



# The Essential Guide to HTML5

Using Games to Learn HTML5  
and JavaScript

—  
*Third Edition*

—  
Jeanine Meyer

Apress®

# **The Essential Guide to HTML5**

**Using Games to Learn HTML5  
and JavaScript**

**Third Edition**

**Jeanine Meyer**

**Apress®**

## ***The Essential Guide to HTML5: Using Games to Learn HTML5 and JavaScript***

Jeanine Meyer  
Purchase, NY, USA

ISBN-13 (pbk): 978-1-4842-8721-7  
<https://doi.org/10.1007/978-1-4842-8722-4>

ISBN-13 (electronic): 978-1-4842-8722-4

Copyright © 2023 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: Divya Modi  
Development Editor: James Markham  
Coordinating Editor: Divya Modi  
Copy Editor: Kim Wimpsett

Cover designed by eStudioCalamar

Cover image designed by Freepik ([www.freepik.com](http://www.freepik.com))

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@springersbm.com](mailto:orders-ny@springersbm.com), or visit [www.springeronline.com](http://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 [booktranslations@springernature.com](mailto:booktranslations@springernature.com); for reprint, paperback, or audio rights, please e-mail [bookpermissions@springernature.com](mailto:bookpermissions@springernature.com).

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 [www.apress.com/bulk-sales](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 <https://github.com/Apress/The-Essential-Guide-to-HTML5-3rd-Edition-by-Jeanine-Meyer>. For more detailed information, please visit [www.apress.com/source-code](http://www.apress.com/source-code).

Printed on acid-free paper

*To Annika, Daniel, Aviva, and Anne, and to Esther and Joseph,  
who are still in our lives.*

# Table of Contents

**About the Author ..... xi**

**About the Technical Reviewer ..... xiii**

**Acknowledgments ..... xv**

**Introduction ..... xvii**

**Chapter 1: The Basics ..... 1**

Keywords ..... 1

Introduction..... 1

Critical Requirements ..... 4

HTML5, CSS, and JavaScript Features ..... 4

Basic HTML Structure and Tags ..... 4

Using Cascading Style Sheets..... 10

JavaScript Programming ..... 14

Using a Text Editor ..... 16

Building the Applications ..... 19

Testing and Uploading the Application..... 25

Summary..... 26

**Chapter 2: Dice Game ..... 27**

Introduction..... 27

Critical Requirements ..... 30

HTML5, CSS, and JavaScript Features ..... 31

Pseudorandom Processing and Mathematical Expressions ..... 32

Variables and Assignment Statements ..... 33

Programmer-Defined Functions..... 35

Conditional Statements: *if* and *switch* ..... 37

## TABLE OF CONTENTS

Drawing on the Canvas .....	40
Displaying Text Output Using a Form .....	51
Building the Application and Making It Your Own .....	52
Throwing a Single Die .....	54
Throwing Two Dice .....	61
The Complete Game of Craps.....	68
Making the Application Your Own .....	76
Testing and Uploading the Application .....	78
Summary.....	79
<b>Chapter 3: Bouncing Ball .....</b>	<b>81</b>
Introduction.....	81
Critical Requirements .....	86
HTML5, CSS, and JavaScript Features .....	87
Drawing a Ball or an Image or Images.....	88
Gradients with a Side Trip to Explain Arrays .....	91
Setting Up a Timing Event.....	96
Calculating a New Position and Collision Detection.....	98
Starting, Positioning and Restarting the video with use of an anonymous function .....	102
Validation .....	102
Stopping and Resuming Animation Triggered by Buttons .....	104
HTML Page Reload .....	105
Preloading Images .....	105
Building the Application and Making It Your Own .....	106
Testing and Uploading the Application .....	129
Summary.....	130
<b>Chapter 4: Cannonball and Slingshot .....</b>	<b>131</b>
Introduction.....	131
Critical Requirements .....	135
HTML5, CSS, and JavaScript Features .....	136
Arrays and Programmer-Defined Objects .....	137

Rotations and Translations for Drawing .....	139
Drawing Line Segments .....	145
Mouse Events for Pulling on the Slingshot .....	147
Changing the List of Items Displayed Using Array Splice.....	149
Distance Between Points .....	150
Building the Application and Making It Your Own .....	151
Cannonball: With Cannon, Angle, and Speed .....	158
Slingshot: Using a Mouse to Set Parameters of Flight.....	168
Testing and Uploading the Application .....	179
Summary.....	180
<b>Chapter 5: The Memory (aka Concentration) Game.....</b>	<b>181</b>
Introduction.....	181
Critical Requirements .....	188
HTML5, CSS, JavaScript Features .....	189
Representing Cards.....	190
Using Date for Timing.....	191
Providing a Pause .....	192
Drawing Text .....	193
Drawing Polygons .....	196
Shuffling Cards .....	198
Implementing Clicking on a Card .....	199
Preventing Certain Types of Cheating .....	200
Building the Application and Making It Your Own .....	201
Testing and Uploading the Application.....	223
Summary.....	224
<b>Chapter 6: Quiz .....</b>	<b>225</b>
Introduction.....	225
Critical Requirements for a Quiz Game .....	230
HTML5, CSS, and JavaScript Features .....	231

TABLE OF CONTENTS

Storing and Retrieving Information in Arrays..... 231

Creating HTML During Program Execution..... 234

Using CSS in the Style Element..... 236

Responding to Player Moves..... 237

Presenting Audio and Video ..... 238

Checking the Player’s Answer..... 240

Building the Application and Making It Your Own ..... 241

Testing and Uploading the Application..... 252

Summary..... 253

**Chapter 7: Mazes..... 255**

Keywords ..... 255

Introduction..... 255

Critical Requirements ..... 263

HTML5, CSS, and JavaScript Features ..... 264

Representation of Walls and the Token..... 264

Mouse Events to Build and Position a Wall ..... 265

Detecting the Arrow Keys..... 266

Collision Detection: Token and Any Wall..... 268

Using Local Storage ..... 271

Encoding Data for Local Storage..... 279

Radio Buttons..... 281

Building the Application and Making It Your Own ..... 282

Creating the Travel Maze Application ..... 295

Testing and Uploading Application ..... 306

Summary..... 307

**Chapter 8: Rock, Paper, Scissors..... 309**

Introduction..... 309

Critical Requirements ..... 313

HTML5, CSS, and JavaScript Features ..... 314



Providing Graphical Buttons for the Player .....	314
Generating the Computer Move .....	319
Displaying Results Using Animation.....	325
Audio and DOM Processing.....	329
Starting Off.....	331
Building the Application and Making It Your Own .....	332
Testing and Uploading the Application .....	342
Summary.....	343
<b>Chapter 9: Guess a Word .....</b>	<b>345</b>
Introduction.....	345
Critical Requirements .....	352
HTML5, CSS, and JavaScript Features .....	353
Storing a Word List as an Array Defined in an External Script File .....	353
Generating and Positioning HTML Markup, Then Changing the Markup to Buttons, and Then Disabling the Buttons .....	354
Creating the Feedback About Remaining Wrong Letters.....	358
Maintaining the Game State and Determining a Win or Loss.....	359
Checking a Guess and Revealing Letters in the Secret Word by Setting <code>textContent</code> .....	360
Building the Application and Making It Your Own .....	361
Testing and Uploading the Application .....	371
Summary.....	371
<b>Chapter 10: Blackjack .....</b>	<b>373</b>
Introduction.....	373
Critical Requirements .....	380
HTML5, CSS, and JavaScript Features .....	381
Source for Images for Card Faces and Setting Up the Image Objects .....	381
Creating the Programmer-Defined Object for the Cards .....	382
Starting a Game .....	383
Dealing the Cards.....	384
Shuffling the Deck.....	389

TABLE OF CONTENTS

Capturing Key Presses ..... 390

Using Header and Footer Element Types ..... 392

Building the Application and Making It Your Own ..... 393

Testing and Uploading the Application ..... 409

Summary..... 410

**Appendix: More Techniques for Drawing..... 411**

    Circles and Arrows ..... 411

        Overview..... 413

        Details of Implementation ..... 414

        What You Learned..... 430

    Crossing a Line (Jumping a Fence)..... 430

        Overview..... 434

        Details of Implementation ..... 438

    Using Scalar Vector Graphics ..... 451

    Using SVG to Draw the HTML5 Logo ..... 452

        Overview..... 456

        Details of Implementation ..... 457

    Using SVG to Draw and Modify a Cartoon ..... 465

        Overview..... 466

        Details of Implementation ..... 467

**Index..... 479**

# About the Author



**Jeanine Meyer** is a professor emerita at Purchase College/SUNY and past coordinator of the Mathematics/Computer Science Board of Study. Before Purchase, she taught at Pace University and before that worked as a research staff member and manager in robotics and manufacturing research at IBM Research and as a consultant for IBM's educational grant programs. She is the author or coauthor

of ten books on topics such as educational uses of multimedia, programming (three published by Apress/Springer), databases, number theory, and origami.

She earned a PhD in computer science at the Courant Institute at New York University, an MA in mathematics at Columbia University, and an SB (the college used the Latin form) in mathematics from the University of Chicago. She is a member of Phi Beta Kappa, Sigma Xi, Association for Women in Science, and Association for Computing Machinery, and was a featured reviewer for ACM Computing Reviews.

For Jeanine, programming is both a hobby and a vocation. Every day she plays computer puzzles online (including Words with Friends, various solitaire card games, and Duolingo for Spanish, which she views as a game). She also participates in Daf Yomi, the seven-and-a-half-year study of Talmud, which certainly has puzzle-solving aspects. She tries *The New York Times* crossword puzzle many days but does better at the mini-puzzle, KenKen, and Two Not Touch, in which she sometimes competes with her children. She enjoys cooking, baking, eating, gardening, travel, and a moderate amount of walking. She misses her mother, who inspired many family members to take up piano, and her father, who gave Jeanine a love of puzzles. She is an active volunteer for progressive causes and candidates.

# About the Technical Reviewer



**Vadim Atamanenko** is a software developer with more than 20 years of experience. He participates in international hackathons both as a judge and as a mentor and is a member of the Harvard Square Business Association.

He has developed many complex solutions in various business areas that have helped thousands of people automate manual processes.

Currently he is the CIO at Freedom Holding Corp., but he still finds time to regularly participate in international IT conferences.

He enjoys meeting new people and sharing his knowledge. If you have a question for him, visit <https://www.linkedin.com/in/vadim-atamanenko/>.

# Acknowledgments

Much appreciation to my students and colleagues at Purchase College/State University of New York for their inspiration, stimulation, and support; and to family and friends who indulge me in my use of family photos and video clips for my courses and my books.

Thanks to the crew at Apress and Springer for all their efforts.

# Introduction

When it was first released, there was considerable enthusiasm about the new capabilities of HTML5, and even suggestions that no other technologies or products are necessary to produce dynamic, engrossing, interactive websites. The excitement has not gone away, and the new features are still exciting. HTML is HTML5. It now is possible, using just HTML, Cascading Style Sheets, and JavaScript, to draw lines, arcs, circles, and ovals on the screen and specify events and event handling to produce animation and respond to user actions. You can include video and audio on your website with standard controls, and you can include the video or audio in your application exactly when and where needed. You can create forms that validate the input and provide immediate feedback to users. You can use a facility similar to cookies to store information on the client computer. And you can use new elements, such as headers and footers, to help structure your documents. HTML, CSS, and JavaScript work together. You can use JavaScript to create new HTML elements, and this is helped by what can be done with CSS.

This book is based on my teaching practices and past writings. Delving into the features of a technology or general programming concepts is best done when there is a need and a context. Games, especially familiar and simple ones, supply the context and thus the motivation and much of the explanation. When learning a new programming language, one of my first steps is to program the game of craps. Also, if I can build a ballistics simulation with animation, such as the slingshot game, and make a video or audio clip play when a specific condition occurs, I am happy. If I can construct my own maze of walls, determine ways to provide visual as well as text feedback, and store information on the player's computer, I am ecstatic. That's what we will do in this book. As you learn how to build these simple games, you'll build your expertise as well. I hope you go on to make your own exciting, compelling applications.

This goal of this book, developed with considerable help from the Apress staff and the technical reviewers, is to introduce you to programming, with the motivation of implementing interactive websites to share with others.

## INTRODUCTION

- At the time of updating this book, browser support for HTML5 features is close to complete. The applications have been tested using Chrome and Safari. However, it is important to keep in mind that browsers can change.
- My focus is on plain HTML and JavaScript because it has been my experience that knowledge and experience with the basics is the best introduction. Frameworks and libraries exist and continue to be developed and refined, and at some point, these tools are appropriate to study. This is especially true if you work in an organization that has adopted specific tools. You can turn to these topics after getting comfortable with the basics. Note that I have updated my *HTML5 and JavaScript Projects* book, which is a step up from this one in level of complexity.

## Who Is This Book For?

This book is for people who want to learn how HTML, JavaScript, and Cascading Style Sheets can serve to build dynamic, exciting websites. It's for you if you know something about programming and want to see what the current versions of HTML and JavaScript offer. It's also for you if you have no programming experience whatsoever. Perhaps you're a web designer or website owner and you want to know how to make things happen behind the scenes or how to request features from programmers.

With this book, we want to showcase the new(er) features of HTML5 and demystify the art of programming. Programming is an art, and creating appealing games and other applications requires talent and attention to the audience. However, if you can put together words to form sentences and put together sentences to form paragraphs, then you can program.

## How Is This Book Structured?

The book consists of ten chapters plus an appendix, each organized around a familiar game or similar application. There is considerable redundancy in the chapters, so you can skip around if you like, though the games do get more complex as the book progresses. Each chapter starts by describing the application and listing the technical features and programming concepts that will be covered. We look first at the critical requirements in a

general sense: what we need to implement the application, independent of any specific technology. We then focus on the features of HTML5, JavaScript, Cascading Style Sheets, and general programming methodologies that satisfy the requirements. Finally, we examine the implementation of each application in detail. I break out the code line by line in a table, with comments next to each line. In the cases where multiple versions of a game are described, only the new lines of code may be explained in detail. This isn't to deprive you of information but to encourage you to see what is similar and what is different, and to demonstrate how you can build applications in stages. It certainly is appropriate to consult the commented programs on an as-needed basis. Each chapter includes suggestions on how to make the application your own and how to test and upload the application to a website. The summary at the end of each chapter highlights what you've learned and what you'll find ahead.

The appendix was added in this edition to provide more advanced examples of creating and manipulating graphics on the screen using algebra and geometry and Scalar Vector Graphics images.

## Conventions Used in This Book

The applications in this book are HTML documents. The JavaScript is in a `script` element in the head element, and the CSS is in the `style` element in the head element. The body element contains the static HTML, including any canvas elements. Several examples depend on external image files, and one example requires external video files and audio files and another external audio files.

## Layout Conventions

To keep this book as clear as possible, the following text conventions are used throughout:

- Code is presented in `fixed-width font`.
- The complete code for each application is presented in a table, with each statement explained with a comment.
- Pseudocode is written in *italic fixed-width font*.
- Sometimes code won't fit on a single line in a book. Where this happens, I use an arrow like this: ↪.

So, with these formalities out of the way, let's get started.



# CHAPTER 1

# The Basics

## Keywords

HTML Document; HTML Structure; Hypertext Markup Language (HTML); HTML File; Cascading Style Sheets (CSS).

In this chapter, we cover the following:

- The basic structure of an HTML document
- The `html`, `head`, `title`, `script`, `style`, `body`, `img`, and `a` elements
- A Cascading Style Sheet (CSS) example
- A JavaScript code example, using `Date` and `document.write`

## Introduction

Hypertext Markup Language (HTML) is the language for delivering content on the Web. HTML is not owned by anyone but is the result of people working in many countries and many organizations to define the features of the language. An HTML document is a text document that you can produce using any text editor. HTML documents contain elements surrounded by tags—text that starts with a `<` symbol and ends with a `>` symbol. An example of a tag is ``. This particular tag will display the image held in the file `home.gif`. These tags are the *markup*. It is through the use of tags that hyperlinks, images, and other media are included in web pages.

Basic HTML can include directives for formatting in a language called Cascading Style Sheets (CSS) and programs for interaction in a language called JavaScript. Browsers, such as Firefox and Chrome, interpret the HTML along with any CSS and JavaScript to produce what we experience when we visit a website. HTML holds the

content of the website, with tags providing information on the nature and structure of the content as well as references to images and other media. CSS specifies the formatting. The same content can be formatted in different ways. JavaScript is a programming language that's used to make the website dynamic and interactive. In all but the smallest working groups, different people may be responsible for the HTML, CSS, and JavaScript, but it's always a good idea to have a basic understanding of how these different tools work together. If you are already familiar with the basics of HTML and how CSS and JavaScript can be added together, you may want to skip ahead to the next chapter. Still, it may be worth casting your eye over the content in this chapter to make sure you are up to speed on everything before we start on the first core examples.

The latest version of HTML (and its associated CSS and JavaScript) is HTML5. It has generated considerable excitement because of features such as the canvas for displaying pictures and animation; support for video and audio; and tags for defining common document elements such as header, section, and footer. You can create a sophisticated, highly interactive website with HTML5. As of this writing, not all browsers accept all the features, but you can get started learning HTML5, CSS, and JavaScript now. Learning JavaScript will introduce you to general programming concepts that will be beneficial if you try to learn any other programming language or if you work with programmers as part of a team.

The approach I'll use in this book is to explain HTML5, CSS, and JavaScript concepts in the context of specific examples, most of which will be familiar games. Along the way, I'll use small examples to demonstrate specific features. Ideally, this will help you both understand what you want to do and appreciate how to do it. You will know where we are headed as I explain the concepts and details.

The task for this chapter is to build a web page of links to other websites. In this way, you'll get a basic understanding of the structure of an HTML document, with a small amount of CSS code and JavaScript code. For this and other examples, please think of how to make the project meaningful to you. The page could be a list of your own projects, favorite sites, or sites on a particular topic. For each site, you'll see text and a hyperlink. The second example includes some extra formatting in the form of boxes around the text, pictures, and the day's date and time. Figure 1-1 and Figure 1-2 show examples I've created.

# My games

The [Dice game](#) presents the game called craps.

The [Cannonball](#) is a ballistics simulation. A ball appears to move on the screen in an arc. The program determines when it hits the ground or the target. The player can adjust the speed and the angle.

The [Slingshot](#) simulates shooting a slingshot. A ball moves on the screen, with the angle and speed depending on how far the player has pulled back on the slingshot using the mouse.

The [Concentration/memory game](#) presents a set of plain rectangles you can think of as the backs of cards. The player clicks on first one and then another and pictures are revealed. If the two pictures represent a match, the two cards are removed. Otherwise, the backs are displayed. The game continues until all matches are made. The time elapsed is calculated and displayed.

The [Quiz game](#) presents the player with 4 boxes holding names of countries and 4 boxes holding names of capital cities. These are selected randomly from a larger list. The player clicks to indicate matches and the boxes moved to make the guessed boxes be together. The program displays whether or not the player is correct.

The [Maze](#) program is a multi-stage game. The player builds a maze by using the mouse to build walls. The player then can move a token through the maze. The player also can save the maze on the local computer using a name chosen by the player and retrieve it later, even after closing the browser or even turning off the computer.

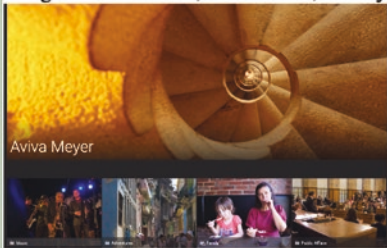
Figure 1-1. An annotated list of games

Sun Jul 10 2022 14:05:15 GMT-0400 (Eastern Daylight Time)

## Favorite Sites

The [website for Purchase College/State University of New York](#).

The [Aviva Meyer's photographs](#) site is a collection of Aviva's photographs stored on a site called smugmug. The categories are Music, Adventures, Family (which requires a password) and others.



[Apress publishers](#) is the site for the publishers of this book.




Figure 1-2. Favorite sites, with extra formatting

When you reload the Favorite Sites page, the date and time will change to the current date and time according to your computer.

## Critical Requirements

The requirements for the list of links application are the very fundamental requirements for building a web page containing text, links, and images. For the example shown in Figure 1-1, each entry appears as a paragraph. In the example shown in Figure 1-2, in contrast, each entry has a box around it. The second example also includes images and a way to obtain the current day, date, and time. Later applications will require more discussion, but for this one we'll go straight to how to implement it using HTML, CSS, and JavaScript.

## HTML5, CSS, and JavaScript Features

As I noted, HTML documents are text, so how do we specify links, pictures, formatting, and coding? The answer is in the markup, that is, the tags. Along with the HTML that defines the content, you'll typically find CSS styles, which can be specified either inside the HTML document or in an external document. You also might include JavaScript for interactivity, again specified in the HTML document or in an external document. We'll start with a look at how you can build simple HTML tags and how you can add inline CSS and JavaScript all within the same document.

## Basic HTML Structure and Tags

An HTML element begins with a starting tag, which is followed by the element content and an ending tag. The ending tag includes a / symbol followed by the element type, for example /head. Elements can be nested within elements. A standard HTML document looks like this:

```
<html>
  <head>
    <title>Very simple example
  </title>
  </head>
```

```

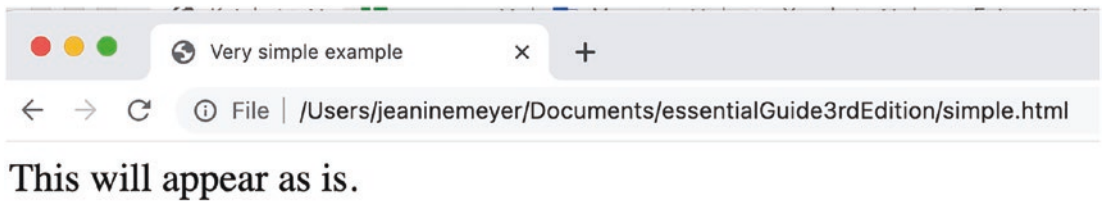
<body>
  This will appear as is.
</body>
</html>

```

Note that I’ve indented the nested tags here to make them more obvious, but HTML itself ignores this indentation (or whitespace, as it’s known), and you don’t need to add it to your own files. In fact, for most of the examples throughout this book, I don’t indent my code.

This document consists of the `html` element, indicated by the starting tag `<html>` and ending with the closing tag: `</html>`.

HTML documents typically have a `head` and a `body` element, as this one has. This `head` element contains one element, `title`. The HTML title shows up different places in different browsers. Figure 1-3 shows the title, “Very Simple Example,” on a tab in Chrome.



**Figure 1-3.** The HTML title on a tab in the Chrome browser

In most cases, you will create something within the `body` of the web page that you’ll think of as a title, but it won’t be the HTML title! Figure 1-3 also shows the `body` of the web page: the short piece of text. Notice that the words *html*, *head*, *title*, and *body* do not appear. The tags “told” the browser how to display the HTML document.

We can do much more with text, but let’s go on to see how to get images to appear. This requires an `img` element. Unlike the `html`, `head`, and `body` elements that use starting and ending tags, the `img` element just uses one tag. It is called a *singleton* tag. Its element type is `img` (not *image*), and you put all the information within the tag itself using what are termed *attributes*. What information? The most important item is the name of the file that holds the image. The tag

```

```

tells the browser to look for a file with the name *frog* and the file type *.jpg*. In this case, the browser looks in the same directory or folder as the HTML file. You can also refer to image files in other places, and I'll show this later. The `src` stands for source. It is termed an attribute of the element. The slash before the `>` indicates that this is a singleton tag. There are common attributes for different element types, but most element types have additional attributes. Another attribute for `img` elements is the `width` attribute.

```

```

This specifies that the image should be displayed with a width of 200 pixels. The height will be whatever is necessary to keep the image at its original aspect ratio. If you want specific widths and heights, even if that may distort the image, specify both `width` and `height` attributes.

---

**Tip** You'll see examples (maybe even some of mine) in which the closing slash is missing that work just fine. It is considered good practice to include it. Similarly, you'll see examples in which there are no quotation marks around the name of the file. HTML is more forgiving in terms of syntax (punctuation) than most other programming systems. Finally, you'll see HTML documents that start with a tag of type `!DOCTYPE` and have the HTML tag include other information. At this point, we don't need this, so I will keep things as simple as I can (but no simpler, to quote Einstein).

---

Producing hyperlinks is similar to producing images. The type of element for a hyperlink is `a`, and the critical attribute is `href`.

```
<a href=http://www.purchase.edu>Purchase College website</a>
```

As you can see, this element has a starting and ending tag. The content of the element, whatever is between the two tags—in this case, *Purchase College website*—is what shows up in blue and is underlined. The starting tag begins with `a`. One way to remember this is to think of it as the most important element in HTML so it uses the first letter of the alphabet. You can also think of an anchor, which is what the `a` actually stands for, but that isn't as meaningful for me. The `href` attribute (think hypertext reference) specifies the website where the browser goes when the hyperlink is clicked. Notice that this is a full web address (called a Universal Resource Locator, or URL, for short).

Web addresses can be absolute or relative. An absolute address starts with `http://`. A relative address is relative to the location of the HTML file. Using relative addressing makes it easier to move your project to a different website, and you can indicate the folder one level up by using

`../`

at the start of the reference. In the favorite sites example, the `avivasmugmug.png` file and the `apressshot.png` file are located in the same folder as the HTML file. They are there because I put them there! For large projects, many people put all the images in a subfolder called `images` and write addresses as `images/postcard.gif`. File management is a big part of creating web pages.

We can combine a hyperlink element with an `img` element to produce a picture on the screen that a user can click. Remember that elements can be nested within other elements. Instead of putting text after the starting `<a>` tag, put an `<img>` tag:

```
<a href="http://apress.com">

</a>
```

Let's put these concepts to work in another example:

```
<html>
<head>
<title>Second example </title>
</head>
<body>
This will appear as is.



<a href=http://faculty.purchase.edu/jeanine.meyer>Jeanine Meyer's Academic
  Activities </a>
<a href=http://faculty.purchase.edu/jeanine.meyer></a>
</body>
</html>
```



I created the HTML file, saved it as `second.html`, and then opened it in the Chrome browser. Figure 1-4 shows what is displayed.



**Figure 1-4.** Example with images and hyperlinks

This produces the text; the image in its original width and height; the image with the width fixed at 200 pixels and height proportional; a hyperlink that will take you to the Purchase College website; and another link that uses an image that will take you to the web page on the Purchase College website for the Mathematics/Computer Science department. However, this isn't quite what I had in mind. I wanted these elements spaced down the page.

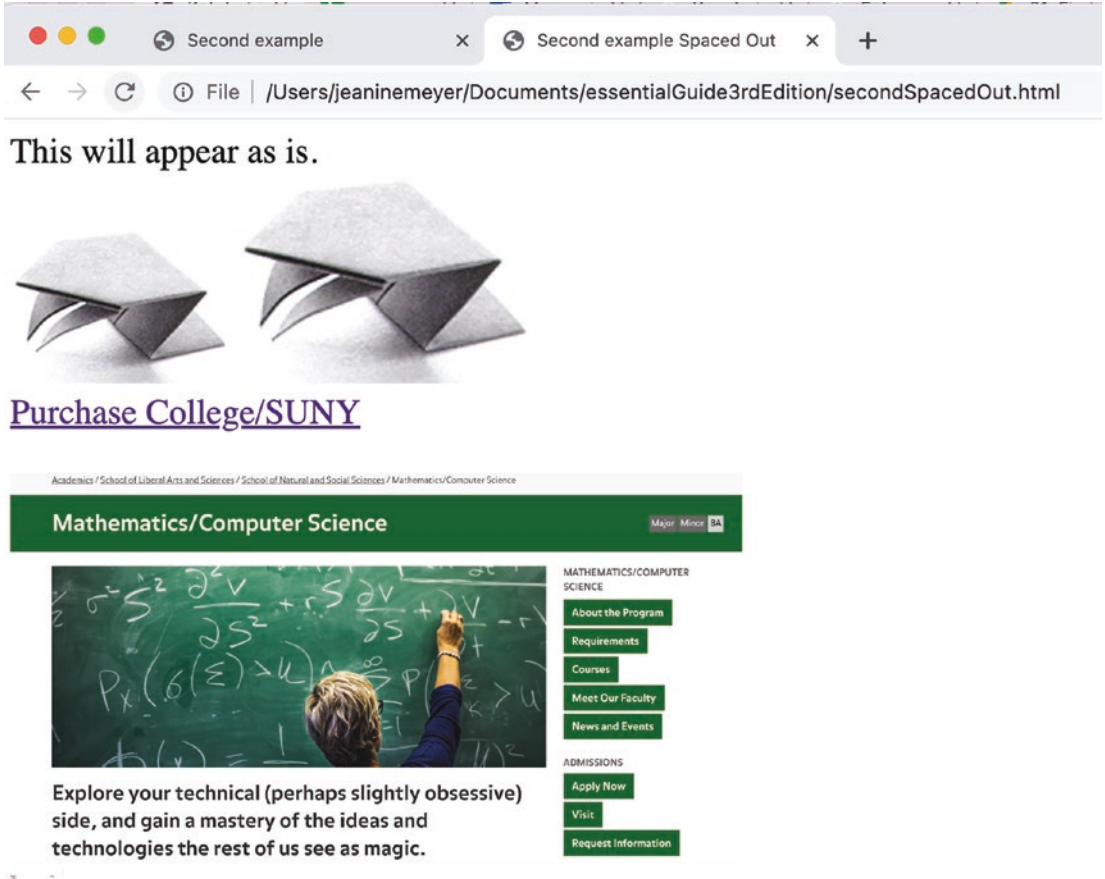
This demonstrates something you need to remember: HTML ignores line breaks and other whitespace. If you want a line break, you have to specify it. One way is to use the `br` singleton tag. I'll show other ways later. Take a look at the following modified code. Notice that the `<br/>` tags don't need to be on a line by themselves.

```
<html>
<head>
<title>Second example Spaced Out</title>
</head>
<body>
This will appear as is. <br/>

 <br/>
<a href=Error! Hyperlink reference not valid. College/SUNY</a><br/> <br/>
<a href=https://www.purchase.edu/academics/mathematics-computer-science/>
</a>
</body>
</html>
```



Figure 1-5 shows what this code produces. Notice that I changed the title. I also decided to leave the origami frog images together, and I put two `<br />` tags after the link to Purchase College/SUNY.



**Figure 1-5.** Text, images, and links with line breaks

There are many HTML element types: the `h1` through `h6` heading elements produce text of different sizes; there are various elements for lists and tables, and others for forms. CSS, as we'll see in a moment, is also used for formatting. You can select different fonts, background colors, and colors for the text, and control the layout of the document. It's considered good practice to put formatting in CSS, create interactivity in JavaScript, and keep the HTML for the content. HTML5 provides new structural elements—such as `article`, `section`, `footer`, and `header`—putting formatting into the `style` element and making use of the new elements, called *semantic* tags, to facilitate working with

other people. However, even when you're working just with yourself, separating content, formatting, and behavior lets you easily change the formatting and the interactions. Formatting, including document layout, is a large topic. In this book, I stick to the basics.

## Using Cascading Style Sheets

CSS is a special language just for formatting. A style is essentially a rule that specifies how a particular element will be formatted. This means you can put style information in a variety of places: a separate file, a style element located in the head element, or a style within the HTML document, perhaps within the one element you want to format in a particular way. The styling information cascades, or trickles down, unless a different style is specified. To put it another way, the style closest to the element is the one that's used. For example, you might use your official company fonts as given in the style section in the head element to flow through most of the text but include a specification within the local element to style one particular piece of text. Because that style is closest to the element, it is the one that is used.

The basic format includes an indicator of what is to be formatted followed by one or more directives. In the examples for this chapter, I'll specify the formatting for elements of type `section`, namely, a border or box around each item, margins, padding, alignment, and a background of white. The complete HTML document in Listing 1-1 is a mixture (some would say a mess!) of features. The elements `body` and `p` (paragraph) are part of the original version of HTML. The `section` element is one of the new element types added in HTML5. The `section` element does need formatting, unlike `body` and `p`, which have default formatting that the `body` and each `p` element will start on a new line. CSS can modify the formatting of old and new element types. Notice that the background color for the text in the section is different from the background color for the text outside the section.

In the code in Listing 1-1, I specify styles for the `body` element (there is just one) and the `section` element. If I had more than one `section` element, the styling would apply to each of them. The style for the `body` specifies a background color and a color for the text. In the beginning, browsers accepted a set of only 16 colors by name, including black, white, red, blue, green, cyan, and pink. However, now the up-to-date browsers accept 140 colors by name.

See [https://www.w3schools.com/colors/colors\\_names.asp](https://www.w3schools.com/colors/colors_names.asp).

You can also specify color using RGB (red, green, blue) hexadecimal codes, but you'll need to use a graphics program—such as Adobe Photoshop, Corel Paint Shop Pro, or Adobe Flash Professional—to figure out the RGB values, or you can experiment. I used Paint Shop Pro to determine the RGB values for the green in the frog head picture and used that for the border as well.

The text-align directives are just what they sound like: they indicate whether to center the material or align it to the left. The font-size sets the size of text in pixels. Borders are tricky and don't appear to be consistent across browsers. Here I've specified a solid green border of 4 pixels. The width specification for section indicates that the browser should use 85 percent of the window, whatever that is. The specification for p sets the width of the paragraph at 250 pixels. Padding refers to the spacing between the text and the borders of the section. The margin is the spacing between the section and its surroundings.

***Listing 1-1.*** A Complete HTML Document with Styles

```
<html>
<head>
<title>CSS example </title>
<style>
body {
    background-color:tan;
    color: #660000;
    text-align:center;
    font-size:22px;
}
section {
    width:85%;
    border:4px #00FF63 solid;
    text-align:left;
    padding:5px;
    margin:10px;
    background-color: white;
}
p {
    width: 75%;
}
```

```

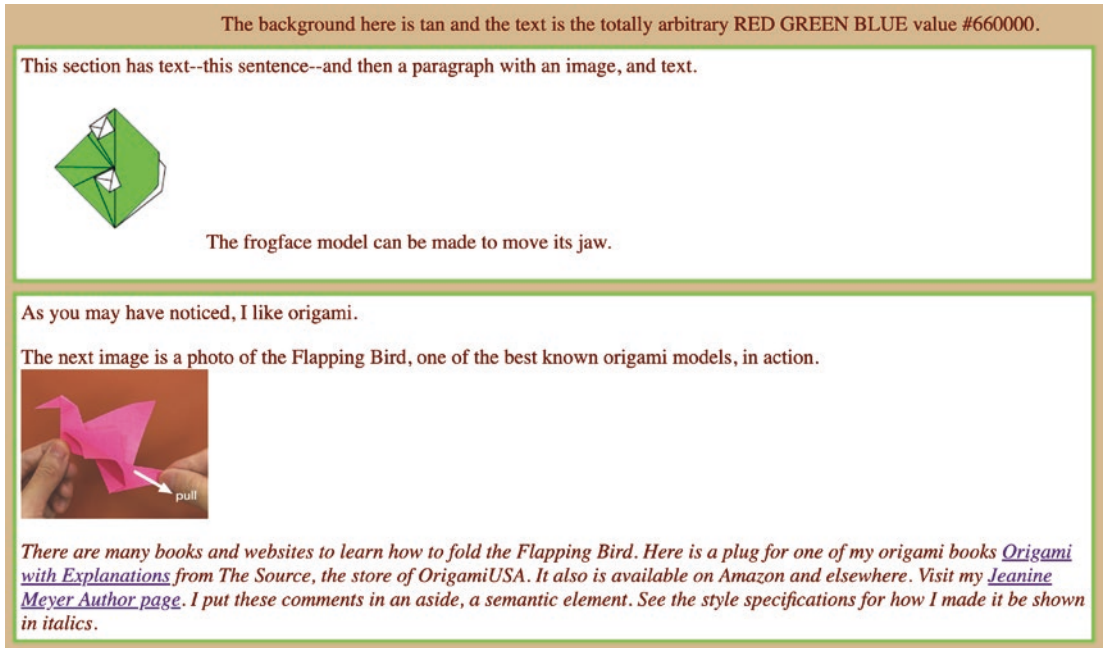
aside {
    font-style: italic;
}
</style>
</head>
<body>
The background here is tan and the text is the totally arbitrary RED
GREEN BLUE
    value #660000. <br/>
<section>
This section has text--this sentence--and then a paragraph with an image,
and text.
<p>
     The frogface model can be made to move its jaw.
</p>
</section>

<section>
As you may have noticed, I like origami. <p> The next image is a photo of
the Flapping Bird, one of the best known origami models, in action.
    <br/>
 </p>

<aside>There are many books and websites to learn how to fold the Flapping
Bird. Here is a plug for one of my origami books <a href="https://
origamiusa.org/catalog/products/origami-explanations">Origami with
Explanations</a> from The Source, the store of OrigamiUSA. It also is
available on Amazon and elsewhere. Visit my <a href="https://www.amazon.
com/Jeanine-Meyer/e/B001JPA5SC%3Fref=db_s_a_mng_rwt_scns_share">Jeanine
Meyer Author page</a>. I put these comments in an aside, a semantic
element. See the style specifications for how I made it be shown in
italics.
</aside>
</section>
</body>
</html>

```

This produces the screen shown in Figure 1-6.



**Figure 1-6.** Sample use of CSS styles

---

**Tip** Don't be concerned if you don't understand everything immediately. Modify these examples and make up your own. You'll find lots of help on the Web. In particular, see the official source for HTML 5 at <http://dev.w3.org/html5/spec/Overview.html>.

---

There are many things you can do with CSS. You can use it to specify formatting for types of elements, as shown here; you can specify that elements are part of a class; and you can identify individual elements using the `id` attribute. In Chapter 6, where we create a quiz, I use CSS to position specific elements in the window and then JavaScript to move them around.

# JavaScript Programming

JavaScript is a programming language with built-in features for accessing parts of an HTML document, including styles in the CSS element. It is termed a *scripting language* to distinguish it from compiled languages, such as C++. Compiled languages are translated all at once, prior to use, while scripting languages are interpreted line by line by browsers. This text assumes no prior programming experience or knowledge of JavaScript, but it may help to consult other books, such as *Getting Started with JavaScript*, by Terry McNavage (friends of ED, 2010), or online sources such as <http://en.wikipedia.org/wiki/JavaScript>.

Each browser owns its version of JavaScript.

An HTML document holds JavaScript in a `script` element, located in the head element. To display the time and date information as shown in Figure 1-2, I put the following in the head element of the HTML document:

```
<script>
document.write(Date());
</script>
```

JavaScript, like other programming languages, consists of statements of various types. In later chapters, I'll show you assignment statements, compound statements such as `if` and `switch` and `for` statements, and statements that create what are called *programmer-defined functions*. A function is one or more statements that work together in a block and can be called any time you need that functionality. Functions save writing out the same code over and over. JavaScript supplies many built-in functions. Certain functions are associated with objects (more on this later) and are called *methods*.

The code

```
document.write("hello");
```

is a JavaScript statement that invokes the `write` method of the `document` object with the argument `"hello"`. An argument is additional information passed to a function or method. Statements are terminated by semicolons. This piece of code will write out the literal string of characters *h, e, l, l, o* as part of the HTML document.

The `document.write` method writes out anything within the parentheses. Since I wanted the information written out to change as the date and time change, I needed a way to access the current date and time, so I used the built-in JavaScript `Date` function.