THE EXPERT'S VOICE® IN WEB DEVELOPMENT

# Pro HTML5 and CSS3 Design Patterns

*APPLY DESIGN PATTERNS TO INCREASE CREATIVITY AND PRODUCTIVITY IN YOUR WEB DESIGNS*

Michael Bowers, Dionysios Synodinos, and Victor Sumner

Apress®

*For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.*


Apress•

# Contents at a Glance

# Introduction

This is a solutions book for styling HTML5 with CSS3. It contains more than 350 design patterns you can put to use right away. Each design pattern is modular and customizable, and you can combine patterns to create an unlimited number of designs.

Each design pattern has been thoroughly tested and proven to work in all major web browsers including Chrome, Firefox, Internet Explorer, Opera, and Safari. All the content in this book is usable and practical. You won't waste time reading about things that don't work! With this book, you will no longer have to use hacks, tricks, endless testing, and constant tweaking in multiple browsers to get something to work.

Using a design pattern is as easy as copying and pasting it into your code and tweaking a few values. You will immediately see which values you can modify and how they affect the result so you can create the exact style and layout you want—without worrying whether it will work.

This is more than a cookbook. It systematically covers several usable features of CSS and combines these features with HTML to create reusable patterns. Each pattern has an intuitive name to make it easy to find, remember, and talk about. Accessibility and best practices are carefully engineered into each design pattern, example, and source code.

You can read straight through the book, use it as a reference, and use it to find solutions. Each example includes a screenshot and all relevant HTML and CSS code so you can easily see how each design pattern works. The explanation for each design pattern is included alongside, so you can easily study the example while you read about how it works.

Design patterns are organized by topic, and all usable CSS rules are covered in depth and in context like no other book. All design patterns are accessible and follow best practices, making this book a worthwhile read from cover to cover, as well as an excellent reference to keep by your side while you are designing and coding.

This book unleashes your productivity and creativity in web design and development. Design patterns are like Legos—you can combine them in countless ways to create any design. They are like tools in a toolbox, and this book arms you with hundreds of tools you can whip out to solve problems quickly and reliably. Instead of hacking away at a solution, this book shows you how to create designs *predictably*—by combining *predictable patterns*.

## Audience

This book is written for those who have some familiarity with CSS and HTML. It is for newcomers who have previously read an introductory book on CSS and HTML. It is for designers and developers who tried CSS at one time and gave up because it never seemed to work right. It is for professionals who want to take their CSS skills to a higher level. It is for all who want to create designs quickly without hacking around until they find something that works in all browsers.

We assume that you know the basics of *coding* CSS and HTML. If you work exclusively in WYSIWYG designers like Dreamweaver or FrontPage and never look at HTML or CSS code, you may find the code in this book overwhelming.

If you like to learn by example, like to see how code works, and have some familiarity with CSS and HTML, you will love this book.

Some design patterns use JavaScript. To fully understand them, you need to understand the basics of JavaScript, but you do not need to know JavaScript to use these patterns. Most importantly, you do not need to know anything about JavaScript to understand and use the remaining 340+ design patterns because they have nothing to do with JavaScript!

# Innovations

This book contains several innovative concepts, terms, and approaches. These are not new or radical: the technology is already built into the major browsers, the concepts are implied in the CSS specification, and the terms are commonly used. What makes them innovative is how we define and use them to show what can be done with CSS and HTML. In other words, they are innovative because they simplify learning, understanding, and using CSS and HTML. These ideas change how you think about CSS and HTML, and that makes all the difference. Furthermore, many of the design patterns in the book are innovative because they document combinations of properties and elements to solve difficult problems like never before.

## Six Box Models

One innovation in the book is the idea that CSS has *six* box models instead of one. CSS officially has one box model that defines a common set of properties and behaviors. A single box model is a very useful concept, but it is oversimplified. Over the years, we learned the hard way that box model properties work differently depending on the type of box.

This is one reason why so many people struggle with CSS. The box model seems simple, yet when one uses a box model property, such as **`width`**, it works only some of the time or may work differently than expected. For example, the **`width`** property sets the interior width of a block box, but on table boxes it sets the outer width of the border, and on inline boxes it does absolutely nothing.

Rather than treating different behaviors as an exception to one very complicated box model, we define six simple box models that specify the behavior for each type of box. Chapter 4 presents the six box models, which are inline, inline-block, block, table, absolute, and float. Since you always know which of these six box models you are using, you always know how each box model property will behave.

Furthermore, each box model defines its own way that it flows or is positioned. For example, inline boxes flow horizontally and wrap across lines. Block boxes flow vertically. Tables flow their cells in columns and rows. Floats flow horizontally, wrap below other floats, and push inline boxes and tables out of the way. Absolute and fixed boxes do not flow; instead, they are removed from the flow and are positioned relative to their closest positioned ancestor.

## Box Model Extents

Another innovation in the book is the concept that there are three ways a box can be dimensioned: it can be sized, shrinkwrapped, or stretched (see Chapter 5). Each type of box requires different combinations of properties and property values for it to be sized, shrinkwrapped, or stretched. Various design patterns in Chapters 5 through 9 show how this is done. These three terms are not official CSS terms, but they are implied in the CSS specification in its formulas and where it mentions “size,” “shrink-to-fit,” and “stretch.”[1]

[1] In the CSS 2.1 specification, the terms “size” and “sized” occur 15 times in Chapters 8, 9, 10, 11, 17, and 18. These occurrences refer to the general sense that a box has size.

Of course, sizing, shrinkwrapping, and stretching are not new ideas. What is innovative is that this book clearly defines these three terms and shows how they are a foundational feature of CSS and a key *generator* of CSS design patterns.

## Box Model Placement

Another innovation is the idea that there are three ways a box can be placed in relation to its container or its siblings: specifically, it can be indented (or outdented), offset from its siblings, or aligned and offset from its container (see Chapter 8). The CSS specification talks much about *offsetting* positioned elements, and it talks a little about *aligning* elements (see Chapter 9 of the CSS 2.1 specification), but it does not discuss how elements can be *indented*, although this behavior is implied in its formulas.

Indenting, offsetting, and aligning are different behaviors. For example, an *indented* box is stretched and its margins shrink its width, whereas an *aligned* box is sized or shrinkwrapped and its margins do not shrink its width. Aligned and indented boxes are aligned to their containers, whereas offset boxes can be offset from their container or offset from their siblings.

Different combinations of properties and property values are needed to indent, offset, and align different types of boxes. The design patterns in Chapters 8 and 9 show how this is done.

Of course, indenting, offsetting, and aligning are not new ideas. What is innovative is that this book clearly defines these three terms and shows how they are a foundational feature of CSS and a key *generator* of CSS design patterns.

## Column Layouts

Another innovation is the discovery, naming, and documenting of 12 automated techniques built into browsers for laying out columns in tables (see Chapter 16).

All the major browsers include these powerful column layout features. They are compatible across the major browsers and are very reliable. Even though using tables for page layout is not recommended,[2] *tabular data* still needs to be laid out, and you can take advantage of these column layouts to make tabular data look great.

## Fluid Layouts

Another innovation is fluid layouts (see Chapter 17). The concept of fluid layouts is not new, but the process of creating them is commonly one of trial and error. In Chapter 17, we present four simple design patterns you can use to create complex fluid layouts with confidence and predictability in all major browsers.

---

The terms “shrink” and “shrink-to-fit” occur nine times in Chapters 9 and 10 of the CSS 2.1 specification. The idea that different boxes can shrinkwrap to fit their content is implied in Sections 10.3.5 through 10.3.9 and Section 17.5.2.

The terms “stretch” and “stretched” occur four times in Chapters 9 and 16. The idea of stretching a box to its container is mentioned in passing as shown in the following quote (italics added), “many box positions *and sizes* are calculated with respect to the edges of a rectangular box called a containing block.” (See Sections 9.1.2, 9.3.1, and 10.1.)

[2] Using tables for layout creates accessibility issues for nonsighted users. Furthermore, fluid layout techniques (as shown in Chapter 17) are completely accessible and much more adaptable than tables.

These design patterns, Outside-In Box, Floating Section, Float Divider, and Fluid Layout, use floats and percentage widths to make them fluid, but they do so without the problems you normally encounter using these techniques, such as collapsed containers, staggered floats, and percentages that push floats below each other.[3]

The Fluid Layout design pattern creates columnar layouts with the versatility of tables but without using tables. Even better than tables, these layouts automatically adjust their width and reflow from columns into rows as needed to fit into narrow displays.

## Event Styling

Another innovation is the Event Styling JavaScript Framework presented in Chapter 17. This is a simple, powerful, open source framework for *dynamically and interactively* styling a document. It uses the latest best practices to ensure that HTML markup is completely free of JavaScript code and completely accessible, and all styling is done with CSS. Furthermore, the framework allows you to select elements in JavaScript using the *same selectors* you use to select elements in CSS. This vastly simplifies and unifies the styling and scripting of a dynamic HTML document!

The book includes this framework to show how to integrate JavaScript, CSS, and HTML so you can use styles interactively. Of course, if you do not want to use JavaScript, you can skip over the five JavaScript design patterns in Chapter 17 and the two JavaScript patterns in Chapter 20—the remaining 343+ design patterns do not use JavaScript.

## Combining HTML5 and CSS3 to Create Design Patterns

The final and most pervasive innovation in the book is the idea of combining general *types* of HTML elements with CSS properties to create design patterns. The book defines four major types of HTML elements in Chapter 2 (structural block, terminal block, multi-purpose block, and inline), and Chapter 4 maps them to the six box models (inline, inline-block, block, table, absolute, and float).

Each design pattern specifies how it applies to *types* of HTML elements. In other words, a design pattern is more than a recipe that works only when you use specific elements; it is a pattern that applies to all equivalent *types* of HTML elements.

For example, the Floating Drop Cap design pattern in Chapter 18 specifies a pattern that uses block and inline elements, but it does not specify which block and inline elements you have to use (see Listing 1). For example, you could use a paragraph for the **BLOCK** element and a span for the **INLINE** element (see Listing 2), or you could use a division for the **BLOCK** and a **** for the **INLINE**, and so forth.

In some exceptional cases, a design pattern may specify an actual element, like a ****. This happens when a specific element is the best solution, the only solution, or an extremely common solution. Even in these cases, you can usually swap out the specified element for another element of the same type.

***1. Listing 1. Floating Drop Cap Design Pattern***

### HTML

```
<BLOCK class="hanging-indent">
  <INLINE class="hanging-dropcap"> text </INLINE>
</BLOCK>
```

[3] Internet Explorer 6 has a number of *bugs* that may occur when you float elements. Unfortunately, there is no way to create a solution that always bypasses these bugs, although the Fluid Layout design pattern does a good job of avoiding them most of the time. Fortunately, Internet Explorer 7 fixes these bugs.

### CSS

```
.hanging-indent { padding-left:+VALUE; text-indent:-VALUE; margin-top:±VALUE; }
.hanging-dropcap { position:relative; top:±VALUE; left:-VALUE; font-size:+SIZE;
  line-height:+SIZE; }
```

*2.* ***Listing 2.*** *Floating Drop Cap Example*

### HTML

```
<p class="hanging-indent">
  <span class="hanging-dropcap" >H</span>anging Dropcap.
</p>
```

### CSS

```
.hanging-indent { padding-left:50px; text-indent:-50px; margin-top:-25px; }
.hanging-dropcap { position:relative; top:0.55em; left:-3px; font-size:60px;
  line-height:60px; }
```

# Conventions

Each design pattern uses the following conventions:

- Uppercase tokens should be replaced with actual values. (Notice how the uppercase tokens in Listing 1 are replaced with values in Listing 2.)
- Elements are uppercase when you should replace them with elements of your choice. If an element name is lowercase, it should not be changed unless you ensure the change produces the same box model. The following are typical element placeholders:
  - `ELEMENT` represents any type of element.
  - `INLINE` represents inline elements.
  - `INLINE_TEXT` represents inline elements that contain text such as `<span>`, `<em>`, or `<code>`.
  - `BLOCK` represents block elements.
  - `TERMINAL_BLOCK` represents terminal block elements.
  - `INLINE_BLOCK` represents inline block elements.
  - `HEADING` represents `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`.
  - `PARENT` represents any element that can be a valid parent of its children.
  - `CHILD` represents any element that can be a valid child of its parent.
  - `LIST` represents any list element including `<ol>`, `<ul>`, and `<dl>`.
  - `LIST_ITEM` represents any list item including `<li>`, `<dd>`, and `<dt>`.

- **Selectors** that you should replace are uppercase. If a selector contains lowercase text, that part of the selector should not be changed unless you also modify the HTML pattern, such as changing a class name. The following are typical placeholders:
    - `SELECTOR {}` represents any selector.
    - `INLINE_SELECTOR {}` represents any selector that selects inline elements.
    - `INLINE_BLOCK_SELECTOR {}` represents any selector that selects inline-block elements.
    - `BLOCK_SELECTOR {}` represents any selector that selects block elements.
    - `TERMINAL_BLOCK_SELECTOR {}` represents any selector that selects terminal block elements.
    - `SIZED_BLOCK_SELECTOR {}` represents any selector that selects sized block elements.
    - `TABLE_SELECTOR {}` represents any selector that selects table elements.
    - `CELL_SELECTOR {}` represents any selector that selects table cell elements.
    - `PARENT_SELECTOR {}` represents any selector that selects the parent in the design pattern.
    - `SIBLING_SELECTOR {}` represents any selector that selects the children in the pattern.
    - `TYPE {}` represents a selector that selects elements by a type of your choice such as `h1` or `span`.
    - `*.CLASS {}` represents a selector that selects elements by a class name of your choice.
    - `#ID {}` represents a selector that selects elements by an ID of your choice.
- **Values** that you should replace are represented by uppercase tokens. If a value contains lowercase text, that part of the value should not be changed. The following are typical value tokens:
    - Some values are literal and not meant to be replaced such as `0`, `-9999px`, `1px`, `1em`, `none`, `absolute`, `relative`, and `auto`. These values are always lowercase.
    - `+VALUE` represents a positive measurement greater than or equal to zero, such as `0`, `10px`, or `2em`.
    - `-VALUE` represents a positive measurement less than or equal to zero, such as `0`, `-10px`, or `-2em`.
    - `±VALUE` represents any measurement.
    - `VALUEem` represents an em measurement.
    - `VALUEpx` represents a pixel measurement.
    - `VALUE%` represents a percentage measurement.

- **VALUE_OR_PERCENT** represents a value that can be a measurement or a percentage.
- **WIDTH STYLE COLOR** represents multiple property values, such as those required by **border**. We use an uppercase token for each value.
- **url("FILE.EXT")** represents a background image where you replace **FILE.EXT** with the URL of the image.
- **CONSTANT** represents a valid constant value. For example, **white-space** allows three constant values: **normal**, **pre**, and **nowrap**. For convenience, we often list the valid constant values in uppercase with underscores in between each possible value, such as **NORMAL_PRE_NOWRAP**.
- **ABSOLUTE_FIXED** represents a list of constant values from which you can choose one value. The underscore separates the constant values. The complete list of values for **position** includes **static**, **relative**, **absolute**, and **fixed**. If a design pattern works only for **absolute** and **fixed**, the pattern specifies **position:ABSOLUTE_FIXED**. If it works for all four values, it specifies **position:STATIC_RELATIVE_ABSOLUTE_FIXED** or **position:CONSTANT**.
- **-(TAB_BOTTOM + EXTRA_BORDER + EXTRA_PADDING)** is an example of a formula that you would replace with a calculated value. The uppercase tokens in the formula are tokens that occur elsewhere in the design pattern. For example, if you assigned **TAB_BOTTOM** to **10px**, **EXTRA_BORDER** to **10px**, and **EXTRA_PADDING** to **10px**, you would replace the formula with **-30px**.

## Using This Book

You can use the book to master CSS. You can read straight through the book to take your CSS skills to a higher level and to discover the many golden nuggets tucked away inside design patterns. Each chapter is organized so that it builds on design patterns presented earlier in the chapter and presented in previous chapters. On the other hand, since individual chapters and design patterns are self-contained, you can read them one by one in any sequence to master a specific topic or technique.

**You can use the book as a reference book.** This book explains all of the usable CSS properties and shows how to use them in examples. Even more importantly, many properties behave differently when combined with other properties. Each design pattern identifies and documents the unique combination of properties required to create a specific result. This makes it a reference book not only for how CSS properties work alone, but also for how they work *in combination*.

**You can use the book to learn by example.** Since all examples in the book follow best practices, you can learn good habits and techniques just by studying them. To make studying the book by example easier, you can use the "See also" sections to look up all related design patterns. This allows you to easily see many examples of how a specific CSS property or feature can be used in a variety of contexts.

**You can use the book as a cookbook** to help you create designs or to solve problems. Design patterns are organized by topic so you can quickly find related solutions.

We have added extra features to the book to make it easy to find a solution when you need it. You can use the table of contents, the index, thumb tabs, chapter outlines, design pattern names, and the "See also" section of each design pattern to quickly find properties, patterns, answers, and solutions. Since the screenshots in each example are in the same location on every page, you can even thumb through the book while looking at screenshots to find a solution. We find visual scanning a very easy, fast, and effective way to find solutions!

## How This Book Is Structured

Chapters 1 through 3 explore the fundamentals of CSS and HTML:

- Chapter 1 shows how design patterns make CSS easy. Here we demonstrate how to combine simple design patterns into more complex and powerful patterns. We also review the syntax of CSS and the cascade order. In addition, we present several charts that make using CSS easier: a list of links to useful CSS resources, a summary of CSS properties, a four-page listing of all usable CSS *properties, values, and selectors* organized by where they can be used; charts on units of measure and font size; two example style sheets for normalizing the styles of elements in all browsers; media queries; transitions; animations; and 2D transformations; and a 12-step guide to troubleshooting CSS.
- Chapter 2 introduces the design patterns that underlie HTML. In this chapter, we present the best practices of using HTML for coding and for coding in XHTML. We also explore the types of structures you can create with HTML, including normal blocks, terminal blocks, multi-purpose blocks, and inlines. We also show how to use IDs and attributes for easy selection by CSS selectors.
- Chapter 3 introduces design patterns for CSS selectors and inheritance. Here we demonstrate how selectors are the bridge between HTML and CSS. We present design patterns for type, class, ID, position, group, universal, descendant, pseudo-class, and subclass selectors. We also explore CSS inheritance.

Chapters 4 through 6 explore the six CSS box models. They show how each HTML element is rendered as one of these six types of boxes (or not rendered at all). They demonstrate how the same properties produce different results in each box model, and how each box model flows differently from the other box models.

- Chapter 4 explores the six box models: inline, inline-block, block, table, absolute, and float.
- Chapter 5 explores the three ways of determining a box's size: shrink-to-append, or stretch-to.
- Chapter 6 explores each of the box model properties: margin, border radius, shadows, outline, padding, background, overflow, visibility, and pagebreak.

Chapters 7 through 9 explore how boxes flow or are positioned.

- Chapter 7 explores the five positioning models (static, absolute, relative, fixed, and floated) and relates them to the six box models.
- Chapter 8 explores the three ways a box can be positioned—for example, a box can be indented or outdented, offset from its siblings, or aligned and offset from its container.
- Chapter 9 combines the patterns in Chapters 7 and 8. The combinations result in more than 50 design patterns for positioning elements—with a particular focus on absolute and fixed positioning.

Chapters 10 through 12 explore in detail how inline boxes flow and how to style, space, and align text and objects.

- **Chapter 10 explores the properties that style text** and also contains three design patterns for hiding text while remaining accessible to nonsighted users. It also presents advanced techniques like text replacement with canvas and vml, and CSS3 font-embedding.
- Chapter 11 shows how to *space* inline content horizontally and vertically.
- Chapter 12 shows how to *align* inline content horizontally and vertically.

Chapters 13 and 14 explore in detail how blocks and images flow and how they can be styled.

- **Chapter 13 explores blocks**, starting with a discussion of the structural meaning of blocks and how you can visually display that meaning. It covers lists, inlining blocks, collapsed margins, run-in blocks, block spacing, and marginal blocks.
- **Chapter 14 explores images**, such as image maps, semi-transparent images, replacing text with images, sprites, shadowed images, and rounded corners.

Chapters 15 and 16 explore in detail how to style and lay out tables and cells.

- **Chapter 15 explores tables** including table selectors, collapsed borders, hiding cells, vertically aligning content in cells, and displaying inline and block elements as tables.
- **Chapter 16 explores laying out table columns using 12 patterns**, which automatically shrinkwrap columns, size them, proportionally distribute them, and so forth.

Chapter 17 explores how the flow of floats can be used to create fluid layouts.

- **Chapter 17 shows how to create fluid layouts** that automatically adapt to different devices, fonts, widths, and zoom factors. It also shows how to create interactive layouts using JavaScript.

Chapters 18 through 20 show how to combine design patterns to create a variety of solutions to the same problem. Each solution addresses different needs and has different advantages and disadvantages. Besides being useful solutions in and of themselves, they demonstrate how you can combine patterns to solve any design problem.

- **Chapter 18 explores drop caps.** Here we cover seven types of drop caps using seven different combinations of design patterns.
- **Chapter 19 explores callouts and quotes.** The chapter demonstrates five types of callouts and three types of quotes.
- **Chapter 20 explores alerts.** Here we present three types of interactive alerts and eight types of text alerts (i.e., attention getters). It also explores HTML5 Form Validation and shows how to natively validate HTML5 forms and alert users for wrong input.

## Downloading the Code

You can download all the code at `www.apress.com` by searching for and going to the detail page for *Pro HTML5 and CSS3 Design Patterns*. On the book's detail page is a link to the sample code compressed into a ZIP file.

## Using the Code

The code is arranged in folders, with a folder for each chapter. To make chapter folders easy to navigate, each folder name includes the chapter number and title. Inside each chapter folder are example folders: one for each design pattern presented in the chapter.

So you can easily find examples, each example folder has the same name as its design pattern. This makes it easy and fast to find design patterns by searching folder names. Since the HTML in each example names and describes its design pattern, you can find a design pattern by searching for words inside HTML files. You could also search inside CSS files for examples that use a particular CSS property, such as **`display`**.

To make it easy to view examples in multiple browsers, we put a file named **`index.html`** in the root folder that links to all design pattern folders. In turn, each folder contains a file named **`index.html`** that links to all the design patterns in that folder. These navigation pages make it quick to find and view each design pattern in each chapter.

Each example folder contains *all* the files needed to make the example work. This makes it a breeze to use the examples in your own work: simply copy a folder and start making changes. You don't have to worry about tracking down and including files from other folders.

The most important files in each example folder are **`example.html`** and **`page.css`**. **`example.html`** contains the HTML code for the example. **`page.css`** is the main style sheet for the example.

Each example also uses a CSS file named **`site.css`**. It contains a few nonessential font and heading rules that give all the examples in the book the same basic look and feel.

In a few exceptional cases, we use an additional CSS file to overcome bugs or nonstandard behavior in Internet Explorer and these rules override rules in **`page.css`**.

The seven JavaScript examples use five JavaScript files. These are explained in the Event Styling design pattern in Chapter 17. **`page.js`** is the most important file because it contains JavaScript code specific to the example. The remaining JavaScript files are open source libraries.

Lastly, each example folder contains all image files used by that example.

## Contacting the Authors

You can contact us at the following addresses:

- Michael Bowers at **`mike@cssDesignPatterns.com`**
- Dionysios Synodinos at **`synodinos@gmail.com`**

We look forward to your comments, suggestions, and questions.

CHAPTER 1

# Design Patterns: Making CSS Easy!

On the surface, CSS seems easy. It has 45 commonly used properties you can employ to style a document. Below the surface, different combinations of properties and property values trigger completely different results. I call this **CSS polymorphism** because the same property has many meanings. The result of CSS polymorphism is a combinatorial explosion of possibilities.

Learning CSS is more than learning about individual properties. It is about learning the contexts in which properties can be used and how different types of property values work differently in each context. As an example, take the `width` property, which has many different meanings depending on how it is combined with other rules and what values are assigned to it. For instance, `width` has absolutely no effect on inlines. `width:auto` shrinkwraps floats to the width of their content. `width:auto` shrinkwraps absolutes when `left` and `right` are set to `auto`. `width:auto` stretches blocks to the width of their parent element. `width:auto` stretches absolutes to the width of their containing block when `left` and `right` are set to `0`. `width:100%` stretches blocks and floats to the width of their parent element as long as they do not have borders, padding, and margins. `width:100%` stretches tables to the width of their parent even if they do have borders and padding. `width:100%` stretches absolutes to the width of their closest positioned ancestor instead of their parent. `width:100em` sizes an element in relation to the height of its `font-size`, which allows the element to be sized wide enough to contain a certain number of characters. `width:100px` sizes an element to a fixed number of pixels regardless of the `font-size` of its text.

To complicate matters further, not all of the rules are implemented by browsers. For example, over 40 out of 122 properties and over 250 out of 600 CSS rules are not implemented by one or more of the major browsers. CSS combines several specifications that define various levels and profiles. Each level of CSS builds upon the last, typically adding new features and typically denoted as CSS 1, CSS 2, and CSS 3. Profiles are typically a subset of one or more levels of CSS built for a particular device or user interface. Browser support for CSS3 is an important issue for developers, especially since it is still rapidly evolving as a specification.

Trying to learn CSS by memorizing the extraordinary number of exceptions to each rule is extremely frustrating.

To make learning CSS easy, this book documents all *usable* combinations of properties and property values. It puts properties in context and paints a complete picture of how CSS works.

Imagine the time you will save by not having to read about rules that do not work and by not having to test every rule to see whether it works in every browser and in combination with other rules. I have already done this for you. I have run many thousands of tests. I have tested every CSS property and every combination of properties in every major browser, including Internet Explorer 6/7/8/9, Firefox 7, Chrome 12, Opera 9, and Safari 5.

I have boiled down these results into simple design patterns—all the CSS and HTML design patterns you need to create stunning, high-performance, and accessible web sites. This edition of the book (2nd) has been updated to include the latest information and tips about HTML5 and CSS3.

After you learn these design patterns, you'll wonder how you ever developed web sites without them!

In this chapter, I discuss the purpose of design patterns and how they work. I give some examples of how to combine design patterns to create new patterns. I also discuss how to use style sheets, CSS syntax, and the cascading order to your advantage.

Next, I present a series of charts that list all the usable CSS properties and units of measure. I then present 12 techniques for troubleshooting CSS quickly. Lastly, I discuss how to standardize the way various browsers style elements—so you can override these default styles with confidence.

## Design Patterns—Structured Recipes

Design patterns have been used with great success in software programming. They improve productivity, creativity, and efficiency in web design and development, and they reduce code bloat and complexity. In the context of CSS and HTML, design patterns are sets of common functionality that work across various browsers and screen readers, without sacrificing design values or accessibility or relying on hacks and filters. But until now they have not been applied systematically to HTML and CSS web design and development.

Design patterns underlie all creative activities. We think in terms of patterns when we talk, write, and create. Design patterns are similar to document templates that we can fill in with our own content. In literature, they are like archetypal characters and plots. In music, they are like themes and variations. In programming, they are similar to reusable algorithms that can be systematically varied and combined with each other to produce a desired result.

Once a design pattern is revealed, it greatly increases creativity and productivity. It can be used by itself to create quick results, and it can be easily combined with other patterns to create more complex results. Design patterns simplify and amplify the creative process. They make creation as easy as building with blocks or Legos. You simply choose predesigned patterns, vary them, and combine them to create the result you want. Patterns do not limit creativity—they unleash creativity.

The seminal work *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (Addison-Wesley, 1995), explains that a design pattern consists of four elements: a pattern name, a problem, a solution, and trade-offs. This book follows this approach.

Since this is a practical book, it focuses directly on the concrete patterns designed into CSS and HTML that are actually implemented in the major browsers. This book also creates new design patterns by combining built-in patterns into higher-level patterns.

In a very real sense, this is a book of patterns that you can use to create your designs.

## Using Design Patterns

Chapters 1 through 7 present the basic properties and elements for styling layout. Chapters 8 and 9 combine these properties to create all possible block, positioned, and floated layouts. Chapters 10 through 12 present the basic properties for styling text and also present combinations of properties you can use to create inline layouts. Chapters 13 through 16 combine design patterns from previous chapters with specialty properties and elements to style blocks, lists, images, tables, and table columns.

Together, Chapters 1 through 16 present over 300 design patterns created by combining 45 common CSS properties with four types of elements (inline, inline-block, block, and table) and five types of positioning (static, relative, absolute, fixed, and float).

This is the great power of design patterns: it is easy to take basic patterns and combine them to form more complex patterns. This makes learning CSS easy, and it makes using CSS very productive. Chapters 17 through 20 show how to combine these design patterns to create fluid layouts, drop caps, callouts, quotes, and alerts.

To illustrate the simplicity and power of design patterns, the next five examples show how to take a series of basic design patterns and combine them into more complex patterns. You do not need to understand the details of each pattern—just the process of combining patterns.

The first example in this series shows the `background` property in action. `background` is a design pattern built into CSS that displays an image behind an element. Example 1-1 shows the `background`

property combined with a division element. The division is sized 250 by 76 pixels so it will reveal the entire background image.[1]

## Example 1-1. Background Image


### HTML

```
<h1>Background Image</h1>
<div></div>
```

### CSS

```
div { background:url("heading2.jpg") no-repeat; width:250px; height:76px; }
```

Example 1-2 demonstrates the Absolute design pattern. The idea behind the Absolute design pattern is to remove an element from the flow and position it relative to another element. CSS provides the `position:absolute` rule for this purpose. When `position:absolute` is combined with the `top` and `left` properties, you can position an element at an offset from the top left of its closest positioned ancestor. I used `position:relative` to position the division so it would be the closest positioned ancestor to the span. I then absolutely positioned the span 10 pixels from the top and left sides of the division.[2]

[1] This example is simple and yet it combines seven design patterns: the Structural Block Elements design pattern in Chapter 2; the Type Selector pattern in Chapter 3; the Block Box pattern in Chapter 4; the Width, Height, and Sized patterns in Chapter 5; and the Background design pattern in Chapter 6.

[2] This example is simple, and yet it combines seven design patterns: the Inline Elements and Structural Block Elements design patterns in Chapter 2; the Class Selector pattern in Chapter 3; the Absolute Box pattern in Chapter 4; and the Absolute, Relative, and the Closest Positioned Ancestor patterns in Chapter 7.

Example 1-2. Absolute


## HTML

```
 <h1>Absolute</h1>

<div class="positioned">
  <span class="absolute">Sized Absolute</span>
</div>
```

## CSS

```
*.positioned { position:relative; }
*.absolute { position:absolute; top:10px; left:10px; }

/* Nonessential styles are not shown */
```

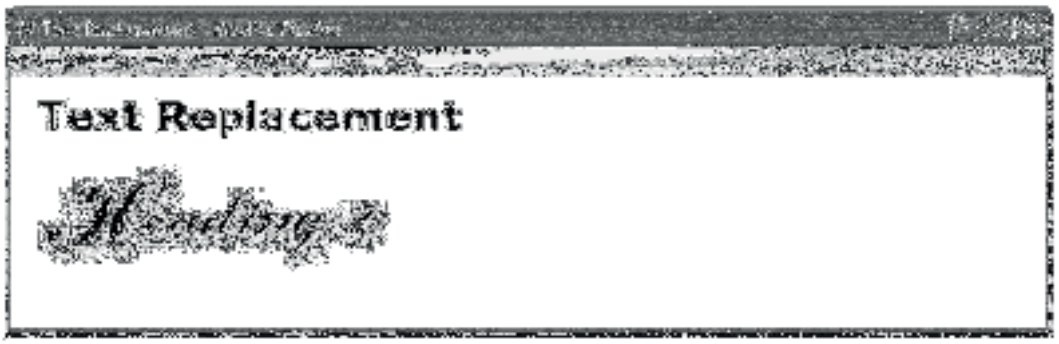
Example 1-3 combines the design patterns in the first two examples to create the Text Replacement design pattern. The idea behind text replacement is to display an image in the place of some text (so you can have more stylistic control over the text because it is embedded in an image). In addition, you want the text to be present behind the image so that it becomes visible if the image fails to download.

I combined the Background and Absolute design patterns to create the Text Replacement pattern. I placed an empty span inside a heading. I relatively positioned the heading so child elements can be absolutely positioned relative to it. I assigned a background image to the span and absolutely positioned it in front of the text in the heading element. I sized the span and the heading to the exact size of the background image.

The end result is that the background image of the span covers the text in the heading, and if the image fails to download, the styled text in the heading is revealed.[3]

[3] The Text Replacement example uses the 14 design patterns shown in the previous two examples. It also introduces the ID Selector design pattern in Chapter 3. You can learn more about the Text Replacement design pattern in Chapter 10.

Example 1-3. Text Replacement




## HTML

```
<h1>Text Replacement</h1>
<h2 id="h2" >Heading 2<span></span></h2>
```

## CSS

```
#h2 { position:relative; width:250px; height:76px; overflow:hidden; }

#h2 span { position:absolute; width:250px; height:76px; left:0; top:0;
  background:url("heading2.jpg") no-repeat; }
```

Example 1-4 demonstrates the Left Marginal design pattern. The idea behind this pattern is to move one or more elements out of a block into its left margin so you can have headings (or notes, images, etc.) on the left and content on the right.[4]

[4] The Left Marginal design pattern combines the Position Selector design pattern in Chapter 3; the Margin pattern in Chapter 6; the Absolute Box pattern in Chapter 4; and the Absolute, Relative, and the Closest Positioned Ancestor patterns in Chapter 7.

Example 1-4. Left Marginal

## HTML

```
<h1>Left Marginal</h1>

<div class="left-marginal" >
  <h2 class="marginal-heading">Heading</h2>
  You want to excerpt an element and move it into the left margin.</div>
```

## CSS

```
*.left-marginal { position:relative; margin-left:200px; }
*.marginal-heading { position:absolute; left:-200px; top:0; margin:0; }
```

Example 1-5 demonstrates the Marginal Graphic Dropcap design pattern. This pattern combines all the design patterns shown in the previous four examples. The idea behind this pattern is to create a graphical drop cap in the left margin of a block with all the advantages of the Text Replacement and Left Marginal design patterns.[5]

To meet these requirements, I used the `indent` class to relatively position the paragraph so that it will be the closest positioned ancestor of the drop cap and to add a 120-pixel left margin to the paragraph to make room for the drop cap. I used the `graphic-dropcap` class to absolutely position the drop cap, to move it into the paragraph's left margin, and to set it to the exact size of the dropcap image. I then absolutely positioned the span inside the graphic drop cap and moved it over the dropcap text so it covers the text with its background image.

Viewed by itself, the Marginal Graphic Dropcap pattern is a somewhat complex combination of 16+ design patterns. On the other hand, when viewed as a combination of the Text Replacement and Left Marginal design patterns, it is quite simple. This is the power of design patterns.

[5] The Marginal Graphic Dropcap design pattern is discussed in detail in Chapter 18.

Example 1-5. Marginal Graphic Dropcap


### HTML

```
<h1>Marginal Graphic Dropcap</h1>

<p class="indent"><span class="graphic-dropcap" >M<span></span></span>arginal
  Graphic Dropcap. The letter M has been covered by the dropcap image.
  Screen readers read the text and visual users see the image.
  If the browser cannot display the dropcap image,
  the text becomes visible.</p>
```

### CSS

```
*.indent { position:relative; margin-left:120px; }

*.graphic-dropcap { position:absolute;
  width:120px; height:90px; left:-120px; top:0; }

*.graphic-dropcap span { position:absolute;
  width:120px; height:90px; margin:0; left:0; top:0;
  background:url("m.jpg") no-repeat; }
```

## Using Style Sheets

You can place styles in three locations: style sheets, `<style>`, and `style`.

A **style sheet** is an independent file that you can attach to an HTML document using the `<link>` element or CSS's `@import` statement. `<style>` is an HTML element that you can embed within the HTML document itself. `style` is an attribute that can be embedded within any HTML element.

I recommend putting styles in style sheets. This reduces noncontent in your HTML documents, and it puts all your styles in files that are easily managed.

I recommend naming style sheets using single-word, lowercase names. This keeps style sheet names simple and easy to remember, and works safely in all operating systems. I suggest you use a name that describes the scope and purpose of the style sheet, such as `site.css`, `page.css`, `handheld.css`, `print.css`, and so forth. The standard extension for a style sheet is `.css`. The standard Internet media type is `text/css`.

I recommend using the location of a style sheet to control its scope. If a style sheet is for an entire web site, you could place it in the root directory of the web site. If a style sheet applies only to a

document, you could place it in the same directory as your document. Another option, depending on how you organize your site, is to keep all style sheets in one directory.

To link a style sheet to an HTML document, you can include a <link> element in the <head> section of HTML documents, and you can place the URL of the style sheet either in the href attribute of the <link> element. Listing 1-1 shows the style sheet links that I use in each example in this book. See the Header Elements and Conditional Stylesheet design patterns in Chapter 2 for more information on linking style sheets.

*Listing 1-1. Attaching Style Sheets*

```
<link rel="stylesheet" href="site.css" media="all" type="text/css" />
<link rel="stylesheet" href="page.css" media="all" type="text/css" />
<link rel="stylesheet" href="print.css" media="print" type="text/css" />
<!--[if lte IE 6]>
<link rel="stylesheet" href="ie6.css" media="all" type="text/css" />
<![endif]-->
```

For increased download performance, you may want to include page-specific styles in the <style> element instead of in a separate page-specific style sheet. Since these styles are page-specific, there is little disadvantage to putting these styles in the header of the page. On the other hand, I do strongly recommend against using the style attribute of HTML elements because this creates very hard to maintain code.

# CSS Syntax

CSS syntax is easy. A style sheet contains styles; a style contains selectors and rules; and a rule contains a property and a value. The following is the design pattern for a style:

```
SELECTORS { RULES }
```

The following is the design pattern for a rule:

```
PROPERTY:VALUE;
```

For example, `p{margin:0;}` is a style. `p` is the selector, which selects all <p> elements in an HTML document. The curly brackets ({}) operators assign the rule, `margin:0;`, to the selector, `p`. The colon (:) operator assigns the value `0` to the property, `margin`. The semicolon (;) operator terminates the rule.

A style may have one or more selectors and one or more rules. For example, `p.tip{margin:0; line-height:150%;}` is a style. The curly bracket operators group the two rules, `margin:0;` and `line-height:150%;`, into a ruleset and assign it to the selector, `p.tip`, which selects all `<p class="tip">` elements in an HTML document.

## CSS Syntax Details

The key points of CSS syntax are as follows:

- Unicode UTF-8 should be used to encode CSS files. The same way you should encode HTML files.
- CSS code should be lowercase. Selectors are *case-sensitive* when referencing element names, classes, attributes, and IDs in X/HTML. CSS properties and values

[1] In HTML, CSS selectors are case-insensitive.

are *case-insensitive*. For simplicity and consistency, I use lowercase characters for all CSS code including elements, classes, and IDs.

- **Element names, classes, and IDs** are restricted to letters, numbers, underscores (_), hyphens (-), and Unicode characters 161 and higher. The first character of an element, class, or ID must not be a number or a hyphen. A classname and ID must not contain punctuation other than the underscore and hyphen. For example, `my_name2-1` is a valid name for a class or ID, but the following are *invalid*: `1`, `1my_name`, `-my_name`, `my:name`, `my.name`, and `my,name`.
- **Multiple classes** can be assigned to an element by separating each class name with a space, such as `class="class1 class2 class3"`.
- **Constant values** should not be placed in quotes. For example, `color:black;` is correct, but `color:"black";` is not.
- **The backslash (\)** can be used to embed characters in a context where they normally cannot occur; for example, `\26B` embeds & in a string or identifier. Anywhere from two to eight hex codes can follow a backslash, or a character can follow a backslash.
- **A string** may contain parentheses, commas, whitespace, single quotes (`'`), and double quotes (`"`) as long as they are escaped with a backslash, such as the following:

```
"embedded left  parentheses \( "
"embedded right parentheses \) "
"embedded comma \, "
"embedded single quote \' "
"embedded double quote \" "
"embedded single quote  ' in a double-quoted string"
'embedded double quote  " in a single-quoted string'
```

- **A semicolon** should terminate each CSS rule and `@import` statement.

```
color:red;
@import "mystylesheet.css";
```

- **Rulesets** are created by enclosing multiple rules in curly braces, such as `{ color:red; font-size:small; }`.
- **The right curly brace (})** immediately terminates a set of properties, unless it is embedded within a string, such as `"}"`.
- **A CSS comment** starts with **/*** and ends with ***/**, such as `/* This is a CSS comment */`. Comments cannot be nested. Thus, the first time a browser encounters ***/** in a style sheet, it terminates the comment. If there are subsequent occurrences of **/***, they are not interpreted as part of the comment—for example:

```
/* This is an incorrect comment
  /* because it tries to nest
    /* several comments. */
      STARTING HERE, THIS TEXT IS OUTSIDE OF ALL COMMENTS! */ */
```

## Using Whitespace in CSS

Whitespace in CSS includes only the following characters: space (`\20`), tab (`\09`), new line (`\0A`), return (`\0D`), and formfeed (`\0C`). A browser will not interpret other Unicode whitespace characters as whitespace—such as the nonbreaking space (`\A0`).

You can optionally place whitespace before and after the following: selectors, curly braces, properties, colons, values, and semicolons. For example, all the following statements are correct and produce the exact same result:

```
body{font-size:20px;line-height:150%;}

body { font-size:20px; line-height:150%; }

body { font-size : 20px ; line-height : 150% ; }

body
{
  font-size:  20px;
  line-height: 150%;
}
```

In this book, I use a compact coding style in which I put no whitespace inside rules, and I put one space in between rules and selectors, such as the following:

```
body { font-size:20px; line-height:150%; }
```

Whitespace never occurs within a property name or within a constant property value. Whenever CSS uses multiple words for a property name or constant property value, it uses a hyphen to separate the words, such as `font-family` and `sans-serif`. On rare occasions, CSS uses CamelCase to combine multiple words into one constant value, such as `ThreeDLightShadow`.

## Using Property Values

Property values come in the following forms: constant text, constant numbers, lengths, percentages, functions, comma-delimited lists of values, and space-delimited series of values. Each property accepts one or more of these types of values.

I have included all common types of values in Example 1-6. But first, I have listed them here along with an explanation:

- **`color:black;`** assigns the constant value `black` to the `color` property. Most properties have unique constant values. For example, the `color` property can be assigned to over 170 constants that represent colors ranging from `papayawhip` to `ThreeDDarkShadow`.
- **`background-color:white;`** assigns the constant value `white` to the `background-color` property. Notice that the following three rules do the same thing as this rule, but use different types of property values. Hex is also commonly used for color properties in styles, e.g., `background-color:#000000;`.
- **`background-color:rgb(100%,100%,100%);`** assigns the CSS function `rgb()` to `background-color`. `rgb()` takes three comma-delimited parameters between its parentheses, which specify the amount of red, green, and blue to use for the color. In this example, percentages are used. One hundred percent of each color makes white.
- **`background-color:rgb(255,255,255);`** assigns white to the `background-color`. In this case, values from 0 to 255 are used instead of percentages. The value 0 is no

sample content of Pro HTML5 and CSS3 Design Patterns (Expert's Voice in Web Development)

- Survivors of Stalingrad: Eyewitness Accounts from the 6th Army, 1942-1943 book
- click A.D. The Bible Continues: The Revolution That Changed the World
- *click Interpreting the Landscape*
- download The Dream Manager for free
- click Wayne of Gotham
- **A Pocket Guide to Writing in History (7th Edition) here**

- http://aseasonedman.com/ebooks/Survivors-of-Stalingrad--Eyewitness-Accounts-from-the-6th-Army--1942-1943.pdf
- http://transtrade.cz/?ebooks/A-D--The-Bible-Continues--The-Revolution-That-Changed-the-World.pdf
- http://korplast.gr/lib/Billy-Bathgate.pdf
- http://conexdxb.com/library/As-Consciousness-Is-Harnessed-to-Flesh--Journals-and-Notebooks--1964-1980.pdf
- http://dadhoc.com/lib/Teor--a-y-pr--ctica-del-amor.pdf
- http://anvilpr.com/library/Ezra-Pound-s-Adams-Cantos--Historicizing-Modernism-.pdf