

VOLUME 3

Functional CSS



Dynamic
HTML
without
JavaScript

A PROJECT-BASED GUIDE

DR. DMITRIY A. NESTERKIN

Functional CSS

Dynamic HTML without JavaScript

VOLUME 3

Dr. Dmitriy A. Nesterkin

© 2015. Dmitriy A. Nesterkin, Ph.D. All rights reserved.

Contents

1. Introduction

[Preface](#)
[Should Volumes 1 and 2 Be Completed First?](#)
[Book Content and Organization](#)
[Intended Audience](#)
[Specific Advice for Working with this Book](#)
[Recommended Reading Media](#)
[Recommended Browsers](#)
[Development Environment](#)
[Code Conventions](#)
[Book Code](#)
[Errata](#)
[About the Author](#)

2. Site Template

[Introduction](#)
[Topics Covered](#)
[HTML Code](#)
[CSS Code](#)
[CSS Tabs](#)
[CSS Counters](#)
[Finishing Touches](#)
[Final Remarks](#)

3. Static Content

[Introduction](#)
[About Page HTML Code](#)
[About Page CSS Code](#)
[Education Page HTML Code](#)
[Education Page CSS Code](#)
[Skills Page HTML Code](#)

[Skills Page CSS Code](#)
[Experience Page HTML Code](#)
[Experience Page CSS Code](#)
[Projects Page HTML Code](#)
[Projects Page CSS Code](#)
[Final Remarks](#)

4. Image Gallery

[Introduction](#)
[Topics Covered](#)
[HTML Code](#)
[CSS Code](#)
[Border Images](#)
[Back to the CSS Code](#)
[Final Remarks](#)

5. Data Form with Input Validation

[Introduction](#)
[Topics Covered](#)
[HTML Code](#)
[CSS Code](#)
[Selector Specificity](#)
[Final Remarks](#)

6. Conclusion

1. Introduction

Preface

Previous volumes of the Functional CSS series have focused on building standalone interactive web components using only HTML and CSS. This book continues with and expands upon this theme and teaches users how to build new functional widgets in the context of a complete resume web site. To accommodate the new approach, several of the chapters are dedicated to static HTML and CSS, which is the necessary glue to assemble different sections of an online resume into a cohesive document. The example material for the site will be my own vita. A full amount of my professional content is provided to make this exercise as realistic as possible; the reader is welcome to adjust the content to suit his or her own needs.

The functional CSS components that are congruent with the resume theme are used. These include click-activated tabs, information-on-demand dropdown controls, image gallery, and data form with input validation and error highlighting. Given the current format, this text is the most involved volume of the series and is a **high-commitment** educational material, a content that demands study. Perusal of the chapters may work for some experts; however, beginners and intermediate-level developers should set aside the necessary time to work through the tutorials and to experiment with the examples.

Should Volumes 1 and 2 Be Completed First?

Yes, for three reasons. First, Volume 1 is foundational and covers in detail many fundamental aspects of HTML and CSS such as element types, selectors, floating, positioning, and text effects. Volume 3 revisits these topics, but substantially scales down the depth of explanation, featuring fewer code commentaries and screenshots. Readers new to HTML and CSS are especially encouraged to start with Volume 1. Second, Volumes 1, 2, and 3 cover HTML and CSS using completely different cases. Consideration of the same feature such as positioning across different contexts will only strengthen and deepen the reader's understanding of the functionality. And third, some features such as flexbox and 3D transforms are not explored in this volume and are covered in Volumes 1 and 2, respectively. So, go ahead and start with the previous volumes and then continue with this book.

Book Content and Organization

This text delves into HTML and CSS integratively. Each chapter focuses on a variety of HTML and CSS elements, concurrently, in the context of practical examples. Instead of dissecting each feature myopically, this book considers each HTML and CSS element in combination with other elements that are needed to make a given case work. For example, in chapter 5, explanations on how to build a data form with input validation and error highlighting are provided, and topics such as positioning, management of pointer events, text effects, selector specificity, and background image placement are covered. The key philosophical principle guiding this series is that even simple software systems are synergistic combinations of diverse technologies and that successful learning of these technologies is best facilitated via detailed and, importantly, immersive examples. Building of functional mini-applications requires the fluent use of a broad range of HTML and CSS vocabularies. The cases presented within these pages fully capture the richness of the HTML and CSS languages.

Because we will be building a complete resume website, it is highly recommended that the reader study the chapters in the order in which they appear. Because a number of HTML and CSS elements are common to several cases, there is some repetition within the examples. This redundancy is intentional: mastery of HTML and CSS is enhanced when the same set of their features is applied in different contexts.

Intended Audience

This book is suitable for both beginners and seasoned developers. Beginners and intermediate-level learners are highly encouraged to complete Volumes 1 and 2 before proceeding with this book.

Specific Advice for Working with this Book

Type the code. It goes without saying that, to learn coding, one must type the code a character at a time. There is no shortcut for that. Pasting the ready-made examples and observing the effect deprives an aspiring developer of an immersive opportunity to deepen his or her expertise. This book lists all of the code needed to implement every single case presented in each chapter; typing this code is necessary for optimal learning.

Type the code one line or one declaration at a time, save, and preview. This advice should be taken to heart, especially among beginners. Just a few lines of CSS code can instruct a browser to complete numerous tasks. Tracing a particular sequence of visual transformations requires that they be implemented in gradual progression.

Read all explanations. Great lengths have been taken to explain the effects of various CSS features. The reader is encouraged to thoroughly study the explanations that follow each block of code.

Use online references. The cases presented in this book draw on a wide variety of CSS features. Those features are explained in as much detail as necessary within the context of each example. The use of online HTML and CSS references is highly encouraged to obtain the most up-to-date and comprehensive overview of a particular specification.

Experiment. Extending or modifying each case with new functionality is advantageous to test your understanding of HTML and CSS. A very powerful way to solidify your knowledge is to create new functionalities drawing on the HTML and CSS elements considered in the particular example. So, if inspiration strikes you, follow it through and see where it takes you. Creating something new is likely to push you to the edge of your skill base. In the end, that is one of the most powerful ways to move towards a greater level of expertise.

Be mindful that the most common error you are likely to make is a simple spelling mistake. Forgetting an equal sign, semi colon, or closing quote; mistyping declarations; or capitalizing letters when no capitalization is needed

can undermine the intended behavior of an application. Keep in mind that CSS declarations are case-sensitive and, conventionally, all HTML is typed in lower case. If you are new to programming, then starting with HTML and CSS languages is a good first step toward habitualizing precision and care when creating software.

Compare your work to the provided screenshots. All cases are visually documented with multiple detailed screenshots that begin with a preview of a bare bones HTML page and end with a fully-fledged, working HTML and CSS application. As you work through a given case and see that your web page looks quite different from the corresponding screenshot, stop, find the mistake(s), and only then proceed with implementing the rest of the code.

Indent your code appropriately. Mistakes are more likely to creep in when your code is difficult to read and navigate. HTML documents are hierarchical, and the best way to indicate that hierarchy is through consistent indentation. CSS files are collections of rule sets, and proper indentation is also helpful when reading them.

Attempt to solve the problems that you will encounter on your own as much as possible. The Internet offers plenty of development community websites where you can obtain help on any topic. However, there is something to be said for developing your own pattern of diagnosing and correcting the software bugs. Of course, whenever you have exhausted all possible courses of personal action, ask for help. And, when able, give help to others as well.

Proceed as slowly as necessary, have heart, and bulldoze your way through the book. As an educator, I can confirm that the best students are not the ones who know the most at the beginning of a class, but rather those that have the most passion for the subject. The road to expertise is pretty simple, and the key element is not so-called talent, but rather the will to master a skill.

Most importantly, have fun with it! Coding is one of the most creative things you can do with instant effects. It is like LEGO™ blocks for adults. By coding, one can build real applications and have a good time doing it.

Recommended Reading Media

This book is a technical and tutorial-driven text. Successful reading and assimilation of the material presented here requires a high-resolution reading medium. Kindle Reader desktop software or Kindle Cloud Reader is excellently suited for displaying the book. Devices such as an Apple iPad, Kindle Fire HDX 8.9, or Samsung Galaxy Tab are also optimal for presenting the code examples set forth in this series. Most modern smartphones and Kindle Paperwhite have been tested to render this text well, too. Older media such as Kindle and Kindle DX may not display this text faithfully. Please make sure that you have a high-quality device, Kindle Reader desktop software, or access to Kindle Cloud Reader before purchasing the book.

Recommended Browsers

All examples have been tested in modern browsers. The reader is encouraged to use the latest version of Chrome, Firefox, Safari, or Opera when working through the cases. All screenshots in this book capture the examples as rendered by Google Chrome. The reader will be prompted to use an appropriate browser whenever an experimental feature that is available only in that browser is used to make a given case work.

Development Environment

Any code editor or IDE is sufficient. The book and the examples presented here were written using Cloud9 online IDE (<https://c9.io>).

Code Conventions

All HTML and CSS code, and file and directory names are monospaced.

Book Code

All of the code needed to make each example work is provided in the pages of this book. The necessary supplementary files may be downloaded from functionalcss.com.

Errata

Please submit any errors that you may find by sending an email with the subject line "Errata" to functionalcss@gmail.com.

About the Author

Dr. Dmitriy A. Nesterkin holds a Ph.D. in Information Systems and Quantitative Analysis from the University of Arkansas. He has been working in universities for more than 10 years as an educator, software developer, course designer, researcher, and published author.

Dr. Nesterkin's software development expertise centers on web applications, Monte-Carlo and agent-based simulations, and data visualization. Some of the software that he has developed includes inventory management, financial, personal budgeting, behavior simulation, statistical analysis, laboratory testing, large data visualization, and demand optimization systems.

He writes on such topics as change management, statistical biases and their correction, expertise and expert performance, work team conflicts, educational issues, and programming.

In his spare time, Dr. Nesterkin enjoys tinkering with new technologies, traveling, writing, practicing yoga and mixed martial arts, ballroom dancing, and cooking. He may be reached at functionalcss@gmail.com.

2. Site Template

Introduction

The theme that we will be implementing in this chapter will present a resume as a loose stack of vintage papers lying on top of a leather desk. Each page represents a section of the resume demarcated by a sticky note on the page's left side (see Figure 2.1). Clicking, for example, on the Photos sticky note should take the visitor to that page. The clicked Photos tab should become highlighted, and the other tabs should become less noticeable (see Figure 2.2). The end of each section should also have an automatically generated page number counter (see Figure 2.3).

Figure 2.1: Front page of the resume

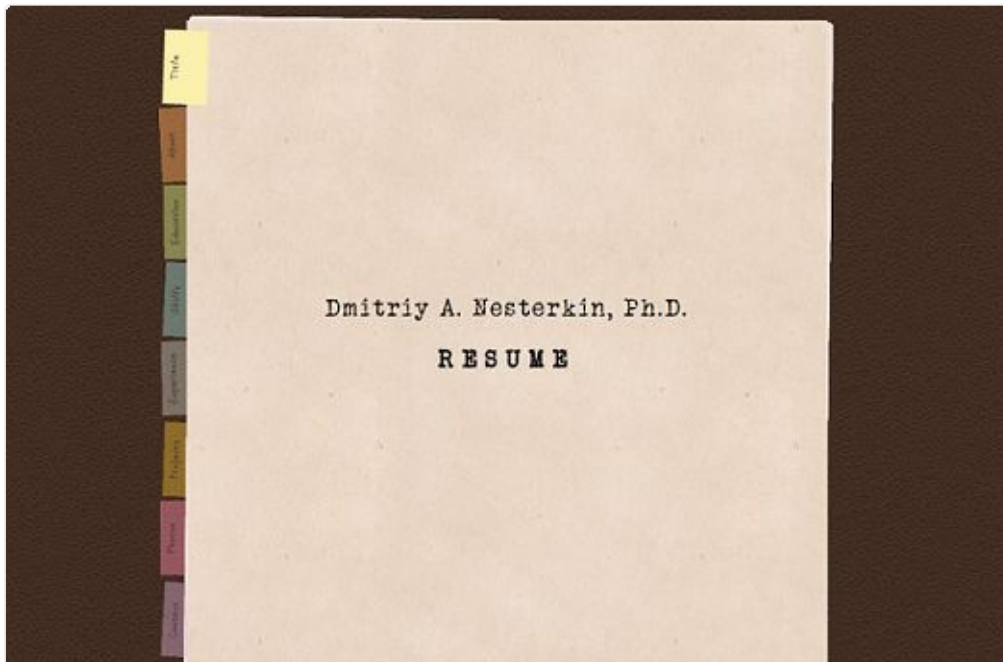


Figure 2.2: Photos section of the resume



Figure 2.3: Page number indicator of the Photos section



Topics Covered

The resume page is assembled using fundamental tags such as `div`, `h1`, `h2`, and `a` and newer HTML5 tags such as `section`, `header`, and `footer`. This chapter again emphasizes the importance of HTML structure in order to enable certain CSS-driven functionalities. New CSS features critical to this tutorial are counters and `:target` pseudo-class. The former is used to dynamically generate page numbers, and the latter is critical to implement a CSS-only click-displayed tabbed content.

HTML Code

In the directory of your choice, create a file named `resume.html`. In the same directory, create three directories: `css`, `fonts`, and `images`. In these directories place the respective `css`, `font`, and `image` files available from functionalcss.com. In the `css` directory, create a file named `resume.css`.

Open `resume.html` in a text editor and type the code shown in Listing 2.1.

Listing 2.1: Add resume site container code `<!DOCTYPE html> <html>
<head> <title>Resume</title> <link rel = "stylesheet" href = "../css/reset.css"
</link rel = "stylesheet" href = "../css/fonts.css" /> <link rel = "stylesheet"
href = "../css/resume.css" /> </head> <body> <div id = "resume"> <section
class = "page" id = "about"> <h1>About</h1> <a href =
"#about">About <div class = "info"> </div> <footer></footer>
</section> </div> </body> </html>`

`#resume` `div` will contain different resume sections. Each `.page` holds a link whose `href` is set to the section's `id`. In the CSS code, a mechanism will be implemented that will display the `.page` once its link is clicked. Each section is visually identified using a heading (`h1`). The content will be placed in the `.info` `div`, and the page number will be automatically generated in the footer.

Immediately after the About section, add the markup for Education, Skills, Experience, and Project `.pages` as displayed in Listing 2.2.

Listing 2.2: Add markup for Education, Skills, Experience, and Project sections
`<section class = "page" id = "education"> <h1>Education</h1> <a href =
"#education">Education <div class = "info"> </div> <footer></footer>
</section> <section class = "page" id = "skills"> <h1>Skills</h1> <a href =
"#skills">Skills <div class = "info"> </div> <footer></footer>
</section> <section class = "page" id = "experience"> <h1>Experience</h1>
Experience <div class = "info"> </div>
<footer></footer> </section> <section class = "page" id = "projects">
<h1>Projects</h1> Projects <div class = "info">
</div> <footer></footer> </section>`

Listing 2.3: Type code for Photos, Contact, and Title .pages

```
<section class =  
"page" id = "photos"> <h1>Photos</h1> <a href = "#photos">Photos</a>  
<footer></footer> </section> <section class = "page" id = "contact">  
<h1>Contact</h1> <a href = "#contact">Contact</a> <footer></footer>  
</section> <section class = "page" id = "title"> <a href = "#title">Title</a>  
<header> <h1>Dmitriy A. Nesterkin, Ph.D.</h1> <h2>Resume</h2>  
</header> </section>
```

When the browser first loads the resume, the Title page should be displayed first, and the other sections should be hidden. To make that happen, using CSS, we will instruct the browser to display the Title page only if none of the previous sections are :targeted by the visitor when the link with a section's id is clicked. To change the state of the Title page, from displayed or hidden, based on the state of its previous siblings, requires for the section to be placed last in the HTML code. This is the case because the only sibling selectors that CSS supports are adjacent and general.

CSS Code

Open `resume.css` and add the code below to specify foundational styles for the document.

Listing 2.4: Specify that sizes set for all elements and pseudo-elements are to include borders and padding `*`, `:before`, `:after` { `-moz-box-sizing`: border-box; `box-sizing`: border-box; }

The universal selector (`*`) will not target pseudo-elements. To find the latter, the `:before` and `:after` keywords are added to the selector string.

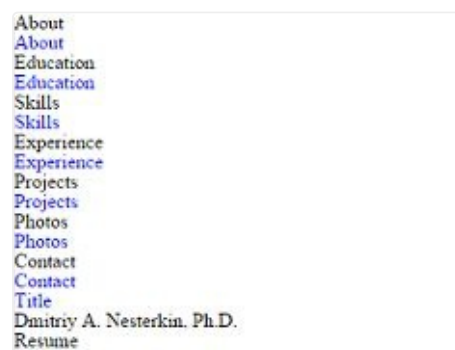
Listing 2.5: Format the links `a` { `text-decoration`: none; } `a:link`, `a:visited`, `a:focus` { `color`: blue; }

The default underlining is removed from all links. The color of the anchors in unclicked (`:link`), `:visited`, or pressed (`:focus`) states is kept blue.

Listing 2.6: Prevent user text selection anywhere in the document `body`, `body *` { `-webkit-user-select`: none; `-moz-user-select`: none; `-ms-user-select`: none; `-o-user-select`: none; `user-select`: none; }

Save `resume.html` and `resume.css` and open `resume.html` in a modern browser. Your preview should look very similar to Figure 2.4.

Figure 2.4: Resume with general styles



Click or press on any link and notice how its color remains blue.

Listing 2.7: Add leather background to the document `body` { `background-color`:

```
#eee; background-image: url("../images/leather.2.jpg"); background-size: 215px 150px; }
```

The background-color is specified as the fallback if the browser is unable to load the background-image. The background image size of 215px by 150px shrinks the original image and makes the leather grain look smaller. By default, the image is repeated horizontally and vertically throughout the body.

Listing 2.8: Make the background-image slightly darker `body:before { content: ""; position: fixed; top: 0; bottom: 0; left: 0; right: 0; background-color: rgba(0, 0, 0, 0.3); z-index: -1; }`

To create better contrast for the pages, the leather background is darkened by placing a fixed semi-transparent screen on the browser viewport. To prevent the screen from blocking user content, the screen is moved behind the content by setting z-index to -1.

Save resume.css and compare your browser to the screenshot in Figure 2.5.

Figure 2.5: Resume with darkened leather background



Listing 2.9: Size and center the .pages .page `{ width: 770px; height: 1020px; position: absolute; margin: 0 auto 20px; top: 0; left: 0; right: 0; padding: 20px 70px 70px 70px; font-family: "Special Elite"; }`

Each page is placed absolutely using the coordinates of the body. It is centered by setting left and right to 0 and by setting left and right margins to auto. The Special Elite typewriter font is used to give the document text a vintage look.

Listing 2.10: Add vintage paper background `.page:before, .page:after { content: ""; position: absolute; top: 20px; left: 0; right: 0; bottom: 0; z-index: -1; }`


```
border: 1px solid #F3E8E2; background-image:  
url("../images/oldpaper.4.jpg"); } .page:after { background-image: none;  
background-color: rgba(255, 255, 255, 0.5); }
```

Two new elements are added to each `.page`. Both are placed at 20px from the top and fill the rest of their container. Each `:before` holds a vintage paper background, and each `:after` is a screen that blurs the background image. Although `:before` and `:after` have the same `z-index` of -1, the `:after` is placed on top, because it appears later. The reader may wonder, why not just place each `.page` 20 pixels down from the top of the body instead of pushing down the pseudo-elements? When an intra-page link is clicked, the top of the page section, to which the link's `href` points, is aligned with the top of the viewport. Switching from section to section by clicking on the respective links would then result in a less optimal user experience, causing the pages to appear to be jumping.

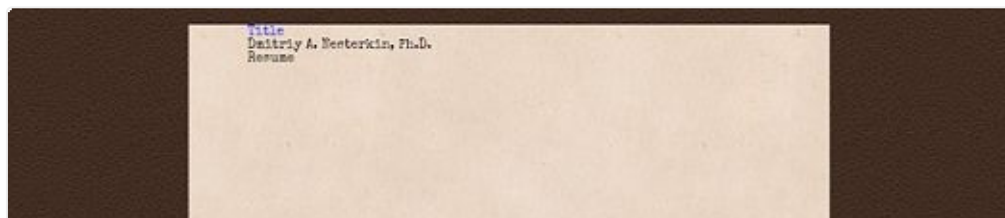
Listing 2.11: Show the Title .page

```
.page:not(:target) ~ .page:last-of-type {  
  z-index: 100;  
}
```

When the intra-page link is clicked, the section to which the `href` of the link points is `:targeted`. The ruleset above will be applied when none of the `.pages` before the last have the `:target`. This style will typically be applied when the document is first loaded.

Save `resume.css`, refresh the browser, and compare the top of the document to Figure 2.6.

Figure 2.6: Resume with Title page on top



CSS Tabs

Each sticky note-looking link will be a tab. As a link is clicked, the browser will display a resume section pointed to by the link. The code below formats and situates the links as side tabs.

Listing 2.12: Format section links as tabs and rotate them about 90 degrees

```
.page > a { position: absolute; text-decoration: none; font: bold 14px/2 "Vag  
Handwritten", Sans-Serif; background-color: #FCF0AD; width: 90px; letter-  
spacing: 1px; height: 53px; text-align: center; color: #525252; left: -30px; top:  
120px; padding: 0 10px; -webkit-transform-origin: top left; transform-origin:  
top left; -webkit-transform: rotate(-88deg); transform: rotate(-88deg); opacity:  
0.5; }
```

Listing 2.13: Visually separate the About page link `.page:nth-of-type(1) > a {`
`top: 215px; -webkit-transform: rotate(-93deg); transform: rotate(-93deg);`
`background-color: #F9A55B; }`

Listing 2.14: Visually separate the Education page link `.page:nth-of-type(2) > a`
`{ top: 310px; transform: rotate(-91deg); background-color: #D0E17D; }`

Listing 2.15: Visually separate the Skills page link `.page:nth-of-type(3) > a {`
`top: 398px; -webkit-transform: rotate(-87deg); transform: rotate(-87deg);`
`background-color: #99C7BC; }`

Listing 2.16: Visually separate the Experience page link `.page:nth-of-type(4) > a`
`{ top: 500px; -webkit-transform: rotate(-92deg); transform: rotate(-92deg);`
`background-color: #BAB7A9; }`

Listing 2.17: Visually separate the Projects page link `.page:nth-of-type(5) > a {`
`top: 590px; -webkit-transform: rotate(-88deg); transform: rotate(-88deg);`
`background-color: #DEAC2F; }`

Listing 2.18: Visually separate the Photos page link `.page:nth-of-type(6) > a {`
`top: 690px; -webkit-transform: rotate(-91deg); transform: rotate(-91deg);`
`background-color: #ED839D; }`

Listing 2.19: Visually separate the Contact page link `.page:nth-of-type(7) > a { top: 785px; -webkit-transform: rotate(-87deg); transform: rotate(-87deg); background-color: #CD9EC0; }`

Save `resume.css` and compare your document to Figure 2.7.

Figure 2.7: Resume with sticky note side tabs



To implement the tabs functionality, add the rulesets that appear below.

Listing 2.20: Display the :targeted .page

```
.page:target {  
  z-index: 100;  
}
```

Listing 2.21: Hide the Title page when one of its previous siblings is :targeted
`.page:target ~ .page:last-of-type { z-index: 1; }`

Save `resume.css`, reload `resume.html`, click on the About link, and click on the Skills link. Your document should look very close to Figure 2.8.

Figure 2.8: Resume with :targeted Skills section



Listing 2.22: Make the link of the `:targeted` `.page` more noticeable `.page:target > a { opacity: 1; }`

Listing 2.23: Make the link of the non-`:targeted` section more noticeable when `:hovered` `.page:not(:target) > a:hover { opacity: 0.85; }`

Listing 2.24: Make the link of the Title section more noticeable when none of the previous sections are `:targeted` `.page:not(:target) ~ .page:last-of-type > a { opacity: 1; }`

Listing 2.25: Make the link of the Title section opaque when one of the previous sections is `:targeted` `.page:target ~ .page:last-of-type > a { opacity: 0.5; }`

Listing 2.26: Make the link of the Title section more noticeable on `:hover` when one of the previous sections is `:targeted` `.page:target ~ .page:last-of-type > a:hover { opacity: 0.85; }`

Selector specificity is used to instruct the browser how to handle different conditions. Save `resume.css`, reload `resume.html`, click on the Experience link, and hover over the Title link. Your browser should look similar to Figure 2.9.

Figure 2.9: Resume template with functioning tabs



To make the pages look loosely stacked, add the rulesets that appear below.

Listing 2.27: Specify the center anchor point from which the .page transformations are to happen `.page { -webkit-transform-origin: center center; transform-origin: center center; }`

Listing 2.28: Shuffle pages 2, 4, 6, and 7

```
.page:nth-of-type(2) {  
  -webkit-transform: rotate(0.6deg);  
  transform: rotate(0.6deg);  
}  
  
.page:nth-of-type(4) {  
  -webkit-transform: rotate(0.8deg);  
  transform: rotate(0.8deg);  
}  
  
.page:nth-of-type(6) {  
  -webkit-transform: rotate(0.7deg);  
  transform: rotate(0.7deg);  
}  
  
.page:nth-of-type(7) {  
  -webkit-transform: rotate(0.44deg);  
  transform: rotate(0.44deg);  
}
```

Save `resume.css` and reload `resume.html`. Make sure your document looks very similar to Figure 2.10.

Figure 2.10: Resume with shuffled pages



CSS Counters

Resumes are dynamic documents and expand as one gains more experience. Instead of hard-coding the page number for each section, we will use the counter features of CSS to generate page numbers dynamically.

Listing 2.29: Position the footers first `.page > footer { position: absolute; bottom: 20px; left: 0; right: 0; text-align: center; font: normal 12px/1 "Special Elite"; }`

Listing 2.30: Initialize the pageNumber counter `body { counter-reset: pageNumber 0; }`

The counter-reset declaration sets the pageNumber counter to 0. The 0 can be omitted, as the browser will initialize the pageNumber counter to that value by default.

Listing 2.31: Dynamically generate page numbers `.page > footer:before { counter-increment: pageNumber 1; content: counter(pageNumber) " of 7"; }`

As the browser selects each footer, the pageNumber counter is incremented by 1. The 1 can be omitted, as the browser will increment pageNumber by that value by default. The content of each :before pseudo-element is set to the value of the pageNumber counter concatenated with a total count string " of 7" pages.

Save resume.css and reload the browser. Click on the Contact tab and scroll down. Compare the bottom portion of the page to Figure 2.11.

Figure 2.11: Contact page with the page number



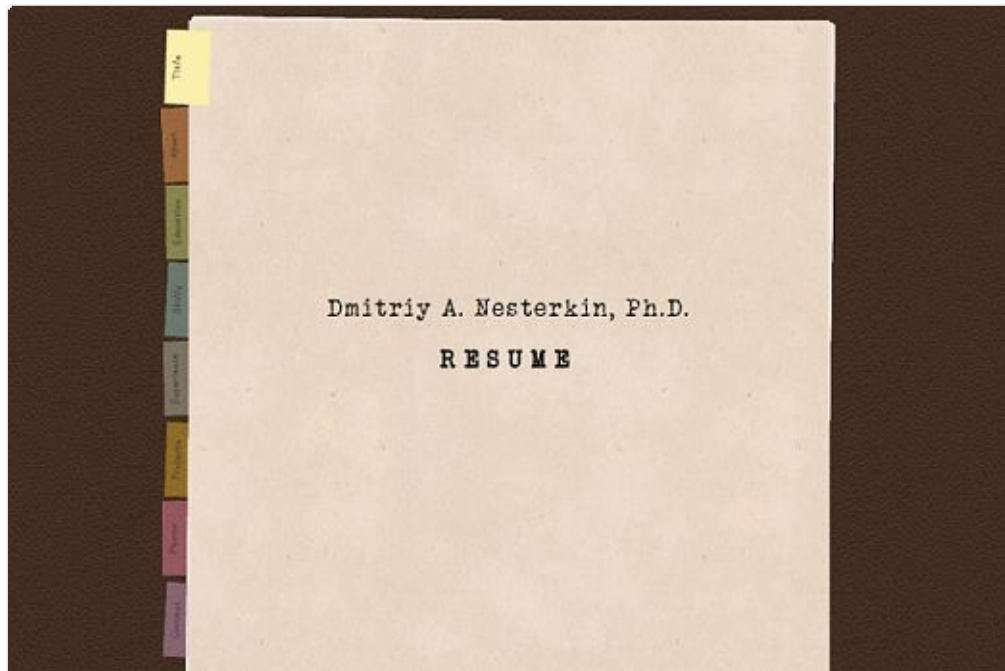
Finishing Touches

To format the Title page and the rest of the headings, add the rulesets that appear below.

Listing 2.32: Format the Title page `#title > header { position: absolute; text-align: center; top: 33%; left: 50%; -webkit-transform: translateX(-50%); transform: translateX(-50%); } #title h1 { margin: 0; font: normal 30px/2 "Veteran Typewriter"; letter-spacing: 3px; white-space: nowrap; } #title h2 { text-transform: uppercase; font: bold 30px/2 "Veteran Typewriter"; letter-spacing: 10px; }`

Save `resume.css` and refresh the browser. Your document's Title page should look similar to Figure 2.12.

Figure 2.12: Formatted Title page



Listing 2.33: Format each section's heading `.page h1 { margin-top: 70px; font: normal 25px/1 "Special Elite"; }`

Save the changes and reload `resume.html`. Click on the Education tab and compare your browser to Figure 2.13.

Figure 2.13: Formatted section heading



Final Remarks

There are other methods to create tabs that do not utilize the `:target` pseudo-class. In this case, however, using anchors and `:target` to show intra-page segments alters the browser's history and is quite desirable and appropriate. This functionality allows the user to bookmark a section of the resume and to navigate the document using the back and forward buttons. In the upcoming Image Gallery chapter, the links and the `:target` mechanism could have been used to display images in the gallery. However, the augmentation of the browser's history in that case was not deemed appropriate, and a different mechanism is used to show the pictures. As always, consideration of user experience is necessary before certain functionality is implemented.

3. Static Content

Introduction

Regular content complements the rest of the document and allows exploration of HTML and CSS features in their native static contexts. The chapter provides an overview of some of the previously covered features, such as floating, and also introduces new CSS features such as columns, filters, and formulae in selectors and new HTML tags such as `details` and `summary`. The latter are used to implement the information-on-demand widget, which via `details` and `summary` tags is natively available from HTML alone. In total, we will add five content sections: About, Education, Skills, Experience, and Projects.

About Page HTML Code

Open `resume.html`, and in the `.info` div of the `#about` section, add the highlighted code as shown in Listing 3.1.

Listing 3.1: Add About page HTML code `<div class = "info"> <img src =
"./images/pictures/dmitriy.jpg" /> <p> Dr. Dmitriy A. Nesterkin holds a Ph.D.
in Information Systems and Quantitative Analysis from the University of
Arkansas. He has been working in universities for over 10 years as an educator,
software developer, course designer, researcher, and published author. </p>
<p> Dr. Nesterkin's software development expertise centers on web
applications, Monte-Carlo and agent-based simulations, and data visualization.
Some of the software that he has developed include inventory management,
financial, personal budgeting, behavior simulation, statistical analysis,
laboratory testing, large data visualization, and demand optimization systems.
</p> <p> He writes on such topics as change management, statistical biases
and their correction, expertise and expert performance, work team conflicts,
educational issues, and programming. </p> <p> In his spare time Dr. Nesterkin
enjoys tinkering with new technologies, traveling, writing, practicing yoga,
ballroom dancing, and cooking. He may be reached at <a href =
"mailto:functionalcss@gmail.com">functionalcss@gmail.com. </p>
</div>`

About Page CSS Code

Add the rulesets that appear below to `resume.css` to format the About section.

Listing 3.2: Push `.info` div down from section title `.page .info { margin-top: 15px; }`

Listing 3.3: Format the picture `#about .info img { float: left; width: 20%; margin: 5px 10px 10px 0; -webkit-filter: sepia(50%); -moz-filter: sepia(50%); filter: sepia(50%); }`

The `sepia(50%)` filter function is used to give the image a more retro look that is in line with the overall resume appearance. `filter` is an experimental property that works in WebKit and Firefox browsers and has functionality to change an element's `brightness()`, `contrast()`, and `opacity()`, to name a few. See a complete online reference for a more up-to-date documentation on the `filter` property.

Listing 3.4: Apply general formatting to all `.info` paragraphs `.page .info p { font: normal 13px/1.5 "Special Elite"; }`

Listing 3.5: Resize `#about` paragraphs to fit to the right of the image `#about .info p { width: calc(80% - 10px); float: right; }`

Listing 3.6: Move down every `#about` paragraph after the first paragraph `#about .info p + p { margin-top: 12px; }`

Listing 3.7: Tell the browser to add a dotted border to the hovered e-mail link `#about .info a:hover { border-bottom: 1px dotted blue; }`

Save `resume.html` and `resume.css`. Open the former in a modern browser and hover over the e-mail link. Your document should look very close to Figure 3.1.

Figure 3.1: About section with hovered e-mail link

About



Dr. Dmitry A. Besterkin holds a Ph.D. in Information Systems and Quantitative Analysis from the University of Arkansas. He has been working in universities for over 10 years as an educator, software developer, course designer, researcher, and published author.

Dr. Besterkin's software development expertise centers on web applications, Monte-Carlo and agent-based simulations, and data visualization. Some of the software that he has developed include inventory management, financial, personal budgeting, behavior simulation, statistical analysis, laboratory testing, large data visualization, and demand optimization systems.

He writes on such topics as change management, statistical biases and their correction, expertise and expert performance, work team conflicts, educational issues, and programming.

In his spare time Dr. Besterkin enjoys tinkering with new technologies, traveling, writing, practicing yoga, ballroom dancing, and cooking. He may be reached at functionalcs@gmail.com.

Education Page HTML Code

In resume.html in the .info div of the #education section add the highlighted code as shown in Listing 3.8.

Listing 3.8: Add .degree information `<div class = "info"> <div class = "degree"> <p>Ph.D. in Information Systems/Quantitative Analysis</p> <p>University of Arkansas (2004 - 2009)</p> <details> <summary>details</summary> <p> One of my favorite quotes is by Frank Herbert from his sci-fi novel - Dune: “Muad'Dib learned rapidly because his first training was in how to learn. And the first lesson of all was the basic trust that he could learn.” Earning a Ph.D. was principal in learning to adopt cognitively. There were no multiple choice, true/false exams. No power point slides to memorize. During the coursework years my peers and I spent 40 hours a week reading. It was a world in which one learned to define problems and to think about them conceptually. The five years that I spent at the University of Arkansas were the renaissance and the major highlight of my life.</p> <p> Cumulative GPA: 3.86 </p> </details> </div> </div>`

The details tag contains supplementary information that the user can view on demand. By default, only the summary content is visible first. When the user clicks on the summary, the rest of the content is presented.

Listing 3.9: Add another .degree after the last `<div class = "degree"> <p> M.S. in Management Information Systems</p> <p>The University of Memphis (2002 - 2004)</p> <details> <summary>details</summary> <p> Studying for a Master of Science went hand in hand with developing one of the largest systems that I have ever delivered: Copier Billings Management System (CBMS™). This enterprise-wide application brought together various departments and handled inventory management, financial accounting, fraud detection, and detailed reporting. The first version of the system went live in April of 2003 and was used by The University of Memphis for over 10 years. </p> <p> Cumulative GPA: 3.94 </p> </details> </div>`

Listing 3.10: Add another .degree after the last `<div class = "degree">` `<p>`
`B.B.A` in Management Information Systems`</p>` `<p>`The
University of Memphis `(1997 - 2002)</p>` `<details>`
`<summary>details</summary>` `<p>` Earning my degree coincided with strong
practical experience that ranged from building servers, configuring and
networking them, designing applications, and then implementing these
applications using such platforms as HTML/CSS, PHP, MySQL, and Apache.
The latter I mastered on my own as universities were slow to introduce
curriculum that taught application development using LAMP. On the side, I also
wrote web applications using C language. During these experimentations I
learned a lot about web applications and programming in general, and even had
to extend one of the cgi libraries by writing an http cookie parser. `</p>`
`<p>`Major GPA: `3.9</p>` `<p>`Cumulative GPA:
`3.449</p>` `</details>` `</div>`

Listing 3.11: Add the last .degree

```
<div class = "degree">
  <p><strong>B.B.A</strong> in Marketing/Management</p>
  <p>The University of Memphis <span>(1997 - 2002)</span></p>
  <details>
    <summary>details</summary>
    <p>
      I double majored in Marketing/Management to obtain a broader perspective
      of the issues that organizations faced.
    </p>
    <p>Major GPA: <strong>3.5</strong></p>
    <p>Cumulative GPA: <strong>3.449</strong></p>
  </details>
</div>
```


Education Page CSS Code

Add the rulesets that appear below to `resume.css` to format the Education section.

Listing 3.12: Move down every `.info` div after the first div

```
.page .info div + div {  
    margin-top: 13px;  
}
```

Listing 3.13: Increase paragraph font size `#education .info p` { font-size: 14px; }

Listing 3.14: Bolden all strong content `#education .info strong` { font-weight: bold; }

Listing 3.15: Push down all `.degree` paragraphs after the first one `#education .degree p + p` { margin-top: 5px; }

Listing 3.16: Move down details

```
#education .degree details {  
    margin-top: 5px;  
}
```

Listing 3.17: Format details paragraphs `#education .degree details p` { font-size: 12px; margin-left: 14px; }

Listing 3.18: Format summary content `#education .degree summary` { font: normal 14px/1.5 "Special Elite"; outline: 0; cursor: pointer; font-style: italic; }

Listing 3.19: Remove default summary dropdown indicator `summary::-webkit-details-marker` { display: none }

The `details-marker` pseudo-element is available only in WebKit browsers. For this exercise, please use Google Chrome when previewing the page.

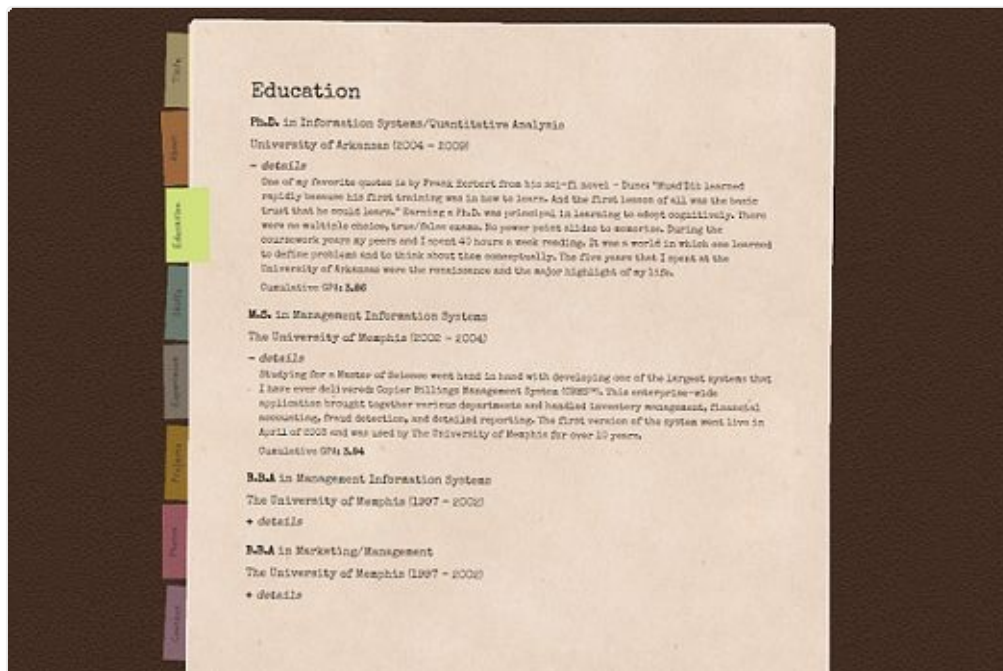
Listing 3.20: Add the custom plus sign dropdown indicator `summary:before` { content: "\f067"; font: 10px "Font Awesome"; display: inline-block; margin-right: 5px; opacity: 0.8; }

Listing 3.21: Tell the browser to change the dropdown indicator to a minus sign when details is open

```
details[open] summary:before {  
    content: "\f068";  
}
```

Save `resume.css`, `resume.html`, refresh the latter in Google Chrome, and click on the Education tab. Click on the first two details indicators. The document should look very similar to Figure 3.2.

Figure 3.2: Education page with first two open details sections



Skills Page HTML Code

In resume.html in the .info div of the #skills section, add the highlighted code as shown in Listing 3.22.

Listing 3.22: Add .skillset information

```
<div class = "info"> <div class =  
"skillset" id = "technical"> <h2>Technical</h2> <ul> <li>Presentation</li>  
<li>HTML</li> <li>CSS</li> <li>SASS</li> <li>LESS</li>  
<li>Compass</li> <li>Design</li> </ul> <ul> <li>Programming</li>  
<li>JavaScript</li> <li>Java</li> <li>PHP</li> <li>C</li> <li>Visual  
Basic</li> <li>C#</li> <li>Python</li> </ul> </div> </div>
```

Listing 3.23: Add the following two lists of skills to the #technical div

```
<ul>  
  <li>Libraries</li>  
  <li>Backbone</li>  
  <li>Angular</li>  
  <li>jQuery</li>  
  <li>Express</li>  
  <li>Socket.IO</li>  
  <li>Bootstrap</li>  
  <li>Zend Framework</li>  
</ul>  
<ul>  
  <li>Databases</li>  
  <li>MySQL</li>  
  <li>MongoDB</li>  
  <li>Access</li>  
  <li>SQL Server</li>  
</ul>
```

Listing 3.24: Add the last two lists of skills to the #technical div

```
<ul>  
  <li>Environments</li>  
  <li>Node.js</li>  
  <li>.NET</li>  
</ul>  
<ul>  
  <li>Utilities</li>  
  <li>Git</li>  
  <li>Bower</li>  
  <li>Gulp</li>  
</ul>
```

Listing 3.25: Add the #quantitative div immediately after the #technical div

```

<div class = "skillset" id = "quantitative">
  <h2>Quantitative</h2>
  <ul>
    <li>General</li>
    <li>Visualization</li>
    <li>Statistics</li>
    <li>Modeling</li>
    <li>Optimization</li>
    <li>Monte-Carlo Simulation</li>
    <li>Agent-Based Simulation</li>
  </ul>
  <ul>
    <li>Software</li>
    <li>R</li>
    <li>SPSS</li>
    <li>SAS</li>
    <li>Excel</li>
  </ul>
</div>

```

Listing 3.26: Add the #communication div immediately after the #quantitative div

```

<div class = "skillset" id = "communication">
  <h2>Communication</h2>
  <ul>
    <li>General</li>
    <li>Writing</li>
    <li>Speaking</li>
    <li>Presenting</li>
    <li>Publishing</li>
    <li>Teaching</li>
  </ul>
  <ul>
    <li>Languages</li>
    <li>English</li>
    <li>Russian</li>
  </ul>
</div>

```

Skills Page CSS Code

Add the rulesets below to `resume.css` to format the Skills section.

Listing 3.27: Apply general formatting to all h2s in a .page

```
.page h2 {  
    font-weight: bold;  
    font-size: 13px;  
    margin-bottom: 10px;  
}
```

Listing 3.28: Push .info in #skills further down #skills .info { margin-top: 20px; }

Listing 3.29: Format .skillset headings #skills .info h2 { font-size: 16px; margin-bottom: 15px; letter-spacing: 1px; }

Listing 3.30: Move down every .skillset after the first .skillset

```
#skills .skillset + .skillset {  
    margin-top: 20px;  
}
```

Listing 3.31: Size each list of skills to one third of its container and stack the lists next to each other #skills .skillset ul { float: left; width: 33%; }

Listing 3.32: Clear floats #skills .skillset { overflow: hidden; }

The `overflow: hidden` declaration is one of the most common ways to keep floats away from other containers. For an explanation on how this method works, please search the web. There are many sources that address this topic.

Listing 3.33: Move down every 4th and up list within its container #skills .skillset ul:nth-of-type(n + 4) { margin-top: 15px; }

Listing 3.34: Set list items' font size #skills .skillset ul > li { font-size: 13px; }

Listing 3.35: Move down every list item after the second one #skills .skillset ul > li:first-of-type ~ li + li { margin-top: 5px; }

Listing 3.36: Format every first list item, which is a list heading #skills .skillset

```
ul > li:first-of-type { text-transform: uppercase; margin-bottom: 7px; font-size: 14px; display: inline-block; border-bottom: 1px dotted #000; }
```

Save `resume.html`, `resume.css`, open the former in a modern browser, and click on the Skills tab. Compare your document to Figure 3.3.

Figure 3.3: Skills page



Experience Page HTML Code

In `resume.html`, add the following code to the `#experience .info` section as displayed in Listing 3.37.

Listing 3.37: Add the first two professional .roles `<div class = "role">`
`<h2>Web Application Developer (1999 ‐ Present)</h2>`
`<p> A bulk of my experience centers on building web-based systems for`
`universities and some companies. The main theme that unites my software is`
`optimization of internal processes. Many of the programs were built because I`
`saw an opportunity to create a better work structure and took the initiative and`
`the responsibility that comes with it. </p> </div> <div class = "role">`
`<h2>User Interface Designer (2013 ‐ Present)</h2>`
`<p> Studying psychology and human-computer interaction, coupled with in-`
`depth knowledge of HTML and CSS, allowed me to venture into the territory of`
`GUI design and build elements that allow for intuitive and laborless user`
`experiences. </p> </div>`

Listing 3.38: Add next two .roles `<div class = "role"> <h2>Database`
`Developer (1999 ‐ Present)</h2> <p> Every single`
`application that I have developed required some form of data storage. Most of`
`the solutions utilized relational databases that had anywhere between 2 to 99`
`tables. Some programs functioned optimally with flat file storage or a`
`combination of both. </p> </div> <div class = "role"> <h2>Simulation`
`Designer (2004 ‐ Present)</h2> <p> As a researcher, I`
`am interested in complex phenomena that do not yield to simple research`
`methodologies. Simulations represent a “summary” of a real`
`world inside a computer memory, allowing investigators greater degrees of`
`freedom to glimpse the essence of the studied issues at hand. </p> </div>`

Listing 3.39: Add another two .roles `<div class = "role"> <h2>University`
`Instructor/Professor (2005 ‐ 2014)</h2> <p> I have`
`taught at the University of Arkansas, Towson University, Coastal Carolina`
`University, and West Texas A&M University. Teaching and designing`
`courses in programming, systems development, statistics, quantitative analysis,`
`and other areas, provided many opportunities to deepen my expertise in these`

subjects. </p> </div> <div class = "role"> <h2>Researcher (2004 ‐ Present)</h2> <p> A process of research involves conceptual and empirical investigation of the issues at hand. As a scientist, I studied team conflict, individual expertise and expert performance, organizational change and its management, and statistical biases. Becoming a researcher was invaluable in learning how to think about immediate unknowns from a set of existing first principles. </p> </div>

Listing 3.40: Add the last two professional experiences <div class = "role"> <h2>Instructional Designer (2010 ‐ 2014)</h2> <p> Most of my curriculum development centers on quantitative methods and programming. I have developed courses for both public and private universities. One of the programs - Statistics for Software Engineers - is used in one of the largest online universities. </p> </div> <div class = "role"> <h2>Writer (2009 ‐ Present)</h2> <p> Writing is akin to teaching, only harder. This especially applies to technical and instructional writing. Encapsulating a dynamic process of programming using a very static media forces one to present cases as simply, yet as fully, as possible. I have learned that a really good way to master a technology is to create a practical guide for it. </p> </div>

Experience Page CSS Code

Add the rulesets below to `resume.css` to present the Experience information in two columns.

Listing 3.41: Stack first four `.roles` vertically in a column `#experience .role:nth-of-type(-n + 4) { float: left; clear: left; width: 48%; margin-right: 2%; }`

The formula `-n + 4` is what selects the first four `.roles`. The `n` is a counter for the elements matched by the selector, which starts at 0 for the first matched element. The `n` is then incremented by 1 for each matched element.

The `float` moves the `.roles` to the left of their container. Setting `clear` to `left` ensures that the floats' left edge avoids contact with any of their siblings' edges. The latter is what makes the four experiences stack vertically.

Listing 3.42: Change the width of the rest of the `.roles` `#experience .role { display: inline-block; width: 50%; }`

This ruleset will be applied only the last four `.roles`, because of the selector specificity. The previous ruleset has a more specific selector, which is why this ruleset's `display` and `width` declarations will not override the previous ones.

Setting the width to 50% ensures that content stays within its `.role` div. Declaring `display` as `inline-block` moves the last four experiences to the right of the first column.

Listing 3.43: Adjust the margin on the fifth `.role` to align it with the first `.role`

```
#experience .role:nth-of-type(5) {  
    margin-top: 0;  
}
```

Save `resume.html` and `resume.css`, reload the former in your browser, and click on the Experience tab. Your resume should look very similar to Figure 3.4.

Figure 3.4: Experience page

Experience

Web Application Developer (1999 - Present)

A bulk of my experience centers on building web-based systems for universities and some companies. The main theme that unites my software is optimization of internal processes. Many of the programs were built because I saw an opportunity to create a better work structure and took the initiative and the responsibility that comes with it.

User Interface Designer (2015 - Present)

Studying psychology and human-computer interaction, coupled with in-depth knowledge of HTML and CSS, allowed me to venture into the territory of GUI design and build elements that allow for intuitive and laborless user experiences.

Database Developer (1999 - Present)

Every single application that I have developed required some form of data storage. Most of the solutions utilized relational databases that had anywhere between 8 to 99 tables. Some programs functioned optimally with flat file storage or a combination of both.

Simulation Designer (2004 - Present)

As a researcher, I am interested in complex phenomena that do not yield to simple research methodologies. Simulations represent a "summary" of a real world inside a computer memory, allowing investigators greater degrees of freedom to glimpse the

University Instructor/Professor (2006 - 2014)

I have taught at the University of Arkansas, Towson University, Coastal Carolina University, and West Texas A&M University. Teaching and designing courses in programming, systems development, statistics, quantitative analysis, and other areas, provided many opportunities to deepen my expertise in these subjects.

Researcher (2004 - Present)

A process of research involves conceptual and empirical investigation of the issues at hand. As a scientist, I studied team conflict, individual expertise and expert performance, organizational change and its management, and statistical biases. Becoming a researcher was invaluable in learning how to think about immediate unknowns from a set of existing first principles.

Instructional Designer (2010 - 2014)

Most of my curriculum development centers on quantitative methods and programming. I have developed courses for both public and private universities. One of the programs - Statistics for Software Engineers - is used in one of the largest online universities.

Writer (2010 - Present)

Writing is akin to teaching, only harder. This especially applies to technical and instructional writing. Incorporating a dynamic process of programming using a very static media forces one to present cases as

Projects Page HTML Code

In `resume.html`, add the code as shown in Listing 3.44 to the `#projects .info` section.

Listing 3.44: Add the first three `.projects` `<div class = "project">`
`<h2>Demand Analyzer</h2>` `<p>` The software aggregated data from multiple Excel spreadsheets into a relational database and provided a wizard tool via which a manager could assess whether client's product demand requirements could be satisfied with the company's present supply (Built with PHP, MySQL, HTML, and CSS). `</p>` `</div>` `<div class = "project">` `<h2>RFID Static Visualization Testing System</h2>` `<p>` The system provided facilities to allow for real-time and selective partitioning of multi-dimensional datasets and for their visual representation. The system has been used to develop product classification rules and heuristics (Built with PHP, MySQL, jpGraph, and PDF). `</p>` `</div>` `<div class = "project">` `<h2>Copier Billings Management System (CMBS™)</h2>` `<p>` CBMS™ handled data on 100s of machines and kept track of about \$1,000,000 in revenue. The system numbered over 30,000 lines of code and was a huge step up from the paper-and-pencil arrangement that was used to track copier operations and finances (Built with HTML, CSS, PHP, and MySQL). `</p>` `</div>`

Listing 3.45: Add next three `.projects` `<div class = "project">` `<h2>Sentiment Analyzer</h2>` `<p>` The system piped sentences into an algorithm that evaluated the emotionality of the text. (Built with JavaScript and Node.js). `</p>` `</div>` `<div class = "project">` `<h2>Social Desirability Bias Simulator</h2>` `<p>` Social desirability bias occurs when survey participants answer questions in a socially acceptable way rather than sharing their true perceptions. The software was constructed to examine how social desirability, under diverse conditions, affected various statistics (Built with JavaScript and Node.js). `</p>` `</div>` `<div class = "project">` `<h2>Budget Tracker</h2>` `<p>` Web-based personal budgeter was written to store expenses and incomes and to generate financial reports (Built with HTML, CSS, C, and MySQL). `</p>` `</div>`

Listing 3.46: Add the last three `.projects` `<div class = "project">` `<h2>Event Scheduler</h2>` `<p>` The event scheduler was implemented as a single-page

web software. The program was written as a technology demonstrator to illustrate some of the latest trends in web application development. (Built with HTML, CSS, JavaScript, jQuery, Backbone, Node, MongoDB, Express, and custom UI widgets).

RFID Project Management System

The goals driving the system's implementation were three-fold: 1) centralize information pertaining to RFID testing via database implementation, 2) stabilize the testing process by embedding it into the software, and 3) create functionality to mine the data stored in the database. RFID Project Management System stands at 75,000 lines of code and was implemented on top of a database that has 99 tables. (Built with HTML, CSS, PHP, and MySQL).

Non-Sampling Bias Simulator

Non-sampling errors are the most pervasive and deleterious biases that affect empirical results. It is very difficult to study these phenomena directly, because the necessary data is next to impossible to obtain. Instead, a simulation software was written that faithfully mimicked non-sampling biases, allowing in-depth understanding of the errors' operation and enabling the derivation of methods to prevent the biases altogether. (Built with Java).

Projects Page CSS Code

Add the ruleset below to `resume.css` to present the Projects information in two columns.

Listing 3.47: Display `#projects .info` content in two columns `#projects .info { -webkit-column-count: 2; -moz-column-count: 2; column-count: 2; }`

Using CSS3, developers can explicitly instruct browsers to present content in column format. Please consult a reputable online reference to learn more about other CSS columns properties such as `column-gap`, `column-fill`, and `column-width`.

Save both files and reload `resume.html` in a browser. Compare your document to Figure 3.5.

Figure 3.5: Projects page



Final Remarks

This chapter was a diversion from functional consideration of CSS. We still managed to consider new CSS features such as columns, filters, and selector formulae. The tutorial also employed a functional information-on-demand HTML feature, which can be readily implemented using `details` and `summary` tags and formatted with specialty CSS. In the next two chapters, we switch back to the functional usage of CSS and finish this book by building an Image Gallery and Data Form with Input Validation components.

4. Image Gallery

Introduction

In this tutorial, we will build a simple photo gallery that will allow the user to display a larger version of an image by clicking on the image's thumbnail. The main image should be encased in a photograph-like frame with an appropriate bottom-situated caption. The clicked thumbnail should be the most visible, and any hovered non-clicked thumbnail should be highlighted (see the figure below).

Figure 4.1: Clicked second thumbnail and hovered fourth thumbnail



Topics Covered

Only one thumbnail can be selected at a time, displaying only one main image. Clicking on a thumbnail should also deselect any previously selected image. To enable this type of functionality the `radio` inputs paired with their `labels` are utilized. Generic paragraphs (`ps`) hold caption text. A `div` is employed to hold a large version of the selected image.

The case introduces border image CSS properties for the first time. The chapter also reviews such CSS features as selector specificity, positioning, pseudo-classes, pseudo-elements, background images, filters, element sizing, and text effects.

HTML Code

Open `resume.html`; in the `#photos` section, right after the anchor tag, add the highlighted code as shown in Listing 4.1.

Listing 4.1: Add #gallery div to #photos section

```
<section class = "page" id = "photos">
  <h1>Photos</h1>
  <a href = "#photos">Photos</a>
  <div id = "gallery">
    <input type = "radio" id = "image1" name = "imageSelector" />
    <input type = "radio" id = "image2" name = "imageSelector" />
    <input type = "radio" id = "image3" name = "imageSelector" />
    <input type = "radio" id = "image4" name = "imageSelector" />
    <input type = "radio" id = "image5" name = "imageSelector" />

    <label for = "image1" class = "thumbnail"></label>
    <label for = "image2" class = "thumbnail"></label>
    <label for = "image3" class = "thumbnail"></label>
    <label for = "image4" class = "thumbnail"></label>
    <label for = "image5" class = "thumbnail"></label>

  <div class = "viewer"></div>

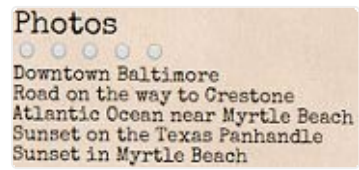
  <p>Downtown Baltimore</p>
  <p>Road on the way to Crestone</p>
  <p>Atlantic Ocean near Myrtle Beach</p>
  <p>Sunset on the Texas Panhandle</p>
  <p>Sunset in Myrtle Beach</p>
</div>
<footer></footer>
</section>
```

The key component to giving the gallery an image switching functionality is the pairing of the `radio` inputs and their labels. The inputs will be hidden; however, clicking a label will check its respective radio button and uncheck other radio inputs. The `:checked` state of a radio button will determine which `.thumbnail` to highlight, which image to display in a bigger format in the `.viewer`, and which paragraph to display as the caption for the selected image. The only sibling selectors that CSS supports are adjacent and general. This is why listing `radio` inputs first in the `#gallery` is necessary to connect the changes of the `radio` inputs to the type of formatting that should be applied to their siblings downstream.

Save `resume.html`, open it in a modern browser, and click on the Photos tab.

Your page should look very similar to Figure 4.2.

Figure 4.2: Unformatted image gallery



CSS Code

Open `resume.css` and add the rulesets that appear below.

Listing 4.2: Hide radio inputs `#gallery input { display: none; }`

Listing 4.3: Size and explicitly position the #gallery

```
#gallery {  
  width: 630px;  
  height: 391px;  
  margin-top: 15px;  
  position: relative;  
}
```

Border Images

The `border-image` properties are used for the first time in this series. These facilitate the creation of borders from custom images and can enhance the overall appearance of an application.

Listing 4.4: Position the image `.viewer` and add the image border `#gallery`

```
.viewer { position: absolute; top: 0; bottom: 0; left: 0; right: 0; border: solid #ccc; border-width: 10px 10px 30px 10px; border-image-source: url("../images/photo_border.png"); border-image-slice: 21 21 97 21; border-image-repeat: stretch; }
```

The `border-image-source` specifies the image to be used for the border. The `border-image-slice` instructs the browser how the source image is to be cropped. The four numbers indicate where the slicing cuts are to be made. The first slicing line is placed horizontally 21 pixels from the top edge of the image. The second slicing line is situated vertically 21 pixels from the right edge of the image. The third line is cut horizontally 97 pixels from the bottom edge of the source photo. The fourth and last line is placed vertically 21 pixels from the left edge of the image. Thus, the image is cut into 9 chunks. Each corner chunk is placed at its respective border corner (e.g., upper left piece is placed in the upper left corner of the border). The chunks are scaled down to fit the top, right, bottom, and left border-widths of 10, 10, 30, and 10, respectively. The outer image chunks that are between corner chunks are placed in their respective border places and are made to stretch their allotted space via the `border-image-repeat` property. Please consult a comprehensive online CSS reference to learn more about border image properties and their values.

Save `resume.css` and refresh the Photos page of the document. Your gallery should look very much like Figure 4.3.

Figure 4.3: `#gallery`'s `.viewer` with image border

Photos

Unknown location
oad on the way to Crestone
atlantic Ocean near Myrtle Beach
unset on the Texas Panhandle
unset in Myrtle Beach

Back to the CSS Code

Listing 4.5: Place and size first main image in the .viewer and give it a retro appearance

```
#gallery .viewer:before { content: ""; position: absolute; width: 100%; height: 100%; background: url(../images/pictures/baltimore_downtown.JPG) center center/cover no-repeat; -webkit-filter: sepia(50%); -moz-filter: sepia(50%); filter: sepia(50%); }
```

Listing 4.6: Format all .thumbnails

```
#gallery .thumbnail { position: absolute; bottom: -72px; width: 100px; height: 62px; left: 10px; background: url(../images/pictures/baltimore_downtown.JPG) center center/cover no-repeat; opacity: 0.6; cursor: pointer; -webkit-filter: sepia(50%); -moz-filter: sepia(50%); filter: sepia(50%); }
```

Listing 4.7: Set a unique image for and reposition the second .thumbnail

```
#gallery .thumbnail:nth-of-type(2) { left: 137.5px; background-image: url(../images/pictures/crestone_road.JPG); }
```

Listing 4.8: Set a unique image for and reposition the third .thumbnail

```
#gallery .thumbnail:nth-of-type(3) { left: 264px; background-image: url(../images/pictures/reading_near_atlantic.JPG); }
```

Listing 4.9: Set a unique image for and reposition the fourth .thumbnail

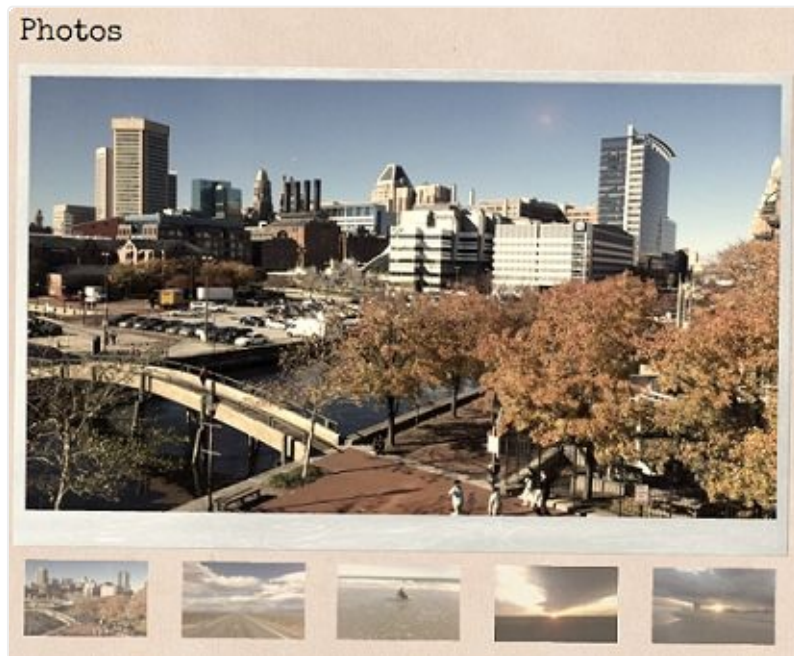
```
#gallery .thumbnail:nth-of-type(4) { left: 391.5px; background-image: url(../images/pictures/texas_sunset.JPG); }
```

Listing 4.10: Set a unique image for and reposition the fifth .thumbnail

```
#gallery .thumbnail:nth-of-type(5) { left: auto; right: 10px; background-image: url(../images/pictures/mb_sunset.JPG); }
```

Save the css file and reload the Photos section of resume.html. Compare your image gallery to Figure 4.4.

Figure 4.4: Image Gallery with `.thumbnails`



Listing 4.11: Highlight hovered `.thumbnail`

```
#gallery .thumbnail:hover {  
  opacity: 0.8;  
}
```

Listing 4.12: Make the first `.thumbnail` most noticeable if no image selection has been made `#gallery input[type = "radio"]:not(:checked) ~ .thumbnail:nth-of-type(1) { opacity: 1; }`

Listing 4.13: Dim the first `.thumbnail` when another `.thumbnail` is clicked `#gallery input[type = "radio"]:not(:first-of-type):checked ~ .thumbnail:nth-of-type(1) { opacity: 0.6; }`

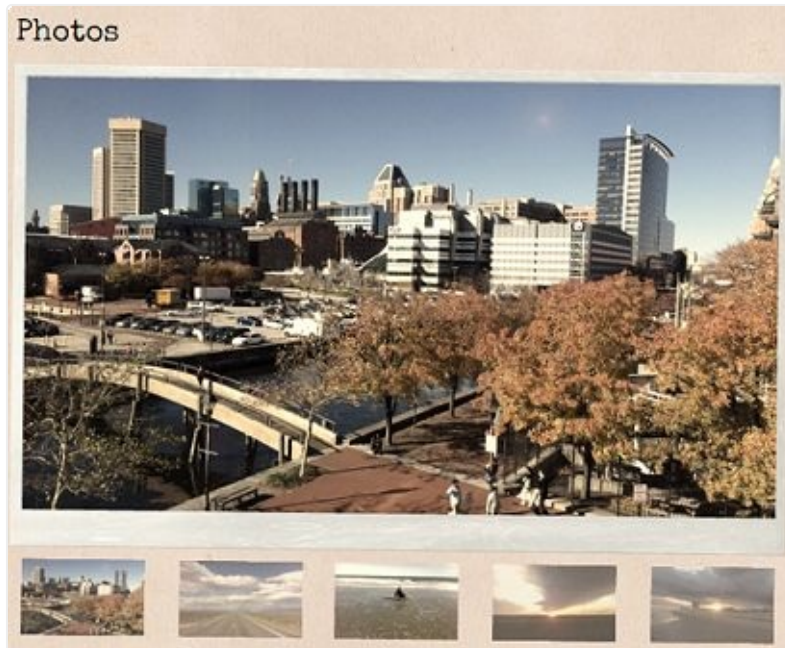
Listing 4.14: Highlight hovered first `.thumbnail` when another `.thumbnail` is selected `#gallery input[type = "radio"]:not(:first-of-type):checked ~ .thumbnail:nth-of-type(1):hover { opacity: 0.8; }`

Listing 4.15: Make any clicked `.thumbnail` most noticeable `#gallery input[type = "radio"]:nth-of-type(1):checked ~ .thumbnail:nth-of-type(1), #gallery input[type = "radio"]:nth-of-type(2):checked ~ .thumbnail:nth-of-type(2), #gallery input[type = "radio"]:nth-of-type(3):checked ~ .thumbnail:nth-of-type(3), #gallery input[type = "radio"]:nth-of-type(4):checked ~ .thumbnail:nth-`

`of-type(4), #gallery input[type = "radio"]:nth-of-type(5):checked ~
.thumbnail:nth-of-type(5) { opacity: 1; }`

Save `resume.css`, refresh the html file, click on the Photos tab, click on the third thumbnail, and hover over the first thumbnail. Your preview should look very similar to Figure 4.5.

Figure 4.5: Gallery with clicked third `.thumbnail` and hovered first `.thumbnail`



Listing 4.16: Display an image in the `.viewer` appropriate for each clicked `.thumbnail`

```
#gallery input[type = "radio"]:nth-of-type(1):checked ~ .viewer:before {  
    background-image: url(../images/pictures/baltimore_downtown.JPG)  
}  
  
#gallery input[type = "radio"]:nth-of-type(2):checked ~ .viewer:before {  
    background-image: url(../images/pictures/crestone_road.JPG);  
}  
  
#gallery input[type = "radio"]:nth-of-type(3):checked ~ .viewer:before {  
    background-image: url(../images/pictures/reading_near_atlantic.JPG);  
}  
  
#gallery input[type = "radio"]:nth-of-type(4):checked ~ .viewer:before {  
    background-image: url(../images/pictures/texas_sunset.JPG);  
}  
  
#gallery input[type = "radio"]:nth-of-type(5):checked ~ .viewer:before {  
    background-image: url(../images/pictures/mb_sunset.JPG);  
}
```


Save your code, reload the Photos page of the resume, and click on the second thumbnail. The main image should now be a larger version of the second thumbnail image (see Figure 4.6). Click on different thumbnails and notice how the image in the `.viewer` appropriately changes.

Figure 4.6: Second `.thumbnail`'s image displayed as a main image



The only remaining components left unformatted are image caption paragraphs. Add the rulesets below to `resume.css` to complete the image gallery.

Listing 4.17: Format each paragraph and position it at the bottom of the `.viewer`

```
#gallery p {  
  position: absolute;  
  left: 10px;  
  right: 10px;  
  bottom: 0;  
  z-index: 2;  
  display: none;  
  color: #484848;  
  font-size: 11px;  
  line-height: 30px;  
  font-weight: normal;  
  font-style: italic;  
  letter-spacing: 1px;  
  text-shadow: 0 0 1px #fff;  
}
```

Listing 4.18: Display the first paragraph when none of the thumbnails have been

```
clicked on #gallery input[type = "radio"]:not(:checked) ~ p:nth-of-type(1) {  
display: block; }
```

Listing 4.19: Hide the first paragraph when any thumbnail, other than the first, has been selected

```
#gallery input[type = "radio"]:not(:first-of-type):checked ~  
p:nth-of-type(1) { display: none; }
```

Listing 4.20: Display a paragraph appropriate for each clicked .thumbnail

```
#gallery input[type = "radio"]:nth-of-type(1):checked ~ p:nth-of-type(1),  
#gallery input[type = "radio"]:nth-of-type(2):checked ~ p:nth-of-type(2),  
#gallery input[type = "radio"]:nth-of-type(3):checked ~ p:nth-of-type(3),  
#gallery input[type = "radio"]:nth-of-type(4):checked ~ p:nth-of-type(4),  
#gallery input[type = "radio"]:nth-of-type(5):checked ~ p:nth-of-type(5) {  
display: block;  
}
```

Listing 4.21: Align the text of the fourth and fifth paragraphs to the right

```
#gallery p:nth-of-type(4), #gallery p:nth-of-type(5) { text-align: right; }
```

The last ruleset accomplishes a visual nuance. When a user clicks on one of the thumbnails on the right, the text for the displayed larger image also appears on the right, making it less laborious for the user to read the caption.

Save the code, refresh the gallery page, and click on the fifth thumbnail. Your preview should look like Figure 4.7.

Figure 4.7: Displayed fifth image with a caption on the bottom right

Photos



Sunset in Myrtle Beach



Final Remarks

CSS can be used to build some interesting functional features. This image gallery example can even be further extended to incorporate a paginated set of thumbnails with left and right chevrons that can be used to switch from one page of thumbnails to the next. I leave it up to the reader to implement that type of functionality or potentially look for it in a future volume of the Functional CSS series.

This tutorial also highlights one serious limitation of the current CSS specifications - lack of generalizability. Notice that relationships between each thumbnail and its main image and its paragraph have to be hard-coded. This makes adding or removing gallery elements somewhat tedious. Perhaps future implementations of CSS may give us a more generalizable and flexible syntax. At this time, this type of flexibility can be added only via inclusion of appropriate JavaScript code.

5. Data Form with Input Validation

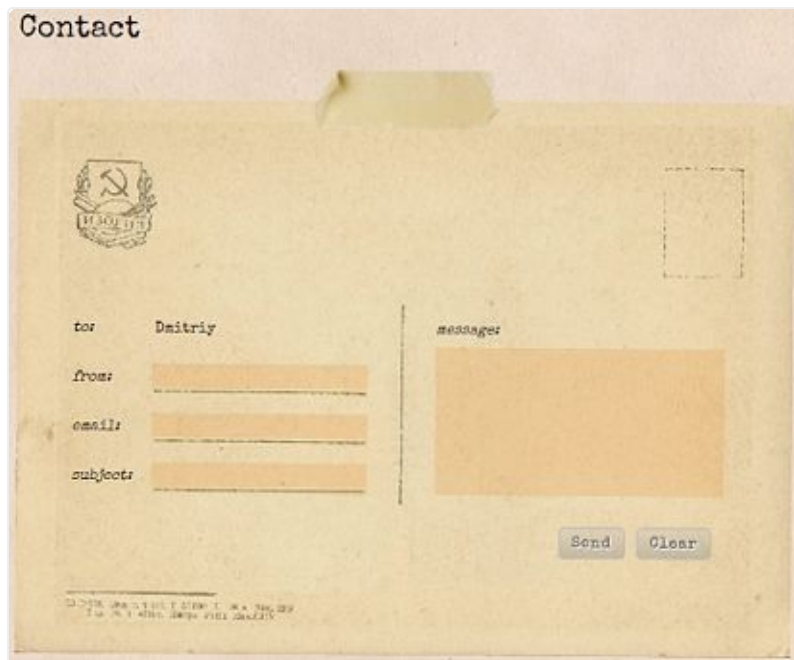
Introduction

Data form and its children (e.g., input, button) are among the most important HTML elements for they enable web-based exchange of data. Paramount to the process of data capture is data validation. Typical approaches for ensuring data quality involve browser-and/or server-based data checking. With the expansion of HTML and CSS specifications via the introduction of HTML5 and CSS3, developers now have non-programmatic tools to make sure that user-submitted data meet the desired formats.

In this chapter, we will build a simple contact form whose required fields will have a faded orange background. Upon the initial loading or when all provided data are invalid, both Send and Clear buttons should be disabled (see Figure 5.1). Once a valid data is provided to one of the fields, the field's background color should change to white, and the Clear button should become enabled in case a user would like to clear field values with one click (see Figure 5.2). Once all fields contain valid data, the Send button should become enabled, and hovering over any enabled button should be made more noticeable by darkening a hovered button's background (see Figure 5.3).

Figure 5.1: Contact form without data

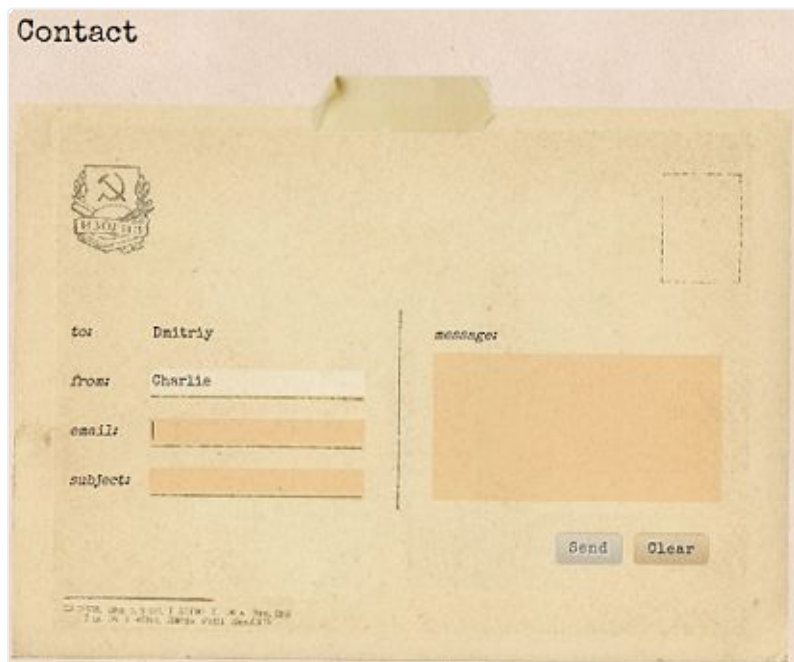
Contact



The image shows a contact form titled "Contact" on a light beige background. In the top left corner is a circular logo featuring a hammer and sickle. Below the logo are four input fields labeled "to:", "from:", "email:", and "subject:". The "to:" field contains the text "Dmitriy". The "from:", "email:", and "subject:" fields are empty. To the right of these fields is a large rectangular area labeled "message:". At the bottom right of the form are two buttons: "Send" and "Clear". Both buttons are disabled, appearing as light gray with a thin border. At the bottom left, there is small, faint text: "© 2008 Web Design Studio, Inc. All Rights Reserved. This is a demo page. Please do not use it for real data."

Figure 5.2: Contact form with valid data for one input

Contact



The image shows the same contact form as in Figure 5.2, but with more data entered. The "to:" field still contains "Dmitriy". The "from:" field now contains "Charlie". The "email:" and "subject:" fields are still empty. The "message:" area is still empty. The "Send" and "Clear" buttons are now enabled, appearing as light gray with a thicker border. The "Send" button is highlighted with a darker gray background, indicating it is the active or hovered button. The logo and footer text are the same as in Figure 5.2.

Figure 5.3: Contact form with all valid inputs, enabled buttons, and hovered Send button

Topics Covered

The `label`, `span`, `input`, `textarea`, and `button` tags compose the Contact form. To achieve the data validation functionality that we are seeking, `type` and required attributes of the `input` and `textarea` elements are utilized.

The vast majority of CSS employed for this tutorial has been covered previously. New material includes `:valid` and `:invalid` pseudo-classes, a custom `::-webkit-scrollbar` pseudo-element, and `resize` property.

HTML Code

Open `resume.html` and in the `#contact` section, right after the anchor tag, add the highlighted code as shown in Listing 5.1.

Listing 5.1: Add #postcard form to the #contact section

```
<section class = "page" id = "contact">
  <h1>Contact</h1>
  <a href = "#contact">Contact</a>
  <form id = "postcard">
    <label>to:</label>
    <span>Dmitriy</span>
    <label>from:</label>
    <input type = "text" required/>
    <label>email:</label>
    <input type = "email" required/>
    <label>subject:</label>
    <input type = "text" required/>
    <label>message:</label>
    <textarea required></textarea>
    <button type = "submit">Send</button>
    <button type = "reset">Clear</button>
  </form>
  <footer></footer>
</section>
```

The form consists of one text element, four data elements, and two buttons. All data elements (inputs and textarea) have required mono-attributes. The latter tells the browser that the provided data must match the type of the data element. text inputs and textarea can receive any type of data. The required email input expects a valid email address. Browser checking of email addresses is somewhat loose. A more rigorous email validation can be accomplished by adding a pattern attribute with the appropriate Regular Expression value to the desired input. The reader is welcome to consult the most up-to-date HTML reference that shows how to set a pattern for an input.

All elements within the form are siblings. This makes it easier to connect changes occurring to data inputs to the formatting changes of the buttons.

Save `resume.html`, open it in a modern browser, click on the Contact tab, and click the Send button. Your page should look like Figure 5.4.

Figure 5.4: Unformatted Contact form with browser-generated error message

Contact

to: Dmitriy from: email: subject:

mess:

 Please fill out this field.

CSS Code

Open `resume.css` and add the rulesets that appear below.

Listing 5.2: Size and position #postcard form

```
#postcard {  
    width: 630px;  
    height: 450px;  
    position: absolute;  
    top: 160px;  
}
```

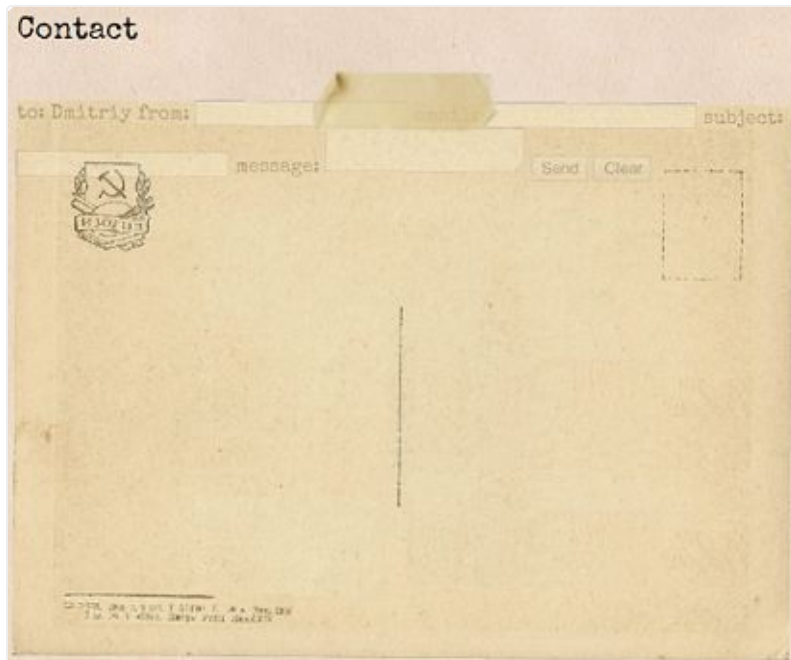
Absolute positioning is used for all elements to achieve the necessary placement precision.

Listing 5.3: Place old Soviet postcard background #postcard:before { content: ""; position: absolute; top: 0; left: 0; width: 100%; height: 100%; background: url(../images/soviet_postcard.5.png) no-repeat top left/630px 450px; opacity: 0.75; }

Listing 5.4: Place sticky tape on top of the postcard #postcard:after { content: ""; position: absolute; width: 150px; height: 50px; background: url(../images/tape.png) no-repeat top left/150px 50px; top: -25px; left: 50%; margin-left: -75px; opacity: 0.75; }

Save the css code, reload the Contact page, and compare the latter to Figure 5.5.

Figure 5.5: Contact form with old Soviet background and sticky tape



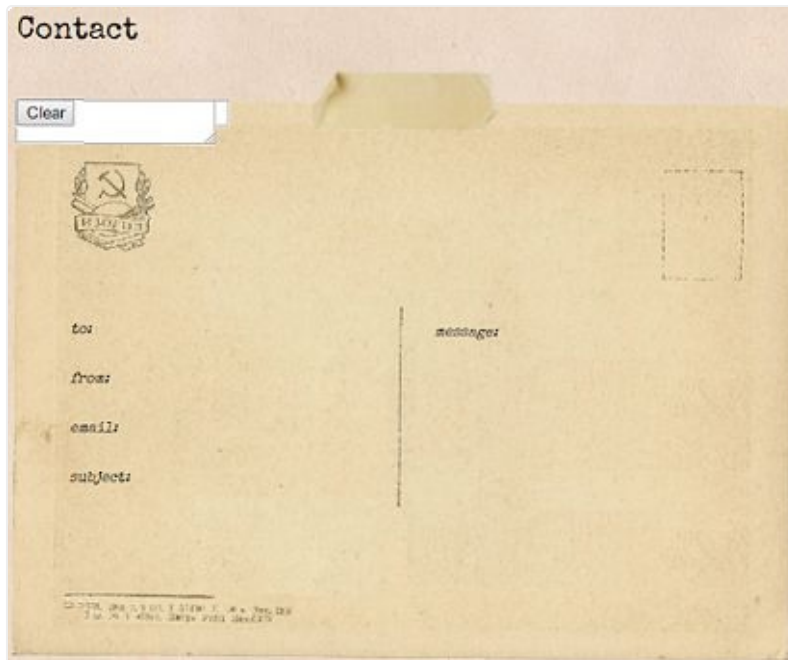
Listing 5.5: Absolutely position all `#postcard` children `#postcard > * { position: absolute; }`

Listing 5.6: Format and position labels `#postcard > label { top: 180px; left: 45px; z-index: 2; font-style: italic; font-size: 12px; width: 55px; }`

Listing 5.7: Uniquely position the last four labels `#postcard > label:nth-of-type(2) { top: 220px; }` `#postcard > label:nth-of-type(3) { top: 260px; }` `#postcard > label:nth-of-type(4) { top: 300px; }` `#postcard > label:nth-of-type(5) { left: 340px; }`

Save `resume.css` and reload Contact page. Your form should look very similar to Figure 5.6.

Figure 5.6: Contact form with positioned labels



Listing 5.8: Format and position `span` and `inputs` `#postcard > span, #postcard > input` { `position: absolute`; `top: 180px`; `font-size: 12px`; `left: 110px`; `width: 175px`; `z-index: 1`; } `#postcard > span` { `padding-left: 2px`; }

Listing 5.9: Include `input`-specific formatting `#postcard > input` { `outline: 0`; `border: 0`; `background-color: rgba(255, 255, 255, 0.25)`; `padding: 0 2px`; `font-family: "Special Elite"`; `line-height: 20px`; }

Listing 5.10: Visually distinguish each `input`

```
#postcard > input:first-of-type {
  top: 215px;
}

#postcard > input:nth-of-type(2) {
  top: 255px;
}

#postcard > input:nth-of-type(3) {
  top: 295px;
}
```

Listing 5.11: Format and position the `textarea`

```
#postcard > textarea {
  position: absolute;
  width: 235px;
  height: 120px;
  top: 200px;
  left: 340px;
  border: 0;
```

```

background-color: rgba(255, 255, 255, 0.25);
outline: 0;
resize: none;
font: normal 12px "Special Elite";
overflow: hidden;
}

#postcard > textarea::-webkit-scrollbar {
display: none;
}

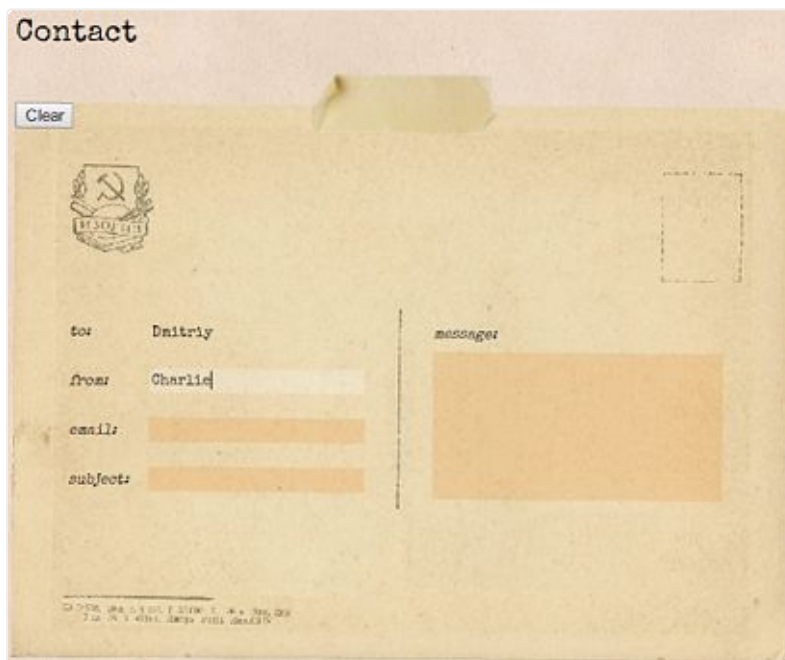
```

Typically, the size of textarea can be changed by dragging the element's lower right corner. User resizing is likely to affect the form's design, and this feature is disabled via `resize: none` declaration.

Listing 5.12: Set faded orange background color for all `:invalid` data elements
`#postcard > input:invalid, #postcard > textarea:invalid { box-shadow: none; background-color: rgba(240, 124, 0, 0.15); }`

Save the css code, reload the Contact form, type something in the first input, and compare the page to Figure 5.7.

Figure 5.7: Contact form with filled-in first input



Listing 5.13: Add underline to each input

```

#postcard > label:nth-last-of-type(2):before,
#postcard > label:nth-last-of-type(3):before,
#postcard > label:nth-last-of-type(4):before {
content: "";

```

```

position: absolute;
height: 5px;
width: 175px;
background: url(../images/underline.1.png) 0 -2px/cover no-repeat;
bottom: -9px;
right: -185px;
}

```

The `:before` and `:after` pseudo-elements are not available on inputs. Therefore, the underline elements are added to labels and then placed under their respective input.

Listing 5.14: Add a slightly different underline image for the last two inputs

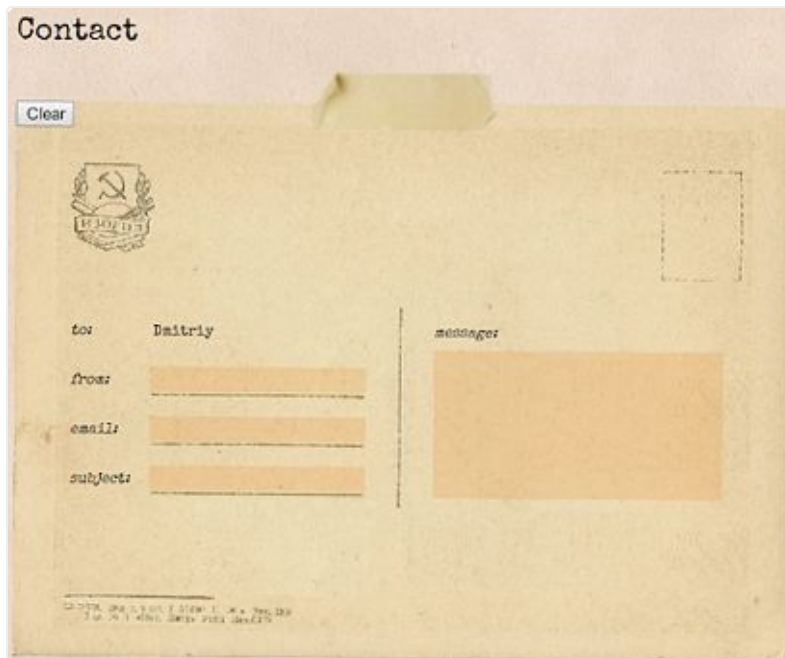
```

#postcard > label:nth-last-of-type(2):before { background-image:
url(../images/underline.2.png); } #postcard > label:nth-last-of-type(4):before {
background-image: url(../images/underline.3.png); }

```

Save the css code, reload the Contact page, and compare the document to Figure 5.8.

Figure 5.8: Contact form with underlined inputs



Selector Specificity

To make the data form user-friendly and intuitive, the Submit button should be clickable only when all of the required inputs are correctly filled. The Clear button should be enabled when at least one of the inputs is completed.

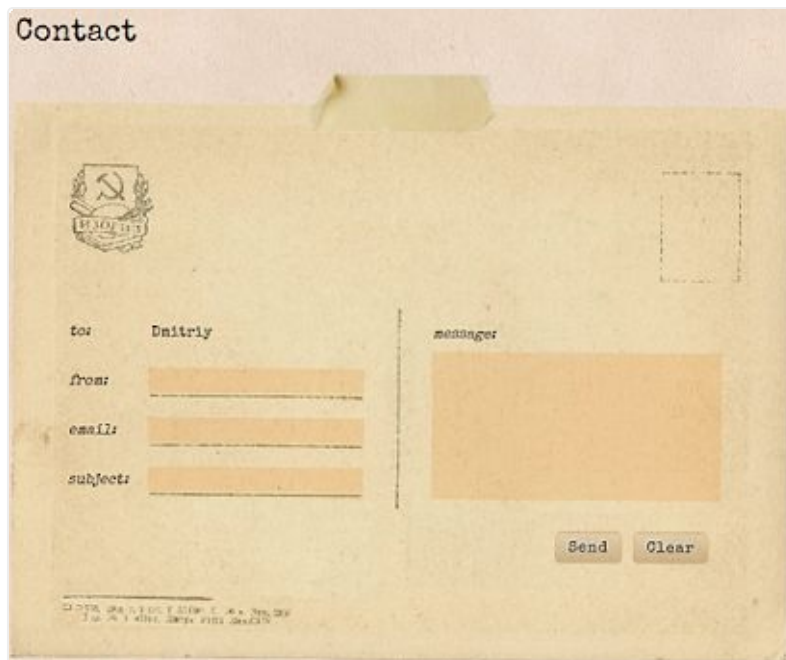
Implementation of the above-listed conditionals requires the careful creation of selectors with precise specificities. To catch all of the nuances of this exercise, save and preview the data form code after adding each block of instructions listed below.

Listing 5.15: Position and format the buttons `#postcard button { position: absolute; top: 345px; right: 65px; font: bold 12px/2 "Special Elite"; letter-spacing: 1px; color: rgba(20, 20, 20, 0.8); text-shadow: -1px -1px 1px #eee; padding: 0 10px; outline: 0; border: 1px solid rgba(129, 75, 0, 0.1); border-radius: 5px; background-color: rgba(255, 255, 255, 0.2); background-image: linear-gradient(to bottom, rgba(129, 75, 0, 0.1), rgba(129, 75, 0, 0.3)); cursor: pointer; pointer-events: all; }`

Listing 5.16: Move the submit button to the left `#postcard > button[type = "submit"] { right: 135px; }`

Save the css code, reload the Contact form, and compare the page to Figure 5.9.

Figure 5.9: Positioned and formatted buttons

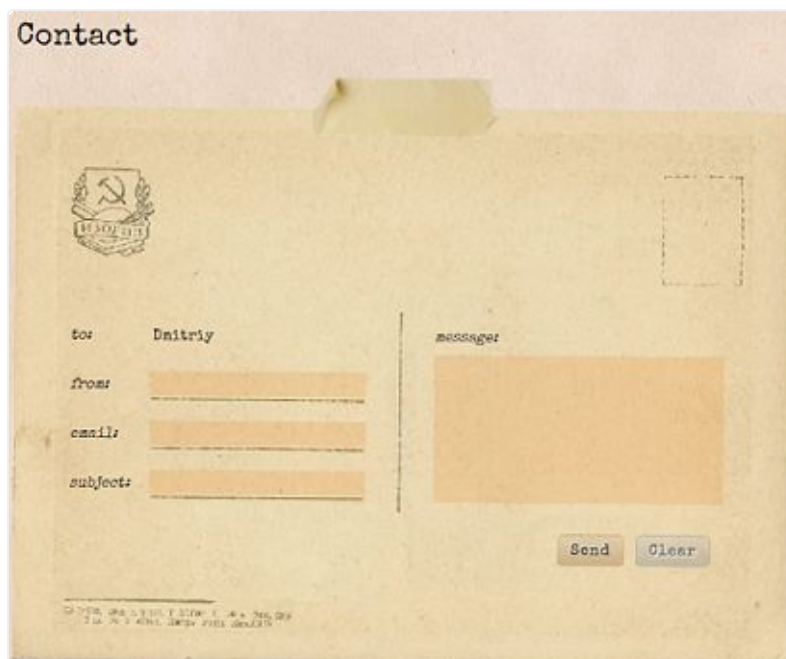


Listing 5.17: Ensure that the `reset` button is disabled upon initial load

```
#postcard > button[type = "reset"] { pointer-events: none; border-color: #bbb;
color: rgba(50, 50, 50, 0.7); text-shadow: 1px 1px 1px #fff; background-image:
linear-gradient(to bottom, rgba(100, 100, 100, 0.1), rgba(100, 100, 100, 0.3)); }
```

Your data form preview should look similar to Figure 5.10.

Figure 5.10: Initially loaded data form



To enable the reset button when at least one of the inputs is correct, add the highlighted portion, shown below, to the code that was listed in Figure 5.15.

Listing 5.15: Enable the reset button when at least one input is valid

```
#postcard button, #postcard > input:valid ~ button[type = "reset"], #postcard >
textarea:valid ~ button[type = "reset"] { position: absolute; top: 345px; right:
65px; font: bold 12px/2 "Special Elite"; letter-spacing: 1px; color: rgba(20, 20,
20, 0.8); text-shadow: -1px -1px 1px #eee; padding: 0 10px; outline: 0; border:
1px solid rgba(129, 75, 0, 0.1); border-radius: 5px; background-color:
rgba(255, 255, 255, 0.2); background-image: linear-gradient(to bottom,
rgba(129, 75, 0, 0.1), rgba(129, 75, 0, 0.3)); cursor: pointer; pointer-events: all;
}
```

Save the css code, refresh the data form, type "Robert" in the from field, and notice that the Clear button has become enabled (see Figure 5.11).

Figure 5.11: Data form with one valid input and enabled Clear button

Clicking the Clear button should erase all form data, thereby disabling the Clear button itself.

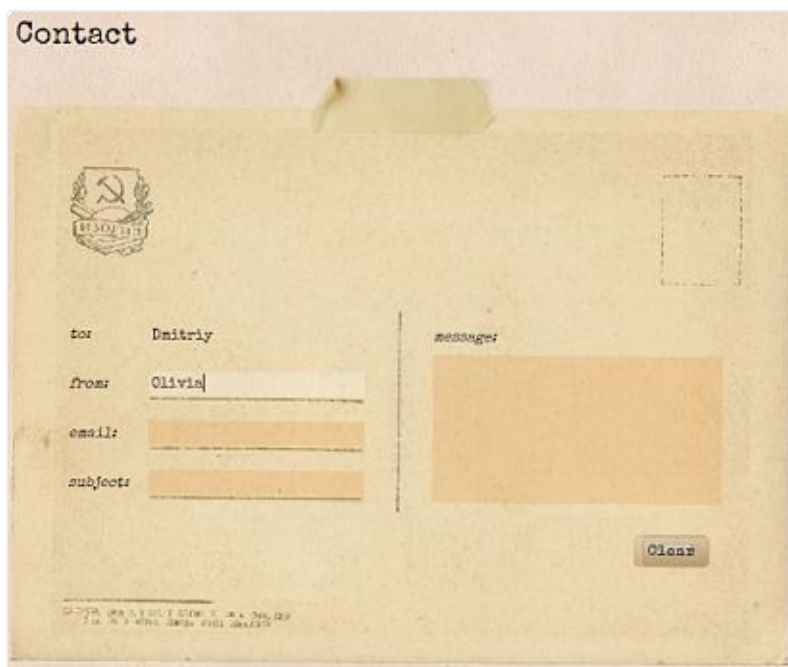
To enable the Submit button, add the highlighted code, shown below, to the previous code listing.

Listing 5.15: Enable the submit button when at least one data field is valid

```
#postcard button, #postcard > input:valid ~ button[type = "reset"], #postcard >
textarea:valid ~ button[type = "reset"], #postcard > input:valid ~ button[type =
"submit"], #postcard > textarea:valid ~ button[type = "submit"] { position:
absolute; top: 345px; right: 65px; font: bold 12px/2 "Special Elite"; letter-
spacing: 1px; color: rgba(20, 20, 20, 0.8); text-shadow: -1px -1px 1px #eee;
padding: 0 10px; outline: 0; border: 1px solid rgba(129, 75, 0, 0.1); border-
radius: 5px; background-color: rgba(255, 255, 255, 0.2); background-image:
linear-gradient(to bottom, rgba(129, 75, 0, 0.1), rgba(129, 75, 0, 0.3)); cursor:
pointer; pointer-events: all; }
```

Save the css code, refresh the data form, type "Olivia" in the "from" field, and notice how the buttons become lumped together (see Figure 5.12).

Figure 5.12: Repositioned submit button



To ensure that the buttons stay separate, add more specific highlighted selectors, shown below, to the code already presented in Listing 5.16.

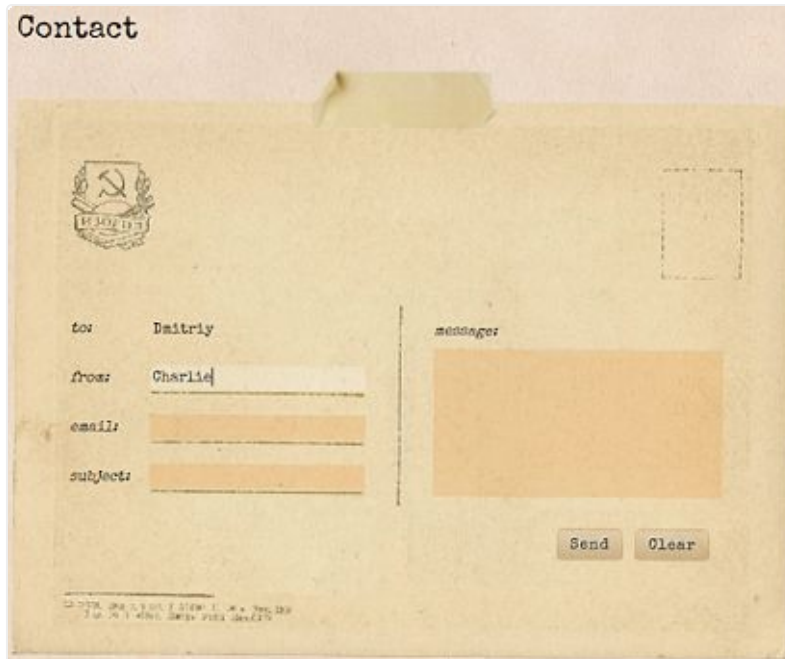
Listing 5.16: Separate two buttons

```
#postcard > button[type = "submit"],
#postcard > input:valid ~ button[type = "submit"], #postcard > textarea:valid ~
button[type = "submit"] { right: 135px; }
```

Save resume.css, reload the data form, type "Charlie" in the from field, and

notice how the two buttons stay separate from each other (see Figure 5.13).

Figure 5.13: Data form with one valid entry and separate Send and Clear buttons



The above functionality does not meet our full specification, because the Send button should become activated only when all required fields are completed. To implement the full specification, add more specific highlighted selectors, shown below, to the code already displayed in Listing 5.17.

Listing 5.17: Show Send button only when all required fields are completed

```
#postcard > button[type = "reset"], #postcard > input:invalid ~ button[type =  
"submit"], #postcard > textarea:invalid ~ button[type = "submit"] { pointer-  
events: none; border-color: #bbb; color: rgba(50, 50, 50, 0.7); text-shadow: 1px  
1px 1px #fff; background-image: linear-gradient(to bottom, rgba(100, 100, 100,  
0.1), rgba(100, 100, 100, 0.3)); }
```

Save resume.css, reload the form, type anything in the "from," "subject," and "message" fields, and notice that the Send button is still disabled (see Figure 5.14).

Figure 5.14: Form with one incomplete required field and accordingly disabled Send button

Contact

to: Dmitriy

from: Homer

email:

subject: code

message: Howdy,

Send Clear

To complete the exercise, add the code below to instruct the browser to contrast an enabled button when it is hovered.

Listing 5.18: Allow highlighting of enabled hovered buttons

```
#postcard >
input:valid ~ button:hover, #postcard > textarea:valid ~ button:hover {
background-image: linear-gradient(to bottom, rgba(129, 75, 0, 0.2), rgba(129,
75, 0, 0.4)); }
```

Save the css code, reload the form, fill the entire form, and hover over the Send button. Compare your page to Figure 5.15.

Figure 5.15: Hovered Send button

Contact



to: Dmitriy

from: Toby

email: tporterfield@towson.edu

subject: research

message:

Hi Dmitriy,

Would you please send me the latest draft of the paper?

Cheers,

Toby

Send

Clear

© 2008, USA & ALL RIGHTS RESERVED. TOWSON UNIVERSITY
TOWSON, MD 21204-2000

Final Remarks

The most optimal web software takes advantage of all the necessary HTML, CSS, and JavaScript features jointly. Relying on HTML and CSS as the solid and continuously maturing technologies that they are, developers are likely to ease the pressure that may be placed on their JavaScript code base. The result is a leaner software that is easier to maintain and extend and more efficient to run. This is especially the case when developers are building internal applications that make it easier to require users to interact with the software via a modern browser.

6. Conclusion

In this and the previous two volumes of the series, we have left almost no CSS stone unturned. This text deviated from the other two books and grounded functional CSS coverage in a context that sought to integrate interactive components with static content in a cohesive and realistic resume web site. By completing this volume, you have contributed significantly to your already strong foundation from which you can confidently venture towards those few new HTML and CSS features and applications that we have not yet explored. The best way to absorb the new information is to merge it with your existing knowledge by writing relevant software. Your personal or professional circumstances may provide these types of opportunities to delve into the presentational languages further.

You are also welcome to continue this journey with me in the next book of the Functional CSS series. Using interactive cases, we will learn new material and solidify what we have acquired already.

Thank you for reading this book.

Dmitriy A. Nesterkin, Ph.D.
Ft. Lauderdale, FL
May 17, 2015 at 5:44 p.m.