions and mobile devices do not support CSS 3 media queries
 - Internet Explorer 9 was the first to support media queries
- A "Mobile First" strategy where the styles loaded without CSS 3 media queries will provide a layout for mobile devices will provide good mobile device support
 - ◇ A "Mobile First" strategy is designing the page with the mobile environment in mind and applying styles to take advantage of larger devices
 - ◇ The problem is that older IE versions would look the same as mobile devices
- To add better support for older Internet Explorer browsers you can use the conditional technique to load the same style sheet you would have loaded for desktop-size browsers with media queries
 - ◇ You might load a "medium" layout as resizing the window of older IE versions will not trigger the media queries recognized by other

browsers

- ◇ Loading the same style sheet you would have with media queries will make the site easier to maintain

Combining Responsive Design Techniques

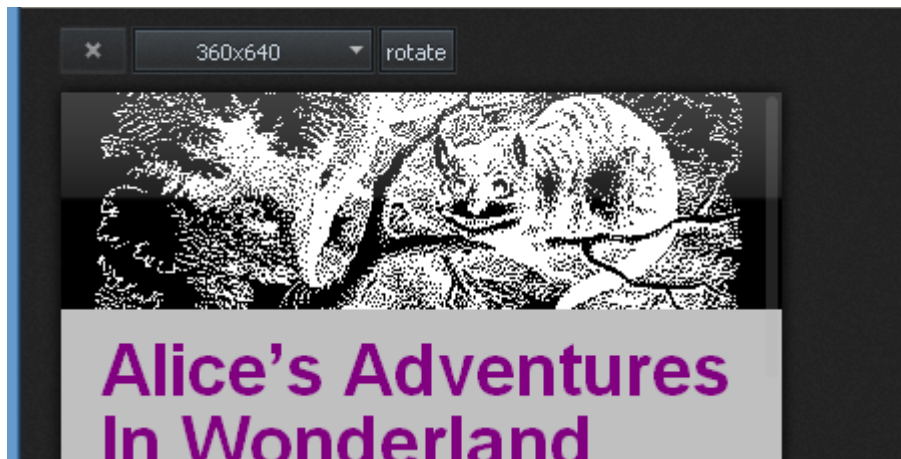
The main issue with loading the "full" styles using conditional IE statements is that the browser will not respond to resizing the window as other browsers would. Using other browsers that recognize CSS 3 media queries, as you shrink the window size, the media queries will trigger a different style for this reduced size. Older Internet Explorer versions will not do this. The best approach might be to load a "medium" layout that would look good at 700-800 pixels and still be OK on slightly larger window sizes. After all, the likelihood someone will be using IE 6 with a high resolution monitor is probably small!

One problem with using conditional IE statements is that they don't support browser resizing. However, there are some polyfill scripts (e.g., Respond) which provide CSS 3 media query support to older versions of IE (6-8) and do support browser resizing:

<https://github.com/scottjehl/Respond>

26.12 Testing Responsive Design

- Several browsers have built-in tools that will allow you to test what a page will look like with different screen resolutions or even interacting with a mobile client
 - ◇ Firefox has the 'Responsive Design View' which lets you change the screen resolution



- ◇ Chrome has 'Mobile Emulation' tools that will also let you emulate

things like mobile touch events and Geolocation

Testing Responsive Design

https://developer.mozilla.org/en-US/docs/Tools/Responsive_Design_View

<https://developers.google.com/chrome-developer-tools/docs/mobile-emulation>

26.13 Summary

- There are a few simple techniques that can allow you to specify style rules for specific client types like old versions of Internet Explorer and modern mobile browsers
- Specifying viewport properties can allow you to control how the browser, especially mobile browsers, initially display your web page
- CSS 3 media queries can be used to apply different styles based on the features of the media device and establish "break points" where the design is modified more significantly to adapt to the various devices

Chapter 27 - Responsive Web Page Layout

Objectives

Key objectives of this chapter

- Quickly review the main layout types
- Review most popular responsive layouts
- Demonstrate a simple technique for building a responsive page layout

27.1 The Main Layout Types

- **Fixed-width:** The simplest layout that is normally used for desktop monitors; as the name suggests, it is totally unresponsive
- **Fluid (Liquid):** Dimensions in a fluid (liquid) layout are specified as percentages; the content expands and contracts as the browser window is resized. In some pages, you may want to "cap" the fluidness of the page by setting the **max-width** and **min-width** CSS properties
- **Elastic:** Dimensions in an elastic layout are set using **ems** (scalable CSS font size units) which results in keeping everything in proportion with the text size
- **Hybrid:** A combination of different layouts (e.g. keeping the core content in a fixed-width layout and letting the rest of the content adapt to the screen size using a fluid or elastic layout)

27.2 Responsive Layouts

- Many of the Responsive Design fluid layouts respond to changed screen sizes using old web design tricks, e.g. the CSS **float** property that has been available since CSS1
- You have a number of layout patterns to choose from depending on your content
- The Responsive Patterns web site (<http://bradfrost.github.io/this-is-responsive/patterns.html>) offers a wide collection of solutions

Layout

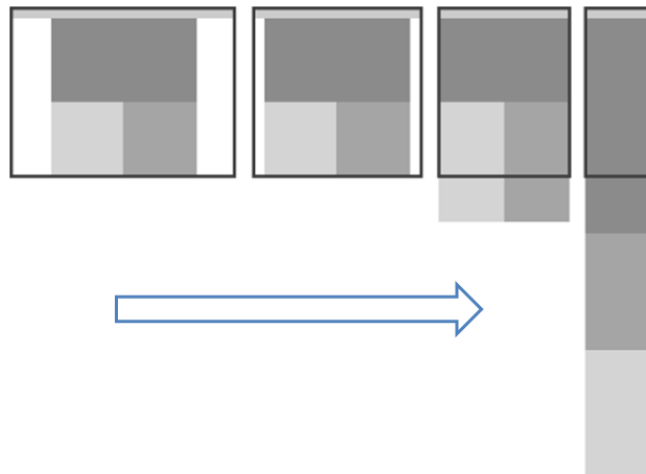
Reflowing Layouts	Equal Width
Mostly Fluid	2 equal-width columns
Column Drop	3 equal-width columns
Layout Shifter	4 equal-width columns
Tiny Tweaks	5 equal-width columns
Main column with sidebar	6 equal-width columns
3 column	
3 column v2	
3 Columns content reflow	
Source-Order Shift	Lists
Table Cell	List with Thumbnails
Flexbox	List with Thumbnails 2

27.3 Popular Layout Patterns

- We will review the following popular layout patterns:
 - ◇ Mostly Fluid
 - ◇ Column Drop
 - ◇ Layout Shifter
- Essentially, they mostly work by stacking page elements vertically on smaller screens forcing the user scroll for content

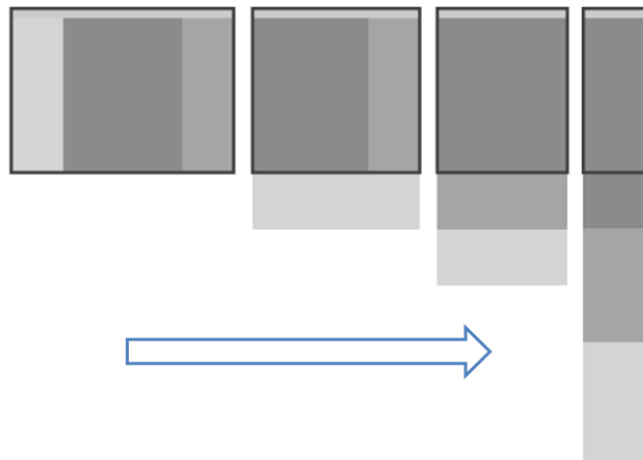
27.4 The 'Mostly Fluid' Layout Pattern

- One of the most popular layout pattern
- Also one of the simplest: a multi-column web page layout on large screens scales down to small screen sizes by stacking the columns vertically at different screen resolution breakpoints
- The core structure of the layout is maintained until the smallest screen width



27.5 The 'Column Drop' Layout Pattern

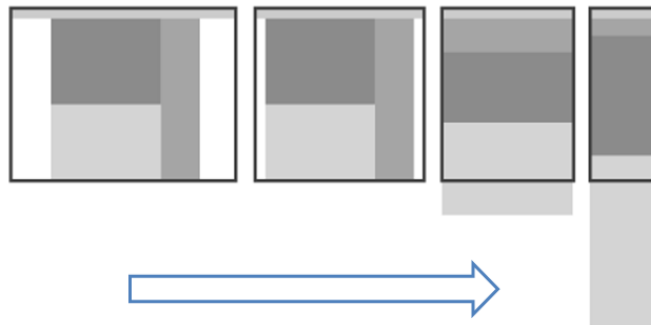
- Also a very popular pattern
- Like the Most Fluid pattern, it starts off with a multi-column layout but start "dropping" columns (moving them down to the bottom of the page) as screen sizes get smaller
- Unlike the Mostly Fluid pattern, the layout tries to maintain consistent overall size of the page elements



27.6 The 'Layout Shifter' Pattern

- The more complex layout pattern, and, therefore, less popular one

- Generates different page layouts for different screens (at specific resolution breakpoints)

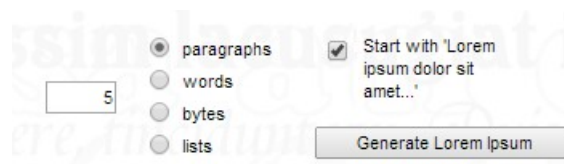


27.7 Other Layout Techniques

- There is a wide variety of techniques used by designers to compensate for smaller device screens
 - ◇ Collapsible / expandable elements (e.g. navigation)
 - ◇ Layered-over elements (e.g. for user input)
 - ◇ Drop-down menu boxes (e.g. for user choices)
 - ◇ etc.
- Such techniques require some user interaction (*taps, drag, spread / pinch*, etc.) as supported by devices with touch screens

27.8 Getting Content Fillers

- For testing layout of your pages, you may need some random text as a filler
- **Lorem Ipsum** (<http://www.lipsum.com>) to the rescue!
 - ◇ ... *Aliquam et leo scelerisque, sagittis leo at, suscipit* ...
- It is the typesetting industry's standard for dummy filler text since the **1500s**
- The filler text has roots in classical Latin literature
- The *Lorem Ipsum* web site offers options to generate lists, paragraphs, etc. of filler text



Notes:

Note 1: Here is a sample Lorem Ipsum list:

- Mauris ac sem quis quam ullamcorper pellentesque.
- Vestibulum pharetra tortor eu facilisis pretium.
- Mauris quis turpis ac lectus ullamcorper facilisis non commodo mauris.
- Vivamus rutrum ipsum at pharetra luctus.

You can use it on a slide to check if the audience is still following your Power Point presentation.

Note 2:

Interestingly enough, the above text (in Latin) *Aliquam et leo scelerisque, sagittis leo at, suscipit* is translated into English by the Google on-line translation service as *Development and evaluation of thermal, stress and a mini-*

27.9 The Float CSS Property

- Many of the fluid layout techniques are based on the **float** CSS property
- The **float** CSS property assigned to an element takes it from the normal page rendering flow and places it along the left or right side of its container
- Syntax:

```
float: left | right | none
```

- Example:

```
.myFluidClass {float: left;}
```

Notes:

JavaScript refers to the **float** CSS property as the **cssFloat** property of the [element].style object, for example:

```
document.getElementById("myFluidImage").style.cssFloat="right";
```

The above command will flush the element referenced by the `myFluidImage` id to the right of its container.

27.10 Combining CSS Styles

- It may be useful to keep the floating aspect of your page design in a separate class and add it to elements as an additional (floating) style rule as needed

- Example:

- ◇ The primary (regular) style:

```
.col_3 {  
    color: blue;  
    padding: 1em;  
}
```

- ◇ The floating style (a separate class):

```
.floatLeft {  
    float: left;  
}
```

- ◇ Combining the classes to add the floating aspect to the *div* element:

```
<div class="col_3 floatLeft">
```

27.11 The Simple Fluid Layout Example

- Let's say the wireframe for the page requires you to have three text columns that has to stack up as the screen gets narrower
- You can build such a page with a fluid layout using the following simple technique (see next slide)
- **Note:** This example is for illustration purposes used to highlight some of the main points covered so far

27.12 The Simple Fluid Layout Technique

- Define a style class that includes the **float** style property (usually, it is set to '*left*'), e.g.

```
.txt_col {  
    max-width: 300px;  
    float: left; }
```

- Place your text in three sequential div's
- Apply the above style to all the three div sections used as containers of your column text

```
<div class="txt_col">The left column's text goes here ...</div>  
<div class="txt_col">The middle column's text goes here ...</div>  
<div class="txt_col">The right column's text goes here ...</div>
```

- Load the page in the browser

27.13 The Results

- Now, when the screen width gets narrower than a size of around $3 * 300 = 900$ px (the width at which all the three columns can be displayed in one row), the rightmost column will be dropped
- At a screen width of about $2 * 300 = 600$ px, the middle column will be dropped and inserted between the left and the right column (now at the bottom of the page)
- This technique works using the 'Column Drop' layout pattern

27.14 Font Size Units

- In your web pages, you can measure the size of your fonts in the following most common typographic units:
 - ◇ Pixel
 - ◇ Percentage (%)
 - ◇ Pt
 - ◇ Em

27.15 Pixel-Sized vs Em-Sized

- Setting sizes in pixels gives you precise control over font sizes
 - ◇ If you set a font-size to 20px, all browsers will display fonts at exactly that size (including IE!)
- Sometimes it is not what you want since

- ◇ Pixel-sized fonts don't support cascading - the font size of the parent element does not cascade to (has no effect on) the font sizes of the child elements. That poses a major web site maintenance headache as you are forced to change all font sizes to scale the page typography up or down
- This limitation is not applicable to **em**'s which are resizable across all browsers
- **Em**'s also support cascading - you may only want to set the initial baseline, and the rest of the content will fluidly maintain font relative proportions

27.16 Font Size Unit Relationships

- By default on most browsers, if you haven't set the font size, the font size is set at 16px which is treated as a 100% size:
$$1\text{em} = 100\% \approx 16\text{px} \approx 12\text{pt}$$
- So, 1.5em would be about 24 (16 + 16/2) px or 150% of the base font size
- **Note:** em's help create a fluid (scalable) version of your texts

Notes:

There is no major difference between **em**'s and percentages (%), however, em's are becoming the prevailing unit of measurement for fonts.

There are some exceptions, however.

For example, using the em measurement unit to set the base font-size for your web page works for all browsers, except for Microsoft Internet Explorer which, in this context, sets up font scaling ratios different from other browser, making the page fonts look disproportionately larger.

Web designers work around IE's "font exaggeration" bug by setting the base font-size on your body using a percentage:

```
body {  
    font-size: 100%;  
}
```

27.17 Pixels to Em Conversion Formula

- Things get a bit complicated, if you do set a font-size (normally done on the *body* element)
 - ◇ So if the font size on your page is set at, say, 20px then 1.5em will be equivalent to about 30px

- Use this formula to find the *em* equivalent for any pixel value:

`em value = target element pixel value / parent element font-size in pixels`

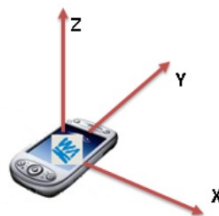
27.18 Other Considerations

- With mobile devices, you also need to be aware of mobile specific usability aspects which are not present in the desktop computers:
 - ◇ Device (and its embedded browser) may change its orientation (portrait to landscape and back)
 - ◇ Device may offer a fine-grain control over its 3D orientation based on its embedded accelerometer readings

Notes:

What is Accelerometer?

The accelerometer is a motion detection sensor that captures device movement relative to the current device orientation in a three-dimensional space (along the X, Y, and Z coordinates).



Determining device's 3D motion involves calculating the difference (delta) between subsequent orientation measurements along each coordinate.

Accelerometer readings are also used in detecting device orientation, e.g. whether the device screen is in the portrait or landscape view.

In some systems, the α , β , and γ characters are used for representing coordinates, where α is Z, β is X, and γ is Y coordinate in the XYZ coordinate notation system.

27.19 Looking into the Future

- Web designers have always used tables for multi-column layouts; this method is now regarded as legacy and layouts using <div> blocks are the preferred method
 - ◇ Table markups are still valid and should be used for displaying data in tabular format
- Back in 2011, W3C published its *CSS Multi-column Layout Module* document (<http://www.w3.org/TR/css3-multicol/>) which puts forward multi-column layout semantics recommendations based on CSS rules
- Browsers implementing this CSS module will, for instance, be able to automatically add / remove columns depending on the screen width
- For example, the following rule will set the *body* element to have as many columns as fit on the screen, each of which being at least 12em wide:

```
body { column-width: 12em }
```

27.20 Summary

- There are several page layout types that you can use in your web page design:
 - ◇ Fixed-width
 - ◇ Fluid (Liquid)
 - ◇ Elastic
 - ◇ Hybrid
- There are also a number of popular responsive layout patterns that allow for adapting the page content to the screen size in a fluid way
- We also looked at the basic steps needed to implement the 'Column Drop' layout pattern using the **float** CSS property

Chapter 28 - Responsive Images

Objectives

Key objectives of this chapter

- Responsive images considerations
- Overview of performance aspects
- The HTML5 picture element
- The Picturefill library

28.1 Responsive Images

- When working with images in responsive web pages, web designers consider these two aspects of responsive images:
 - ◇ Image weight optimization for better page performance
 - ◇ Appropriate resolution and sizes for target device screens

28.2 Performance Considerations

- Images make your web pages significantly bigger in size
- Mobile networks have less throughput than their cable counterparts
- Mobile devices have slower page rendering times than those of desktop browsers
- So, images may have a serious impact on page load time, particularly, on mobile devices and performance analysis involving images is required
 - ◇ which includes analysis of the need for an image on the page in the first place, its visual impact, and its size
- Smaller image sizes help reduce page load time, conserve bandwidth and reduce storage needs on the target device

28.3 Shrinking Images

- This is a "take the problem by the horns" approach

- There are a number of tools available to help you place your images on a diet
- The TinyPNG (<https://tinypng.com>)
 - ◇ It uses a lossy compression algorithm to reduce the size of your PNG and JPEG files by selectively reducing the number of colors in the image. You can reduce PNG image sizes by more than 70%!
- JPEGmini (<http://www.jpegmini.com>)
 - ◇ Free trial is available

28.4 Traditional Image Handling Techniques

- The standard techniques, like the CSS **display: none** style rule only suppresses the view of the image on the page that is still requested and downloaded (and then hidden) by the browser
- Other techniques, while solving some parts of the problem, introduce other, e.g. complexity of the solution, maintenance difficulties, etc.
- Making images respond to the ambient browser environment needs a better solution

28.5 Media Queries Don't Always Help With Performance

- Media queries with the image **width** style parameter , e.g.

```
@media screen and ( min-width:301px ) and ( max-width: 500px ) {  
    img.responsive_image { width: 200px; }  
}
```

still result in the image download (only image resizing ratio will be applied)

- Media queries can only help with background images that can be wrapped into a CSS file which would contain a reference to the background image, e.g.

```
body {background: url('small_image.gif') no-repeat 0 0;}
```

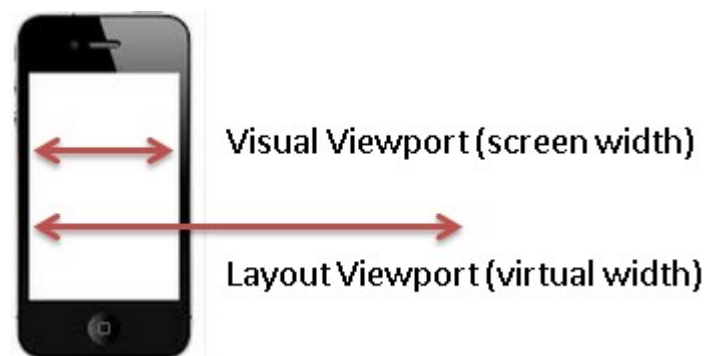
and loading that CSS file based on the media query rule match, e.g.

```
<link rel="stylesheet" media="screen  
and (max-device: 320px)" href="small_background_image.css" />
```


28.6 A "Fluid" Pixel

- In the browser world, pixels come in two types:
 - ◇ device pixels and CSS pixels
- Device pixels are physical pixels as we know them
- CSS pixels are more "fluid" to account for the differences in the visual viewport width of the screen on mobile devices and the (virtual) layout viewport, as well as screen pixel density
 - ◇ You will remember that different viewports allow viewing desktop browser-oriented web pages on smaller mobile devices
 - For example, the iPhone has a layout viewport width of 980px and Android WebKit has 800px

28.7 The Device Viewports



28.8 CSS Pixels

- On mobile devices, CSS pixels are defined within the visible area of the screen (the Visual Viewport), and not in the layout viewport
- So, CSS pixels may not be the same as device pixels
 - ◇ This is the case on some high-resolution displays, such as the Retina display of the iPhone where one CSS pixel is actually equal to two device pixels

- The physical device pixel count stays the same while the CSS pixel count changes
- The CSS pixels changes when a user zooms in or out of a page
 - ◇ For example, a 200% zoom results in twice the number of CSS pixels in the height and the width

28.9 The Power of Simplicity

- The following CSS style rule will make your image fully resizable and fluid:

```
img {  
    max-width: 100%;  
}
```

- It will also (in most cases) automatically maintain the width / height ratio, so you don't need to add the following CSS rule:

```
height: auto;
```

- You can change the max-width value to a smaller than 100 value to use only part of the screen real estate, e.g. 50% to take half of the screen

28.10 Using Google App Engine to Resize Images

- The Google App Engine cloud platform includes an Images Service API that can be used for resizing images.
- The service allows developers to:
 - ◇ resize, rotate, flip, and crop images; build composite multiple images into a single image as well as convert image data between several formats and much more.
- You can upload images to your App Engine cloud application which will contact the Images Service through its API, or you can use Google Cloud Storage as the source of the images for processing.
- The service accepts image data in the JPEG, PNG, WEBP, GIF (including animated GIF), BMP, TIFF and ICO formats

Notes:

<https://developers.google.com/appengine/>

Here is a Java example of using the Google App Engine's Images Service for resizing an image to 200 px by 300 px.

```
import com.google.appengine.api.images.Image;
import com.google.appengine.api.images.ImagesService;
import com.google.appengine.api.images.ImagesServiceFactory;
import com.google.appengine.api.images.Transform;

// ...
byte[] oldImageData; // ...

ImagesService imagesService = ImagesServiceFactory.getImagesService();

Image oldImage = ImagesServiceFactory.makeImage(oldImageData);
Transform resize = ImagesServiceFactory.makeResize(200, 300);

Image newImage = imagesService.applyTransform(resize, oldImage);

byte[] newImageData = newImage.getImageData();
```

28.11 The Picture Element

- Support for adaptive images is part of the current HTML5 specification.
- An adaptive image is represented by the *picture* element used for displaying an image that can come from a range of sources
- Which image the browser, or, more generically, a user agent displays depends on the algorithm for deriving the source image
- User agents that don't support the *picture* element can fallback on the standard legacy *img* element equipped with additional attributes

The Picture Element

Originally the 'picture' element was being developed as an HTML extension. It was decided to include it in a future version of the core HTML specification instead.

<http://www.w3.org/TR/html-picture-element>

28.12 The srcset Attribute

- The *picture* (and *img*) sources under the new spec will have the **srcset**

attribute to specify a list of sources for an image

- Images are transparently fetched from the target server based on the pixel density or size of the user's screen
- Selecting appropriate resources before fetching them from the server helps avoid unnecessary image assets downloads
 - ◇ A user agent (e.g. browser) can use some heuristics to smartly pick up the image to download (e.g. falling back on low-resolution images in situations with slow or unstable network services)

28.13 More on the `srcset` Attribute

- The `srcset` attribute includes a list of comma-separated values which combines image URLs and the conditions under which the image will be fetched from the server (and shown)
- The list of proposed conditions include: pixel density and viewport width of the device screen:
 - ◇ The *width* descriptor is a positive integer directly followed by '**w**'. The default value, if missing, is the infinity
 - ◇ A *pixel density* descriptor is a positive floating number directly followed by '**x**'. The default value, if missing, is 1x

28.14 Examples of the `srcset` Attribute

```

```

- The above code (which uses the *img* element, rather than *picture*) instructs the browser to download and show the image *small_image.png* unless the device has a high-resolution display with a double pixel density (e.g. the iPhone's Retina display)

```

```

- The above line sets the rule (similar to Media Query rules) to use the

small.jpg image on devices with screens up to 480px and the *large.jpg* image on device screens up to 1024px; otherwise, use the *default.jpg* image

28.15 What is Picturefill?

- Older browsers may not support certain HTML5 features (like the *Picture* element). In those cases you can use the "Picturefill" library to support the feature.
- Picturefill is a JavaScript responsive image polyfill library (<http://scottjehl.github.io/picturefill>)
 - ◇ Polyfills (a.k.a shims, and fallbacks) are libraries that emulate HTML5 features (e.g. Storage, Web Sockets, etc.) in browsers that don't natively support them
- Picturefill supports the HTML5 *picture* element that will allow web developers create responsive images depending on the device's screen, viewport size, etc.

28.16 Using Picturefill

- *Picturefull's* code is bundled as a single downloadable js file *picturefill.js*
- After you download the file from their website, use it in your code:

```
<script src="picturefill.js"></script>
```

- Picturefill adds support for both the *picture* element and also new responsive attributes to the *img* element, e.g.

```
<img srcset="images/large.png 1024w, images/medium.png 640w, images/small.png 320w" sizes="100vw" alt="Responsive images" />
```

- The **sizes** attribute indicates the ratio (as percentage) of the visual viewport width to layout viewport
 - ◇ it is expressed in **vw** units and is normally 100vw meaning that they are the same - a 1:1 or 100% ratio

28.17 Summary

- Creating responsive images is a multi-facet activity that should take into consideration image sizes and its adequate resolution
- There is a number of techniques to deal with these issues
- The HTML *picture* element address challenges related to responsive images in a standard uniform way

Chapter 29 - Bootstrap Overview

Objectives

In this module we will discuss:

- The history of Bootstrap
- Responsive web development (RWD)
- Mobile First
- Using Bootstrap
- Features

29.1 What Is Bootstrap

- Bootstrap [<http://getbootstrap.com/>] is a front-end framework for creating responsive web sites (more on this in a moment).
- In a nutshell, Bootstrap contains a set of stylesheets and jQuery-based plugins for implementing common and advanced UI elements (buttons, lists, pagination, navigation bars, etc.) on top of existing HTML elements in a web page.
 - ◇ **Note:** jQuery is only needed for select advanced features
- Used in numerous commercial sites.
- The <http://themes.getbootstrap.com/> site offers (not for free!) a variety of themes for using in SysAdmin, marketing, and other dashboards.

29.2 Keywords from package.json

- The Bootstrap's `package.json` file lists the following keywords that help summarize the key aspects of the framework:

```
"keywords": [  
  "css",  
  "sass",  
  "mobile-first",  
  "responsive",  
  "front-end",  
  "framework",
```

```
"web"  
]
```

29.3 Bootstrap History

- Originally, *Twitter Blueprint*
- Created as an internal project at Twitter
 - ◇ Mark Otto (@mdo)
 - ◇ Jacob Thornton (@fat)
- Open source
- Bootstrap is now released under the MIT license
 - ◇ It was Apache License 2.0 prior to version 3.1.0

29.4 Responsive Web Development

- *"Responsive Web design (RWD) is a Web design approach aimed at crafting sites to provide an optimal viewing experience—easy reading and navigation with a minimum of resizing, panning, and scrolling—across a wide range of devices (from mobile phones to desktop computer monitors"* [Wikipedia]
- Bootstrap supports RWD with the following features:
 - ◇ Responsive Grid and Flexible layout options
 - ◇ Responsive GUI components
 - ◇ JavaScript plug-ins
 - ◇ Mobile-Friendly CSS Styles

29.5 Responsive Grid Layout

- Bootstrap Grid
 - ◇ Core layout framework

- ◇ Responsive
- 12 Columns
- Four tiers of column styles
 - ◇ **xs** – phones
 - ◇ **sm** – tablets
 - ◇ **md** – desktops
 - ◇ **lg** – large desktops

29.6 Reusable GUI Components

- Alerts
- Buttons
- Navigation Components
- Labels
- Badges
- Jumbotron
- And more. see <https://getbootstrap.com/docs/4.0/components>

29.7 JavaScript

- Some dynamic Bootstrap components use JavaScript
- Built on jQuery
- Alternative implementations exist
 - ◇ Ng-Bootstrap built on Angular
 - ◇ UI Bootstrap built on AngularJS
 - ◇ Dojo Bootstrap – built on Dojo
- Components include

- ◇ Accordion, Modal, Dropdown, Carousel, Datepicker, Progressbar, ...

29.8 The Mobile First Philosophy

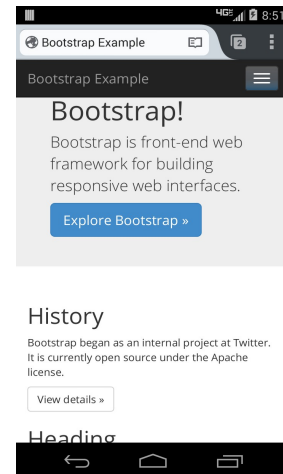
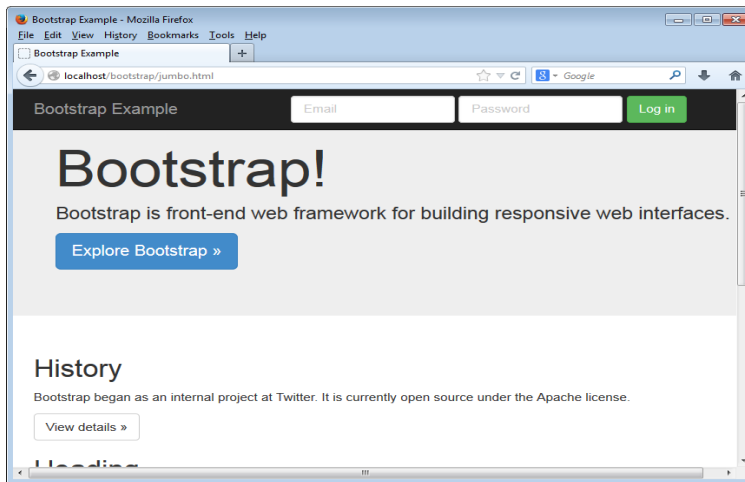
- Bootstrap supports a *Mobile First* development philosophy
 - ◇ Design with the mobile environment in mind and let the design grow to larger device displays.
 - ◇ Same design adapts to multiple form factors (responsive).
 - ◇ All styles are *mobile styles*.
 - ◇ Responsiveness is enabled by default.
- Mobile devices and tablets have overtaken desktop PCs with regard to internet usage.

29.9 Why RWD Matters

- Responsive Web Design improves the user experience.
- Modern users access the Web via a number of different devices:
 - ◇ Desktop computers
 - ◇ Laptop computers
 - ◇ Tablets
 - ◇ E-readers
 - ◇ Mobile phones
 - ◇ and more

29.10 Responsive Page Views

- A responsive page viewed in a large desktop browser and mobile browser.



29.11 SASS

- Sass is an open source scripting language that outputs CSS stylesheets when compiled.
- Features
 - ◇ Variables — for reuse within a stylesheet
 - ◇ Mixins — objects that contain a combination of properties from other objects
 - ◇ Nesting — simplified syntax for style inheritance and selectors
 - ◇ Functions — defining relationships between properties
- Bootstrap is available in Sass as well as CSS format.
- The issue with CSS is that it is static. All values are explicitly defined and while they are reusable through cascading, making wholesale changes to a stylesheet is difficult.

29.12 Getting Bootstrap

- There are several ways to get Bootstrap: [<http://getbootstrap.com/getting-started/introduction>]:
 1. A ZIP file containing compiled CSS and JavaScript files in both minified and uncompressed (assembled from various source files) forms, as well as fonts

2. Source files containing the above distribution files plus Sass stylesheets (templates) for CSS compilation, docs, examples, dependencies, metadata files for various systems, etc.
 3. Link a CDN version of bootstrap into your application.
 4. Use a package manager to pull bootstrap files into your project.
 - Various package managers are supported including npm, Ruby Gems, Composer, NuGet, etc...
- **Note:** Some Bootstrap components (i.e. advanced buttons, drop-downs, carousel functionality, etc.) require jQuery.

29.13 Bootstrap Content Delivery Network

- Content Delivery Networks (CDNs) are standard locations on the Internet where project files can be hosted and accessed in production.
- Bootstrap recommends MaxCDN URLs for
 - ◇ The main Bootstrap CSS file:

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
```

- ◇ The Bootstrap JavaScript file:

```
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpilMquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
```

- Other CDNs can be used for the supporting files:
 - ◇ jQuery
 - ◇ Popper
- See the following page for more details:

<http://getbootstrap.com/docs/4.0/getting-started/download/>

29.14 Other Setup Options

- In addition to the above ways (traditional ZIP download and CDNs) of getting Bootstrap and depending on your package manager preferences, you can get and manage Bootstrap with
 - ◇ npm
 - ◇ RubyGems
 - ◇ Composer
 - ◇ NuGet
- When using a package manager you'll need a Sass compiler and Autoprefixer to match official compiled versions.

29.15 The Bootstrap Core Files

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-grid.css
│   ├── bootstrap-grid.css.map
│   ├── bootstrap-grid.min.css
│   ├── bootstrap-grid.min.css.map
│   ├── bootstrap-reboot.css
│   ├── bootstrap-reboot.css.map
│   ├── bootstrap-reboot.min.css
│   └── bootstrap-reboot.min.css.map
└── js/
    ├── bootstrap.bundle.js
    ├── bootstrap.bundle.min.js
    ├── bootstrap.js
    └── bootstrap.min.js
```

Notes:

The above diagram shows what is included in the downloaded bundle. You only need to use a subset of these files in any given project. For more information about the above files see:

<http://getbootstrap.com/docs/4.0/getting-started/contents/#comparison-of-css-files>

29.16 To Min or Not to Min

- The *bootstrap.[css | js]* files are compiled and uncompressed and are usually used in development.
- The files with *min* in their name (e.g. *bootstrap-min.css*, *bootstrap-theme.min.css*, and *bootstrap.min.js*) are compiled and minified (compressed).
- The minified versions are made smaller by removing white spaces, name shortening, and other techniques.
- The minified versions should be used in production for faster network access.
- **Note 1:** CSS source maps (*bootstrap.*.map*) are used for viewing minified versions of CSS files in developer tools embedded in most modern browsers.
- **Note 2:** If you need a map file for debugging Bootstrap's JavaScript minified (production) version, you will need to get the jQuery map file [<http://jquery.com/download/>]

29.17 Summary

- In this module, we examined Bootstrap, a framework for responsive web development. We looked at:
 - ◇ The history of Bootstrap
 - ◇ Bootstrap Features
 - ◇ Responsive and Mobile First web development
 - ◇ Bootstrap core files

Chapter 30 - Using Bootstrap

Objectives

In this module we will discuss:

- Including Bootstrap files
- Layouts
- Navigation
- Application Icons

30.1 Including Bootstrap CSS Files

- Since Bootstrap is primarily composed of CSS the minimum requirement for using Bootstrap is to link the CSS files
- Bootstrap CSS files are actually compiled SASS files
 - ◇ SASS allows you to quickly customize the Bootstrap CSS files
- Required file
 - ◇ bootstrap.css
- Bootstrap CSS (and JavaScript files are also available in a minified form)
 - ◇ bootstrap.min.css
 - ◇ Optimized for web transfer (whitespace, comments, etc. removed)
- Bootstrap CSS is included *normally* in the head section of a document

30.2 Including Bootstrap JavaScript Files

- The Bootstrap JavaScript is not required for other elements of Bootstrap
- Some Bootstrap components are built on top of jQuery
- These components require
 - ◇ bootstrap.js (or bootstrap.min.js)
 - ◇ jquery-x.y.z.js (jquery-x.y.z.min.js)
- For performance reasons, the Bootstrap JavaScript can be linked at the

end of the body section

- ◇ However, if you are writing your own custom JavaScript that uses jQuery, the inclusion of the Bootstrap and jQuery libraries must come before your custom code

Including Bootstrap JavaScript Files

The actual version of jQuery required will vary based on the version of the bootstrap library used. One thing to keep in mind is the bootstrap files do not have a version number embedded in the file name. The bootstrap CDN uses the URL or path to encode version number.

e.g. <http://netdna.bootstrapcdn.com/bootstrap/3.0.0/js/bootstrap.min.js>

If you are writing no custom JavaScript that uses jQuery you could include the link to the Bootstrap and jQuery libraries at the end of the body section for better performance. If you are writing your own custom JavaScript that uses jQuery (since it is also being made available) your custom jQuery code must come after the inclusion of the jQuery library. This means you might instead put the link to the jQuery and Bootstrap libraries in the head section.

30.3 Viewport Meta Tags

- The "viewport" meta tag is used to configure the way a mobile browser handles zooming
- Causes CSS media queries to return physical device size not zoomed size
- Properties
 - ◇ width – usually "device-width" physical width
 - ◇ height – usually "device-height" physical height
 - ◇ initial-scale – usually "1.0" not zoomed
 - ◇ minimum-scale – min allowable zoom "1.0" means no zoom
 - ◇ maximum-scale – max allowable zoom "1.0" means no zoom
 - ◇ user-scalable – "yes" user can zoom / "no" user cannot zoom

30.4 Example

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0">
    <link href="/css/bootstrap.min.css" rel="stylesheet" media="screen">
  </head>
  <body>

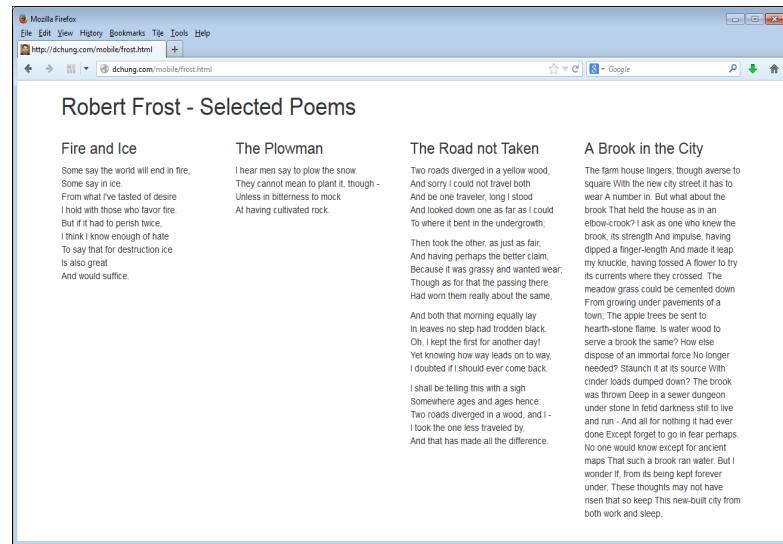
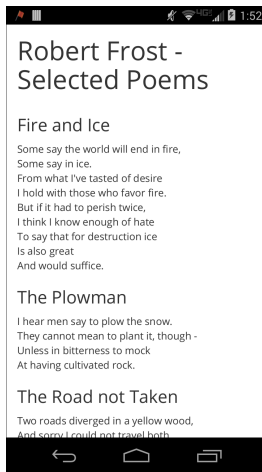
    <!-- add content here -->

    <script src="/js/jquery-1.9.1.js"></script>
    <script src="/js/bootstrap.min.js"></script>
  </body>
</html>
```

30.5 Layouts

- One of the very powerful features of Bootstrap is its responsive grid layout
- Scales up to 12 columns
- Made up of a series of <div> tags
- Pre-defined style classes to support grids
 - ◇ .container
 - ◇ .row
 - ◇ .col-xs-1,col-xs-2 ... col-xs-12,
 - xs refers to the device size (xs,sm,md,lg)
 - numeric value is the number of grid cells the column spans

30.6 Grid



Grid

All poems shown are in the public domain

30.7 Grid Source

```
<div class="container">
  <div class="row">
    <div class="col-sm-12">
      <h1>Robert Frost - Selected Poems</h1>
    </div>
  </div>
  <div class="row">
    <div class="col-sm-3">
      <h3>Fire and Ice</h3>
      Some say the world will end in fire,<br>
    </div>
    <div class="col-sm-3">
      <h3>The Plowman</h3>
      I hear men say to plow the snow.<br>
    </div>
    <div class="col-sm-3">
      <h3>The Road not Taken</h3>
      Two roads diverged in a yellow wood,<br>
    </div>
    <div class="col-sm-3">
      <h3>A Brook in the City</h3>
      The farm house lingers, though averse to square
    </div>
  </div>
</div>
```

30.8 Grid Explained

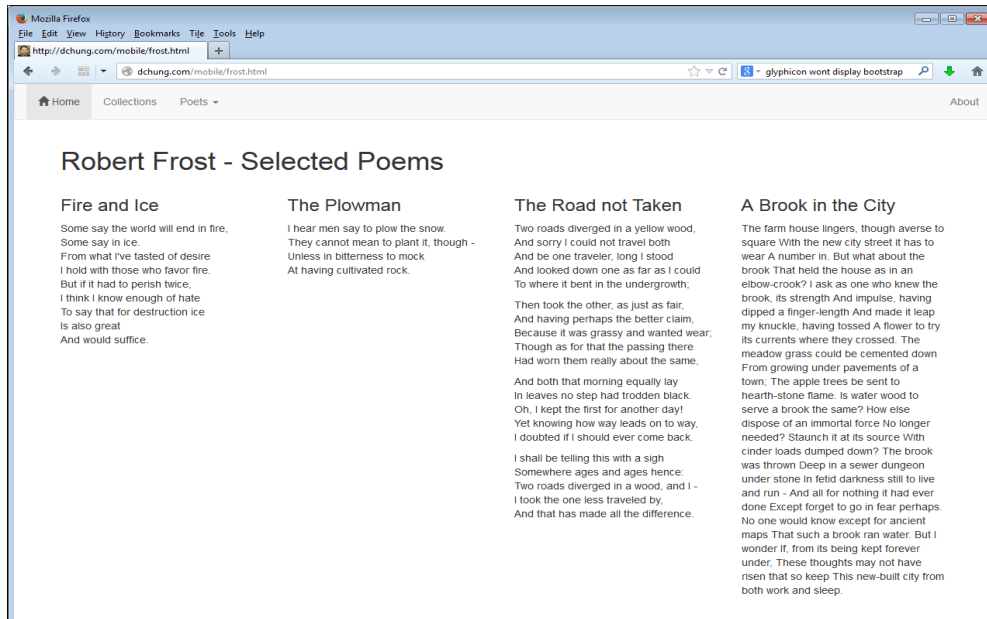
- The grid is contained within a <div> tag marked up with the *.container* CSS class
- There are two rows, one for the page title and one for the poems
- The page title is in a column with class "col-sm-12"
 - ◇ It is for small screens and spans 12 columns
- Each poem is in a column with class "col-sm-3"
 - ◇ Spans 3 (of 12) columns
- In the desktop browser you see the title and each of the four poems in a column

- As you shrink the display horizontally, things compress until eventually the columns stack up vertically
- In the mobile device the columns start out vertically stacked

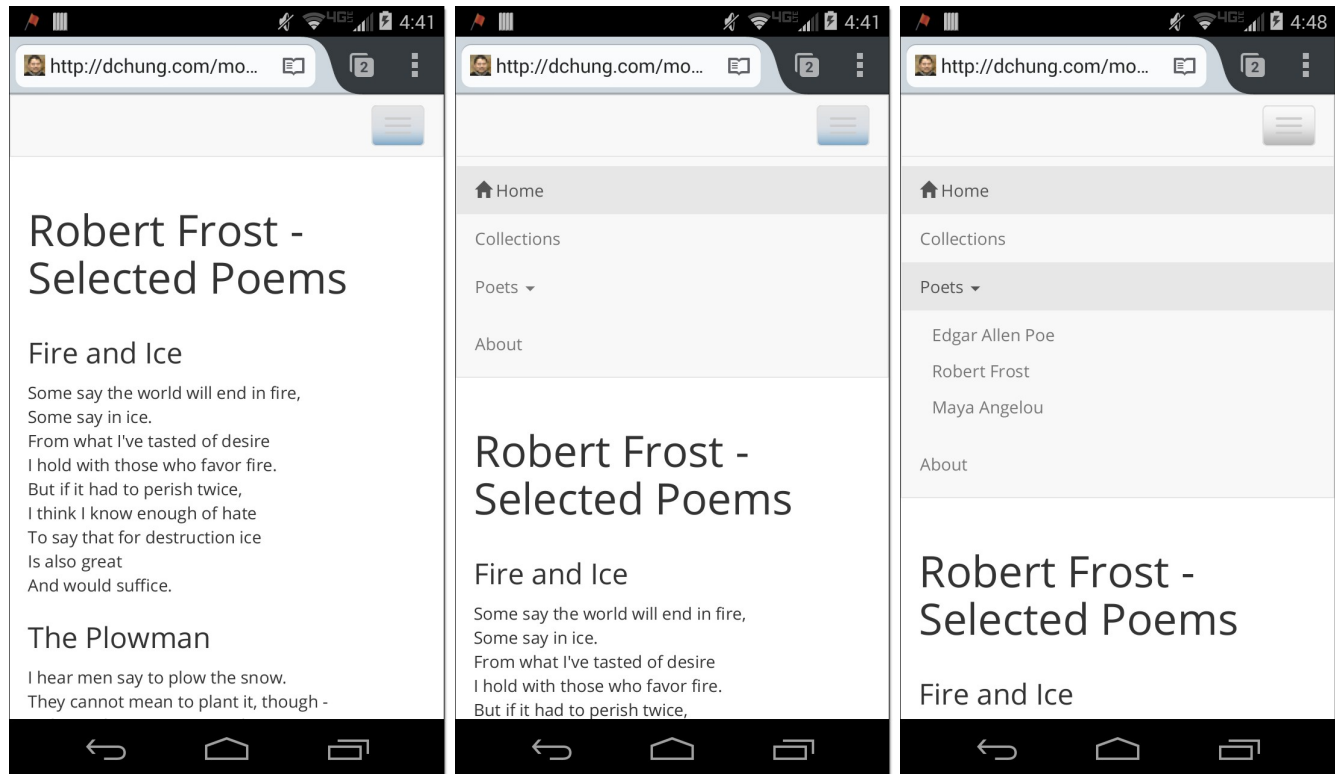
30.9 Navigation

- Navigation is an important part of every application
- Bootstrap provides rich responsive navigational functionality
 - ◇ navbar
 - ◇ embedded forms
 - ◇ dropdowns
 - ◇ fixed elements
- Collapsed mode

30.10 Navigation (Desktop)



30.11 Navigation (Mobile)



30.12 Navigation Source

```
<nav class="navbar navbar-default" role="navigation">
  <div class="navbar-header">
    <button type="button" class="navbar-toggle" data-toggle="collapse"
      data-target="#bs-example-navbar-collapse-1">
      <span class="sr-only">Toggle navigation</span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
    </button>
  </div>
  <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
    <ul class="nav navbar-nav">
      <li class="active"><a href="#">
        <span class="glyphicon glyphicon-home"></span> Home</a></li>
      <li><a href="#">Collections</a></li>
      <li class="dropdown">
        <a href="#" class="dropdown-toggle" data-toggle="dropdown">Poets <b class="caret"></b></a>
        <ul class="dropdown-menu">
          <li><a href="#">Edgar Allen Poe</a></li>
          <li><a href="#">Robert Frost</a></li>
          <li><a href="#">Maya Angelou</a></li>
        </ul>
      </li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
      <li><a href="#">About</a></li>
    </ul>
  </div>
</nav>
```

30.13 Navigation Explained

- This example created a navigation bar with
 - ◇ Links
 - ◇ A dropdown menu with its own links
 - ◇ An About link pinned to the right side of the display
- The navigation bar was also collapsible
 - ◇ In collapsed mode it showed up as three horizontal bars

30.14 Navigation Elements and Styles

- The entire navbar is contained within the new HTML5 <nav> element
- <div> for the navbar header
 - ◇ Defines collapsed look and feel
- <div> for the collapsible navbar

- ◇ Defines 'normal' look and feel
- defines main navbar elements and .navbar-right elements
 - ◇ The dropdown element is a jQuery plugin

30.15 Application Icons

- Prior versions of Bootstrap included the Glyphicons icon set
- As of version 4.0 the Bootstrap team suggests that developers use one of the following 3rd party icon sets:
 - ◇ Iconic - <https://useiconic.com/open/>
 - ◇ Octicons - <https://octicons.github.com/>
 - ◇ Font Awesome - <https://fontawesome.com/>
 - ◇ Glyphicons - <https://glyphicons.com>

- Additional choices for icon sets can be found here:

<https://getbootstrap.com/docs/4.0/extend/icons/>

30.16 Summary

- In this module, we looked at how to use Bootstrap, including:
 - ◇ Bootstrap files
 - ◇ Layouts
 - ◇ Navigation
 - ◇ Application Icons

Chapter 31 - Bootstrap Miscellaneous Topics

Objectives

In this module we will discuss:

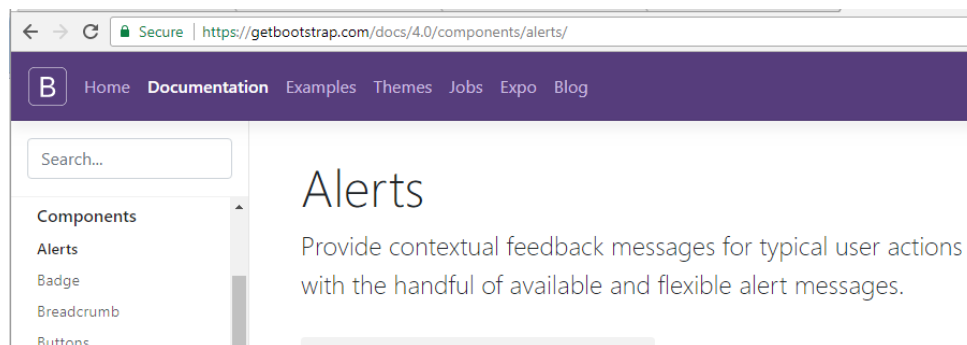
- Bootstrap components
- Integration with jQuery
- Customizing Bootstrap

31.1 Bootstrap Components

- Bootstrap is shipped with a number of useful re-useable components grouped into functional categories, e.g.
 - ◇ Alerts
 - ◇ Labels
 - ◇ Buttons
 - ◇ etc.

31.2 Bootstrap Components Web Page

- The Bootstrap Components Web page (<https://getbootstrap.com/docs/4.0/components/>) provides a list of available components along with their usage examples:



31.3 Integrating Bootstrap Components with jQuery

- Bootstrap components are mostly static elements that lack dynamic characteristics
- This is by design as the main focus of Bootstrap is on ensuring web page responsiveness to various screen sizes and the target browser's capabilities
- Bootstrap components' run-time dynamics (such as element hiding/showing, CSS class changing, progress bar updates, etc.) can be easily achieved by integrating components on your web pages with jQuery (covered in previous modules)
 - ◇ **Note:** You will remember that jQuery is also required by Bootstrap's JavaScript plugins (packaged in the *bootstrap[.min].js* file) as evidenced by the following code snippet from that file:

```
if (typeof jQuery === "undefined") {  
    throw new Error("Bootstrap requires jQuery") }
```

31.4 Identifying the Required Version of jQuery

- To identify which version of jQuery is required for your Bootstrap package, check the Bootstrap package's *package.json*
 - ◇ *use npm to install bootstrap:*

```
npm install bootstrap
```
 - ◇ Then check the *package.json*

```
\node_modules\bootstrap\package.json
```
- The *package.json* file's content (abridged for space) is shown below indicating that jQuery version 1.9.1 is required for bootstrap v4.0.0.

```
{  
  "name": "bootstrap",  
  "_id": "bootstrap@4.0.0",  
  . . .  
  "peerDependencies": {  
    "jquery": "1.9.1 - 3"  
  }  
}
```

```
}
```

31.5 Customizing Bootstrap

- Bootstrap can be customized via Sass
- For example you can choose to include all of bootstrap by adding the following to the *custom.scss* file:

```
// custom.scss
@import "node_modules/bootstrap/scss/bootstrap";
```

- Or you can include just the parts you need like this:

```
// custom.scss
// Required
@import "node_modules/bootstrap/scss/functions";
@import "node_modules/bootstrap/scss/variables";
@import "node_modules/bootstrap/scss/mixins";

// Optional
@import "node_modules/bootstrap/scss/reboot";
@import "node_modules/bootstrap/scss/type";
@import "node_modules/bootstrap/scss/images";
@import "node_modules/bootstrap/scss/code";
@import "node_modules/bootstrap/scss/grid";
```

31.6 Customizing Sass variables

- You can further customize Bootstrap by overriding any of the Sass variables.
- Sass variable overrides are added before the "Required" section of your *custom.scss* file:

```
// Your variable overrides
$body-bg: #000;
$body-color: #111;
```

```
// Bootstrap and its default variables
@import "node_modules/bootstrap/scss/bootstrap";
```

- Sass default variables can be found in the following file:

```
\node_modules\bootstrap\scss\_variables.scss
```

- Do not modify this file directly! Instead, use the above method to override the variable values.

31.7 More Customization

- More information regarding suggested techniques for customizing Bootstrap appear on the following page:

```
https://getbootstrap.com/docs/4.0/getting-started/theming/
```

- Some of the topics covered include:
 - ◇ Sass Customization
 - ◇ Sass Global Options
 - ◇ Color Settings
 - ◇ CSS Variables

31.8 Customizing Bootstrap Components

- Bootstrap components can be further customized through light customizations and overhauls
- Customizations involve changing the visual aspects of Bootstrap components like colors and fonts
- Customization uses a simple principle: you use Bootstrap's default CSS file as a base and override some properties with classes from your custom stylesheet file

- ◇ An overhaul is a more complicated customization technique than light customizations and it is not reviewed here

31.9 Light Customizations Steps

- To customize the visual aspect of an existing "stock" or "base" class, e.g. *.btn*, you create a custom CSS class that overrides some or all of the properties of the base class
- You save your custom class in a separate stylesheet file
- Your custom class must replicate the properties of the base class you want to modify across all the desired control states, e.g. active, mouse hover, etc.
 - ◇ **Note:** You can simply copy & paste the base class' properties from the Bootstrap CSS file and override them in your custom CSS file
- Use the class chaining ***class="base-class your-class"*** technique in your HTML pages to modify the base class' behavior

31.10 Summary

- In this module, we looked at how to use Bootstrap, including:
 - ◇ Bootstrap components
 - ◇ Integration with jQuery
 - ◇ Customizing Bootstrap