HTML5 and JavaScript Projects

Build on your Basic Knowledge of HTML5 and JavaScript to Create Substantial HTML5 Applications

—

Second Edition

—

Jeanine Meyer

Apress®

HTML5 and JavaScript Projects

Build on your Basic Knowledge of HTML5 and JavaScript to Create Substantial HTML5 Applications

Second Edition

Jeanine Meyer

Apress®

HTML5 and JavaScript Projects

Jeanine Meyer
New York, USA

ISBN-13 (pbk): 978-1-4842-3863-9 ISBN-13 (electronic): 978-1-4842-3864-6
https://doi.org/10.1007/978-1-4842-3864-6

Library of Congress Control Number: 2018954635

Copyright © 2018 by Jeanine Meyer

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Louise Corrigan
Development Editor: James Markham
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Cover image designed by Mattia Serrani on Unsplash

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484238639. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To my family, including my parents, who still take care of me

Table of Contents

About the Author

Jeanine Meyer is a full professor at Purchase College/State University of New York. She teaches courses to students majoring in mathematics/computer science and also enjoys the frequent presence of others in her classes, including new media, music, dance, economics, and chemistry majors. She developed and teaches courses satisfying the mathematics general education requirement, including one on math in the news and one on origami. The website for her academic activities is `http://faculty.purchase.edu/jeanine.meyer`. Before coming to academia, she was a research staff member and manager at IBM Research, focusing on robotics and manufacturing research, and she later worked on the corporate manufacturing staff and as a research consultant at IBM for educational grant programs.

She has enjoyed working with Apress, including updating the HTML5 books. She continues with the practice of building programming examples using media featuring her family and her activities and hopes that inspires readers to create work on topics that are important to them. Her hobbies and interests include studying Spanish and piano, playing computer games, doing origami, and volunteering for progressive candidates and causes. She enjoys her daughter Aviva's cooking while doing some baking herself and looks forward to travel this year.

About the Technical Reviewer

Takashi Mukoda is an international student at Purchase College/State University of New York. He is currently taking a semester off and back home in Japan. At Purchase College, he majors in Mathematics/Computer Science and New Media and has worked as a teaching assistant for computing and mathematics courses.

Takashi likes playing the keyboard and going on hikes in the mountains to take pictures. His interest in programming and art motivates him to create multimedia art pieces. Some of them are built with Processing and interact with human motion and sounds.

(See his website at `http://www.takashimukoda.com`.)

Acknowledgments

Much appreciation to my students and colleagues at Purchase College/State University of New York. In particular, for Chapter 5, which covers the map portal quiz, I want to thank Jennifer Douglas, Jeremy Martinez, and Nik Dedvukaj, for the maze video clip produced in my Robotics class in 2008, which I retained for the new edition. I want also to thank Takashi Mukoda for his photograph of the Great Torii, in addition to everything else he has contributed to this and other projects. I re-used an audio recording of my mother playing piano. Thanks to all my family members (Aviva Meyer, Daniel Meyer, Annika Meyer, Anne Kellerman, Palmer Agnew, Debbie Torres, and Joshua Torres) for being the subjects of photographs and videos and for making the photos, video, and audio. Thanks to Daniel Davis for his HTML5 logo and his technical assistance with the first edition.

Thanks to the crew at Apress, including Nancy Chen, James Markham, and the technical reviewer Takashi Mukoda, as well as others I do not know by name.

Introduction

This book continues my exploration of HTML5. My approach in developing the projects was to combine features such as canvas and video, attempt more intricate drawing by using mathematics, and use standard programming techniques such as object-oriented programming and separation of content and logic. I was also interested in building applications combining HTML5 and JavaScript with other technologies, including Google Maps.

Each chapter in the book is focused on an application or set of related applications. This is because my experience as a teacher and a learner has shown that concepts and mechanics are best understood in the context of actual use. The applications start off with drawing the HTML5 official logo. As you will find out in Chapter 1, the way I developed this application prompted use of coordinate transformations.

The project in Chapter 2, involving a family collage, was inspired by my growing family and the desire to teach about object-oriented programming. It is a good application for you to use as a foundation to create your own, with your own photos and objects of your own invention. Chapter 3, which shows how to create a bouncing video, was built on other two-dimensional applications I have created, and features two different ways to combine canvas and video.

Chapters 4 and 5 demonstrate use of the Google Maps API (Application Programming Interface), a powerful facility that allows you to incorporate access to Google Maps as part of your own projects. Chapter 4 presents a user interface combining map and canvas and includes a custom-designed cursor and the use of alpha (transparency) in drawing paths. The map quiz in Chapter 5 demonstrates the use of mapping as a portal to media. The application shows you how to separate content and logic so you can scale up to various applications (e.g., a tour of a region or a geography quiz with many locations).

Chapter 6 features a game called Add to 15, which turned out to be an excellent example of arrays. It also demonstrated the necessity to prepare for bad behavior on the part of players.

In Chapter 7, I use the task of producing directions for origami to show how to combine line drawings, often using mathematical expressions, and video and photographs. You can use this as a model for your own set of directions for a task in which drawings, video, or images would be most appropriate. Or you can let the reading refresh your memory for topics in algebra and geometry. Chapter 8 features a jigsaw puzzle that is transformed into a video when it's completed. Chapter 9 is an educational game with questions on the states of the USA, and it includes the challenge of a jigsaw puzzle. The jigsaw puzzle includes the feature of saving the puzzle-in-progress using `localStorage`.

For Chapter 10, I decided to address challenges of responsive design and accessibility as being more appropriate for an HTML and JavaScript book than what I had before. My examples demonstrate ways to incorporate touch in addition to mouse actions, to respond to different screen dimensions, and to specify tab order to ease the use of screen readers.

Who Is This Book For?

I do believe my explanations are complete, but I am not claiming, as I did for my previous book, *The Essential Guide to HTML5*, that this book is for the total beginner. This book is for the developer who has some knowledge of programming and who wants to build (more) substantial applications by combining features of JavaScript and going beyond the basics. It also can serve as an idea book for someone working with programmers to get an understanding of what is possible.

How Is This Book Structured?

This book consists of 10 chapters, each organized around an application or type of application. You can skip around, though there are cross-references between chapters, indicated in the text. Each chapter starts with an introduction to the application, with screenshots of the applications in use. The chapters continue with a discussion of the critical requirements in which concepts are introduced before diving into the technical details. The next sections describe how the requirements are satisfied, with specific constructs in HTML5, JavaScript, and CSS. I then show the application coding line by line with comments. You can decide how to read these tables. You may decide to use them as a reference when writing your own programs. Each chapter ends with instructions and tips for testing and uploading the application to a server, and a summary of what you learned.

The code is included as downloads available from the publisher. Go to `https://github.com/Apress/html-js-projs`. In addition, the figures are available as full-color TIFF files. Of course, you will want to use your own media for the projects. My media (video, audio, and images) is included with the code and this includes images for the 50 states for the states game in Chapter 9. You can use the project as a model for a different part of the world or a puzzle based on an image or diagram. There are extras: a program for an origami frog included with the code in Chapter 7 and a version of the jigsaw turning into a video from Chapter 8 adapted for use on devices requiring touch is included with the source code for Chapter 10.

Let's get started.

CHAPTER 1

Building the HTML5 Logo: Drawing on Canvas with Scaling and Semantic Tags

In this chapter, you will learn the following:

- Drawing paths on a canvas
- Placing text on a canvas
- Coordinate transformations
- Fonts for text drawn on canvas and fonts for text in other elements
- Semantic tags
- The range input element

Introduction

The project for this chapter is a presentation of the official HTML5 logo, with accompanying text. The shield and letters of the logo are drawn on a canvas element and the accompanying text demonstrates the use of semantic tags. The viewer can change the size of the logo using a slider input device. It is an appropriate start to this book, a collection of projects making use of HTML5, JavaScript, and other technologies, because of the subject matter and because it serves as a good review of basic

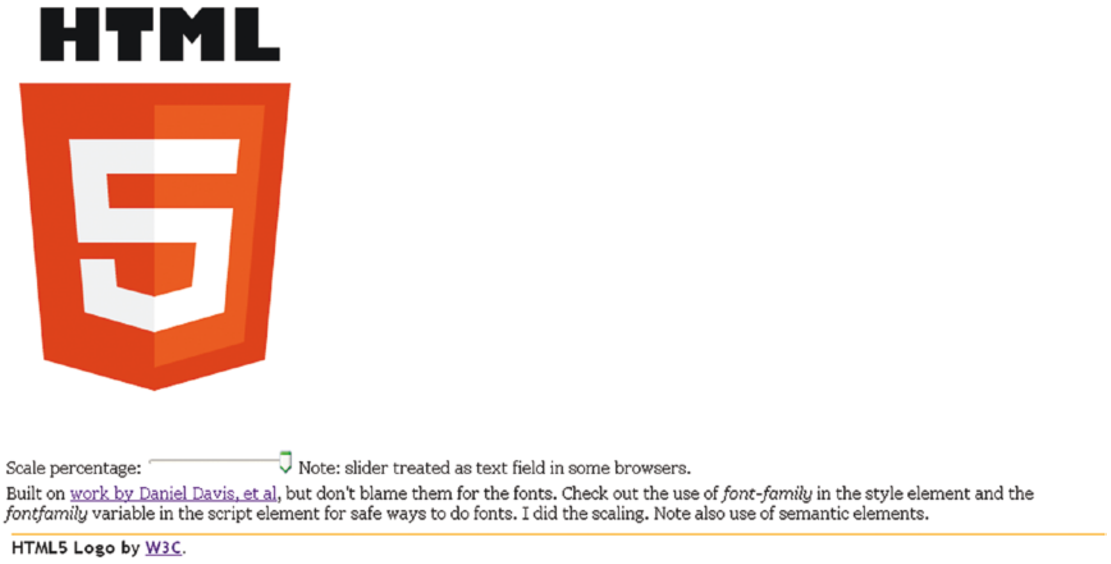
© Jeanine Meyer 2018
J. Meyer, *HTML5 and JavaScript Projects*, https://doi.org/10.1007/978-1-4842-3864-6_1

event-driven programming and other important features in HTML5. The way I developed the project, building on the work of others, is typical of how most of us work. In particular, the circumstances provide motivation for the use of coordinate transformations.

The approach of this book is to explain HTML5, Cascading Style Sheets (CSS), and JavaScript chapters in the context of specific examples. The projects represent a variety of applications and, hopefully, you will find something in each one that you will learn and adapt for your own purposes.

Note If you need an introduction to programming using HTML5 and JavaScript, you can consult my book, *The Essential Guide to HTML5* or other books published by Apress or others. There also is considerable material available online, for example, at W3Schools.

Figure 1-1 shows the opening screen for the logo project on the Chrome browser. It is important to realize that browsers can be different. Look ahead to how this appeared using Firefox when I first wrote this example.



Scale percentage: Note: slider treated as text field in some browsers.

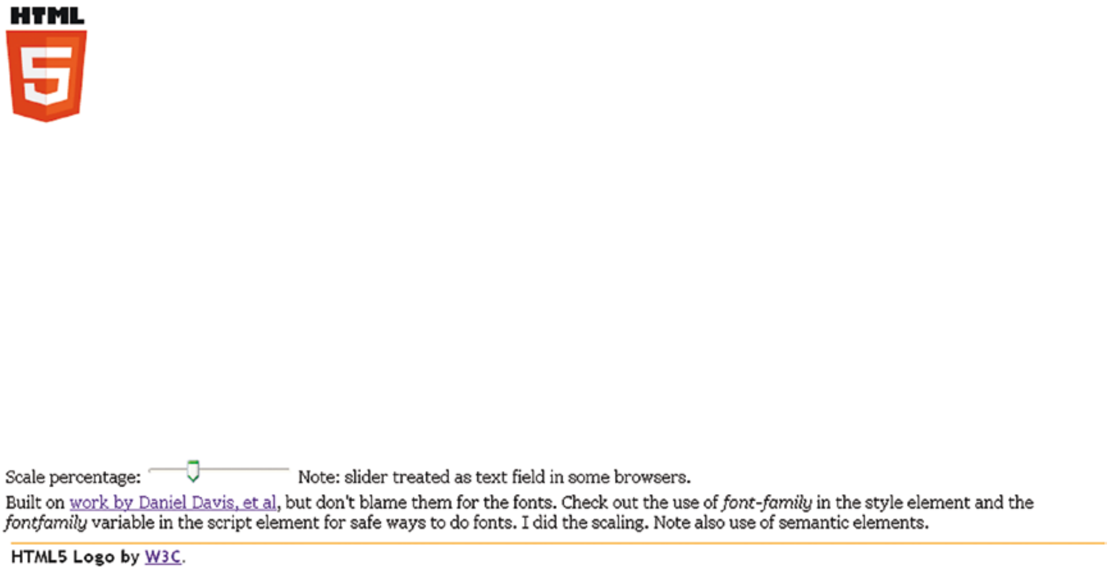
Built on work by Daniel Davis, et al, but don't blame them for the fonts. Check out the use of *font-family* in the style element and the *fontfamily* variable in the script element for safe ways to do fonts. I did the scaling. Note also use of semantic elements.

HTML5 Logo by W3C.

Figure 1-1. *Opening screen for HTML5 logo*

Notice the slider feature, the accompanying text, which contains what appears to be a hyperlink, and the text in a footer below a yellow line. The footer also includes a hyperlink. As I will explain later, the function and the formatting of the footer and any other semantic element is totally up to me, but providing a reference to the owners of the logo, the World Wide Web Consortium would be deemed an appropriate use.

The viewer can use the slider to change the size of the logo. Figure 1-2 shows the application after the slider has been adjusted to show the logo reduced to about a third in width and in height.



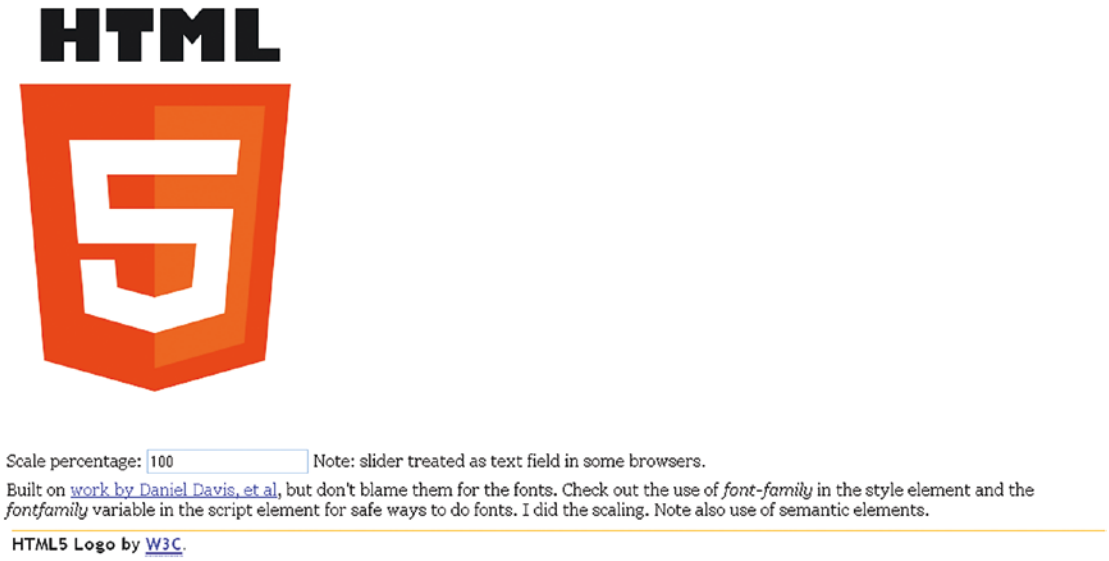
Scale percentage: Note: slider treated as text field in some browsers.

Built on work by Daniel Davis, et al, but don't blame them for the fonts. Check out the use of *font-family* in the style element and the *fontfamily* variable in the script element for safe ways to do fonts. I did the scaling. Note also use of semantic elements.

HTML5 Logo by W3C.

Figure 1-2. *Logo scaled down*

The implementation of HTML5 is complete, or pretty close, in all browsers. However, I want to show you something from the past to illustrate the term *graceful degradation*. Figure 1-3 shows the opening screen in the older Firefox. The range input is treated as text. Notice the initial value is displayed as 100.



Scale percentage: 100 Note: slider treated as text field in some browsers.

Built on work by Daniel Davis, et al, but don't blame them for the fonts. Check out the use of *font-family* in the style element and the *fontfamily* variable in the script element for safe ways to do fonts. I did the scaling. Note also use of semantic elements.

HTML5 Logo by W3C.

Figure 1-3. *Application using Firefox*

As will be the practice in each chapter, I now explain the critical requirements of the application, more or less independent of the fact that the implementation will be in HTML5, and then describe the features of HTML5, JavaScript, and other technologies as needed that will be used in the implementation. The "Building" section includes a table with comments for each line of code and guidance for building similar applications. The "Testing" section provides details for uploading and testing. This section is more critical in some projects than others. Lastly, there is a "Summary" section that reviews the programming concepts covered and previews what is next in the book.

Project History and Critical Requirements

The critical requirements for this project are somewhat artificial and not easily stated as something separate from HTML. For example, I wanted to draw the logo as opposed to copying an image from the Web. My design objectives always include wanting to practice programming and prepare examples for my students. The shape of the shield part of the logo seemed amenable to drawing on canvas and the HTML letters could be done using the draw text feature. In addition, there are practical advantages to drawing images instead of using image files. Separate files need to be managed, stored, and downloaded. The image shown in Figure 1-4 is 90KB. The file holding the code for the program is only 4KB. Drawing a logo or other graphic means that the scale and other attributes can be changed dynamically using code.

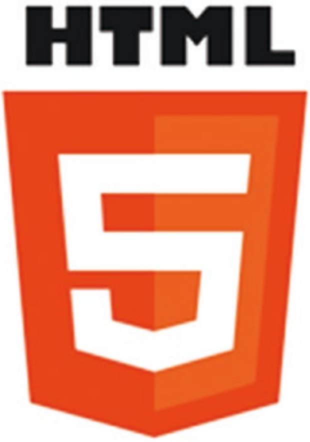


Figure 1-4. *Image of a logo*

I looked online and found an example of just the shield done by Daniel Davis, `@ourmaninjapan`. This was great because it meant that I did not have to measure a copy of the logo image to get the coordinates. This begs the question of how he determined the coordinates. I don't know the answer, even though we had a pleasant exchange of emails. One possibility is to download the image and use the grid feature of image processing programs such as Adobe Photoshop or Corel Paint Shop Pro. Another possibility is to use (old-fashioned) transparent graph paper.

However, there was a problem with building on Daniel Davis's work. His application did not include the HTML letters. The solution to this was to position the letters on the screen and then move down, so to speak, to position the drawing of the shield using the coordinates provided in Daniel's example. The technical term for "moving down the screen" is performing a coordinate transformation. So the ability to perform coordinate transformations became a critical requirement for this project.

I chose to write something about the logo and, in particular, give credit and references in the form of hyperlinks. I made the decision to reference the official source of the logo as brief text at the bottom of the document below a line. The reference to Daniel Davis was part of the writing in the body. We exchanged notes on font choices and I will discuss that more in the next section.

In order to give the viewer something to do with the logo, I decided to present a means of changing the size. A good device for this is a slider with the minimum and maximum values and steps all specified. So the critical requirements for this application include drawing shapes and letters in a specific font, coordinate transformations, formatting a document with a main section and a footer section, and including hyperlinks.

HTML5, CSS, and JavaScript features

I assume that you, the reader, have some experience with HTML and HTML5 documents. One of the most important new features in HTML5 is the canvas element for drawing. I describe briefly the drawing of filled-in paths of the appropriate color and filled-in text. Next, I describe coordinate transformations, used in this project for the two parts of the logo itself and for scaling, changing the size, of the whole logo. Lastly, I describe the range input element. This produces the slider.

Drawing Paths on Canvas

Canvas is a type of element introduced in HTML5. All canvas elements have a property (aka an *attribute*) called the 2D context. The context has methods for drawing, which you will see in use. Typically, a variable is set to this property after the document is loaded:

```
ctx = document.getElementById('canvas').getContext('2d');
```

It is important to understand that canvas is a good name: code applies color to the pixels of the canvas, just like paint. Code written later can put a different color on the canvas. The old color does not show through. Even though our code causes rectangles and shapes and letters to appear, these distinct entities do not retain their identity as objects to be re-positioned.

The shield is produced by drawing six filled-in paths in succession with the accumulated results, as shown in Figure 1-5. You can refer to this picture when examining the code. Keep in mind that in the coordinates, the first number is the distance from the left edge of the canvas and the second number is the distance from the top edge of the canvas.

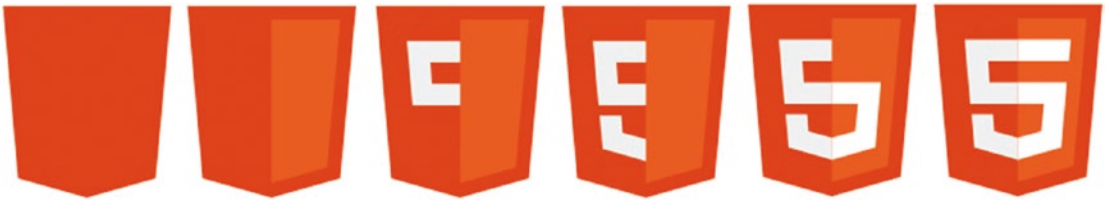


Figure 1-5. *Sequence of paths for drawing the logo*

By the way, I chose to show you the sequence with the accumulated results. If I displayed what is drawn, you would not see the white parts making up the left side of the five. You can see it because it is two white filled-in paths on top of the orange.

All drawing is done using methods and properties of the ctx variable holding the 2D context property of the canvas element. The color for any subsequent fill operation is set by assigning a color to the fillStyle property of the canvas context.

```
ctx.fillStyle = "#E34C26";
```

This particular color, given in the hexadecimal format—where the first two hexadecimal (base 16) digits represent red, the second two hexadecimal digits represent green, and the last two represent blue—is provided by the W3C website, along with the other colors, as the particular orange for the background of the shield. It may be counterintuitive, but in this system, white is specified by the value #FFFFFF. Think of this as all colors together make white. The absence of color is black and specified by #000000. The pearly gray used for the right side of the 5 in the logo has the value #EBEBEB. This is a high value, close to white. It is not necessary that you memorize any of these values, but it is useful to know black and white, and that a pure red is #FF0000, a pure green is #00FF00, and a pure blue is #0000FF. You can use the eyedropper/color picker tool in drawing programs such as Adobe Photoshop, Corel Paint Shop Pro, or the online tool http://pixlr.com/ to find out values of colors in images *or* you can use the official designation, when available, for official images.

All drawing is done using the two-dimensional coordinate systems. Shapes are produced using the path methods. These assume a current location, which you can think of as the position of a pen or paint brush over the canvas. The critical methods are moving to a location and setting up a line from the current location to the indicated location. The following set of statements draws the five-sided orange shape starting at the lower, left corner. The closePath method closes up the path by drawing a line back to the starting point.

```
ctx.fillStyle = "#E34C26";
ctx.beginPath();
ctx.moveTo(39, 250);
ctx.lineTo(17, 0);
ctx.lineTo(262, 0);
ctx.lineTo(239, 250);
ctx.lineTo(139, 278);
ctx.closePath();
ctx.fill();
```

If you haven't done any drawing on canvas, here is the whole HTML script needed to produce the five-sided shape. The onLoad attribute in the <body> tag causes the init function to be invoked when the document is loaded. The init function sets the ctx variable, sets the fillStyle property, and then draws the path.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML5 Logo</title>
<meta charset="UTF-8">
<script>
function init() {
 ctx = document.getElementById('canvas').getContext('2d');
 ctx.fillStyle = "#E34C26";
 ctx.beginPath();
 ctx.moveTo(39, 250);
 ctx.lineTo(17, 0);
 ctx.lineTo(262, 0);
 ctx.lineTo(239, 250);
 ctx.lineTo(139, 278);
 ctx.closePath();
 ctx.fill();
}
</script>
</head>
<body onLoad="init();">
<canvas id="canvas" width="600" height="400">
Your browser does not support the canvas element.
</canvas>
</body>
</html>
```

Do practice and experiment with drawing on the canvas if you haven't done so before, but I will go on. The other shapes are produced in a similar manner. By the way, if you see a line down the middle of the shield, this is an optical illusion.

Placing Text on Canvas and in the Body of a Document

Text is drawn on the canvas using methods and attributes of the context. The text can be filled in, using the `fillText` method or drawn as an outline using the `strokeText` method. The color is whatever the current `fillStyle` property or `strokeStyle` property holds. Another property of the context is the `font`. This property can contain the size of the text and one or more fonts. The purpose of including more than one font is to provide options to the browser if the first font is unavailable on the computer running the browser. For this project, I use

```
var fontfamily = "65px 'Gill Sans Ultra Bold', sans-serif";
```

and in the `init` function

```
ctx.font = fontfamily;
```

This directs the browser to use the Gill Sans Ultra Bold font if it is available and if not, use whatever the default sans serif font on the computer.

I could have put this all in one statement, but chose to make it a variable. You can decide if my choice of font was close enough to the official W3C logo.

Note There are at least two other approaches to take for this example. One possibility is *not* to use text but to draw the letters as filled-in paths. The other is to locate and acquire a font and place it on the server holding the HTML5 document and reference it directly using `@font-face`.

With the font and color set, the methods for drawing text require a string and a position: x and y coordinates. The statement in this project to draw the letters is

```
ctx.fillText("HTML", 31,60);
```

Formatting text in the rest of the HTML document, that is, outside a canvas element, requires the same attention to fonts. In this project, I choose to make use of the semantic elements new to HTML5 and follow the practice of putting formatting in the style element. The body of my HTML script contains two article elements and one footer elements. One article holds the input element with a comment and the other article holds the rest of the explanation. The footer element contains the reference to W3C. Formatting and using these are up to the developer/programmer. This includes making sure the footer is the last thing in the document. If I placed the footer before one or both articles, it would no longer be displayed at the foot, that is, the bottom of the document. The style directives for this project are the following:

```
footer {display:block; border-top: 1px solid orange; margin: 10px; ↩
 font-family: "Trebuchet MS", Arial, Helvetica, sans-serif; font-weight:
bold;}
article {display:block; font-family: Georgia, "Times New Roman", Times,
serif; margin: 5px;}
```

The styles each set up all instances of these elements to be displayed as blocks. This puts a line break before and after. The footer has a border on the top, which produces the line above the text. Both styles specify a list of four fonts each. So the browser first sees if Trebuchet MS is available, then checks for Arial, then for Helvetica and then, if still unsuccessful, uses the system default sans serif font for the footer element. Similarly, the browser checks for Georgia, then Times New roman, then Times and then, if unsuccessful, uses the standard serif font. This probably is overkill, but it is the secure way to operate. The footer text is displayed in bold and the articles each have a margin around them of 5 pixels.

Formatting, including fonts, is important. HTML5 provides many features for formatting and for separating formatting from structure and content. You do need to treat the text on the canvas differently than the text in the other elements.

Coordinate Transformations

I have given my motivation for using coordinate transformations, specifically to keep using a set of coordinates. To review, a coordinate system is the way to specify positions on the canvas. Positions are specified as distances from an origin point. For the two-dimensional canvas, two coordinates are necessary: the first coordinate governs the horizontal and is often called the x and the second coordinate governs the vertical and is called the y. A pesky fact is that when drawing to screens, the y axis is flipped so the vertical is measured from the top of the canvas. The horizontal is measured from the left. This means that the point (100,200) is further down the screen than the point (100,100).

In the logo project, I wrote code to display the letters HTML and then moved the origin to draw the rest of the logo. An analogy would be that I know the location of my house from the center of my town and so I can give directions to the center of town and then give directions to my house. The situation in which I draw the letters in the logo and "move down the screen" requires the translate transformation. The translation is done just in the vertical. The amount of the translation is stored in a variable I named `offsety`:

```
var offsety = 80;
...
ctx.fillText("HTML", 31, 60);
ctx.translate(0, offsety);
```

Since I decided to provide a way for the viewer to change the size of the logo, I used the scale transformation. Continuing the analogy of directions, this is equivalent to changing the units. You may give some directions in miles (or kilometers) and other directions in yards or feet or meters or, maybe, blocks. The scaling can be done separately for each dimension. In this application, there is a variable called `factorvalue` that is set by the function invoked when the input is changed. The statement

```
ctx.scale(factorvalue, factorvalue);
```

changes the units for both the horizontal and vertical direction.

HTML5 provides a way to save the current state of the coordinate system and restore what you have saved. This is important if you need your code to get back to a previous state. The saving and restoring is done using what is termed a *stack*: last in first out. Restoring the coordinate state is termed *popping the stack* and saving the coordinate state is *pushing* something onto the stack. My logo project does not use this in its full power, but it is something to remember to investigate if you are doing more complex applications. In the logo project, my code saves the original state when the document is first loaded. Then before drawing the logo, it restores what was saved and then saves it again so it is available the next time. This is overkill for this situation, but it is a good practice just in case I add something in the future. Do your own experiments! The code at the start of the function `dologo`, which draws the logo, starts as follows:

```
function dologo() {
var offsety = 80 ;
ctx.restore();
ctx.save();
ctx.clearRect(0,0,600,400);
ctx.scale(factorvalue,factorvalue);
ctx.fillText("HTML", 31,60);
ctx.translate(0,offsety);
```

```
// 5 sided orange background
ctx.fillStyle = "#E34C26";
ctx.beginPath();
ctx.moveTo(39, 250);
ctx.lineTo(17, 0);
ctx.lineTo(262, 0);
ctx.lineTo(239, 250);
ctx.lineTo(139, 278);
ctx.closePath();
ctx.fill();

// right hand, lighter orange part of the background
ctx.fillStyle = "#F06529";
ctx.beginPath();
ctx.moveTo(139, 257);
ctx.lineTo(220, 234);
ctx.lineTo(239, 20);
ctx.lineTo(139, 20);
ctx.closePath();
ctx.fill();
...
```

Note that the canvas is cleared (erased) of anything that was previously drawn.

Using the Range Input Element

The input device, which I call a *slider*, is the new HTML5 input `type range`, and is placed in the body of the HTML document. Mine is placed inside an article element. The attributes of this type and other input elements provide ways of specifying the initial value, the minimum and maximum values, the smallest increment adjustment, and the action to take if the viewer changes the slider. The code is

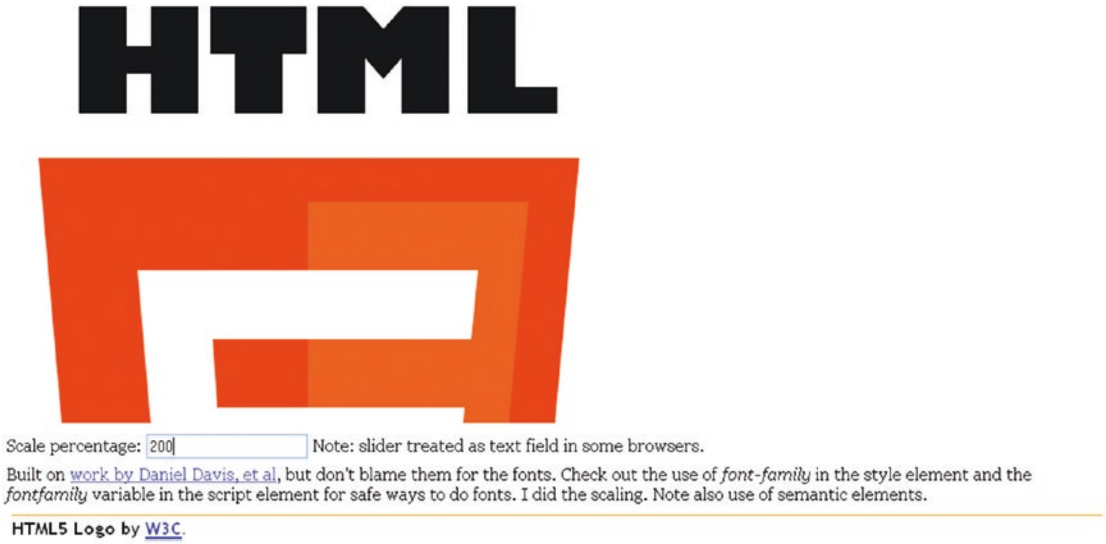
```
<input id="slide" type="range" min="0" max="100" value="100"➥
 onChange="changescale(this.value)" step="10"/>
```

The `min`, `max`, (initial) `value`, and `step` can be set to whatever you like. Since I was using percentage and since I did not want the logo to get bigger than the initial value or deal with negative values, I used 0 and 100.

In the proper implementation of the slider, the viewer does not see the initial value or the maximum or minimum. My code uses the input as a percentage. The expression `this.value` is interpreted as the value attribute of *this* element, emphasis given to convey the switch to English! The term `this` has special meaning in JavaScript and several other programming languages. The `changescale` function takes the value, specified by the parameter given in the assignment to the `onChange` attribute, and uses it to set a global variable (a variable declared outside of any function so it persists and is available to any function) named `factorvalue`.

```
function changescale(val) {
        factorvalue = val / 100;
        dologo();
}
```

It is part of the specification of HTML5 that the browsers will provide form validation, that is, browsers will check that the conditions specified by attributes in the input elements are obeyed. This can be a significant productivity boost in terms of reducing the work programmers need to do and a performance boost since the checking probably would be faster when done by the browser. In the HTML5 logo project, an advantage of the slider is that the viewer does not need to be concerned with values but merely moves the device. There is no way to input an illegal value. Figure 1-6 shows the results of entering a value of 200 in the input field.



Scale percentage: 200 Note: slider treated as text field in some browsers.

Built on work by Daniel Davis, et al, but don't blame them for the fonts. Check out the use of *font-family* in the style element and the *fontfamily* variable in the script element for safe ways to do fonts. I did the scaling. Note also use of semantic elements.

HTML5 Logo by W3C.

Figure 1-6. *Display in Firefox of scale of 200*

The canvas is of fixed width and height and drawing outside the canvas, which is what is done when the scaling is done to accept numbers and stretch them out to twice the original value, is ignored.

Building the Application and Making It Your Own

The project does one thing, it draws the logo. A function, dologo, is defined for this purpose. Informally, the outline of this program is

1. init*: Initialization*
2. dologo*: Draw the logo starting with the HTML letters and then the shield*
3. changescale*: Change the scale*

Table 1-1 shows the relationship of the functions. The dologo function is invoked when the document is first loaded and then whenever the scale is changed.

Table 1-1. *Functions in the HTML5 Logo Project*

Function	Invoked/Called By	Calls
init	Invoked by action of the onLoad attribute in the <body> tag	dologo
dologo	Invoked by init and changescale	
changescale	Invoked by action of the onChange attribute in the <input type="range"...> tag	dologo

The coding for the dologo function puts together the techniques previously described. In particular, the code brings back the original coordinate system and clears off the canvas.

The global variables in this application are

```
var ctx;
var factorvalue = 1;
var fontfamily = "65px 'Gill Sans Ultra Bold', sans-serif";
```

As indicated earlier, it would be possible to not use the fontfamily but use the string directly in the code. It is convenient to make ctx and factorvalue global.