

# CHAPTER 9

## STYLING TEXT

### King Kong

#### The Legend

#### The Extravaganza

#### References

King Kong is the name of a *fiction*, who has appeared in several works and include the groundbreaking 1933 and numerous sequels.

In the original film, the character's inhabitants of Skull Island, where it which miraculously escaped extinction. American film crew led by Carl Denham New York City to be exhibited as the

#### REFERENCES

1. [The Lumiere Reader – King Kong](#)
2. [King Kong – An Entertaining](#)
3. [Kong is King – The History of](#)

In the last chapter we introduced you to the basics of CSS, adding a little style to our King Kong web page. In this chapter we go a bit further, styling all of the typographic elements on the mighty gorilla’s web page. Cue thunderous roar from the jungle.

We deliberately kept things simple with our King Kong page in the last chapter, styling only a few elements: the `body`, `h1`, `h2`, and `p`. In this chapter we’ll build on what we demonstrated in the last chapter and delve a little bit deeper into CSS, introducing and expanding on some underlying concepts along the way.

This chapter’s focus is specifically on styling text, an area CSS is particularly well suited to. By the end of the chapter you should not only have a deeper understanding of how to use CSS, but also have an insight into how powerful it can be in transforming any well-structured XHTML web page into a well-styled, great-looking web page. This chapter also introduces a number of different methods of specifying font sizes in CSS: pixels, ems, and keywords. All have strengths. Unfortunately, all *also* have weaknesses.

We conclude the chapter with a walkthrough of our new, improved King Kong page, along the way introducing a wide variety of ways to make your text more readable and more pleasing to the eye.

## Typography on the Web

Working through another pass of the CSS of our King Kong web page will introduce you to some of the fundamentals of typography, including some general typographic principles and a few new terms. It will also highlight how designing for the Web, particularly when it comes to typography, has its limitations, but equally can offer a number of opportunities and advantages.

Before we embark on a look at the specifics of typography on the Web, it’s worth defining what exactly typography is.

### What is typography?

Typography is often defined in traditional terms referring to the world of print—books, magazines, newspapers . . . in short, anything *printed*. However, typography isn’t just appropriate to print, it’s also appropriate to everything we create onscreen. In short, anywhere that type is used.

We can define typography as follows:

**typography** –*n.* **1** the art or process of setting and arranging types and printing from them **2** the style and appearance of printed matter

*Concise Oxford Dictionary* (Clarendon Press, 1990)

However, this definition implies typography is limited to print, but as you learned hands-on in the last chapter, typography is also used online. After all, we set and arranged some type when we specified both a font-family and a font-size for the h1, h2, and p elements on our King Kong web page.

Clearly typography is *also* of interest on the Web.

*In traditional typographic terminology, a typeface and a font are not the same. A **typeface** refers to a whole family of type, for example, Times New Roman. A **font**, however, refers to one instance of this typeface, for example, Time New Roman, Italic, 12 pt. This is because historically every typeface would have been comprised of a number of fonts, all cast in metal and set by hand. In a web-based context, however, the terms typeface and font are used interchangeably.*

One of the greatest benefits of using CSS to style text is the control it gives us over typography and, equally importantly, its flexibility.

## CSS: Our flexible friend

The separation of your content, marked up as well-structured XHTML, and its visual presentation, controlled by your CSS rules, makes for a remarkably flexible and efficient way of applying pages.

Style is not only easy to apply, but also easy to change: simply alter a rule in your style sheet and *Hey Presto!* your page is displayed using a different typeface at a different size. This enables you to quickly prototype and test design ideas, adjusting your page's typography through nothing more than simple changes and modifications to your style sheet. Herein lies the real power of CSS.

You'll discover this to your delight if approaching a redesign of your XHTML pages. In most cases you can simply take your existing content, marked up semantically, rework the CSS rules in your style sheet, and have a new design ready in an instant.

## Making your text accessible

As Joe Clark, a noted accessibility advocate, states,

*Reading is the primary activity of the Web. For people with impaired vision who do not use screen readers, colour choices and, to a far lesser extent, type size become the accessibility issues.*

[www.joeclark.org/book/sashay/serialization/Chapter09.html](http://www.joeclark.org/book/sashay/serialization/Chapter09.html)

An added benefit of CSS, which leverages the separation of content and presentation inherent in the Web Standardistas' approach, lies in the ability to create alternative style

sheets for different users. One example would be the creation of a high-contrast style sheet for visually impaired users—easy to create, given the solid foundation of semantic markup that you are now used to creating.

Another advantage of CSS is that it gives a greater level of control to the user, for example, allowing the user to resize text or increase the contrast on the page, making the pages more accessible. While at first this lack of control can appear daunting from a designer’s perspective, it’s important to remember that the primary purpose of the Web as envisaged by Tim Berners-Lee was to make as much information as possible accessible to as wide an audience as possible. A good thing.

## Inheritance and specificity

In the last chapter we applied some basic style to the elements on our King Kong page. To refresh your memory, among a few other rules, we specified a font-family and text color for our web page, as shown in the following example:

```
body
{
font-family: Arial;
color: #F3F1EC;
...
}
```

Looking at this rule again you might be forgiven for asking, “Why are we styling a font-family and color on the body element? Why not on the elements themselves? The h1s, h2s, ps, and so on?”

The answer, as we alluded to in the previous chapter, lies in the issue of *inheritance*. Although it can take a little getting used to, the concept of inheritance lies at the heart of CSS and is essential to grasp if you want to create efficient web pages with a minimum of markup. Let’s dive right in.

## Inheritance

In Chapter 2 we introduced the idea of the document tree, explaining how elements can be nested. You might recall that our simple “Hello World!” web page had a single p nested within the body. Another way of describing the relationship between these elements would be to state that the p element is *descended* from its parent element, the body.

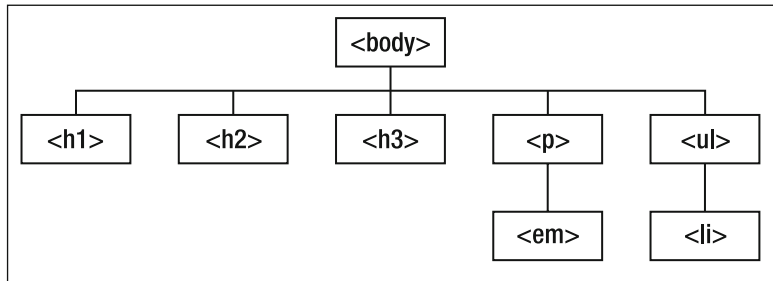
So the p is a child of the body, and conversely, the body is a parent of the p. The important point to note is that *the elements have a relationship to each other*.

Just like children inherit characteristics from their parents in the real world, child elements inherit characteristics from their parent elements in the world of CSS.

If we look at our King Kong web page, we can see that the body element contains the following child elements: h1–h4, p, em, strong, blockquote, cite, abbr, ul, ol, li, a, and img.

Considerably more child elements than the “Hello World!” web page we first built, proof—if it were needed—of your growing capabilities as a Web Standardista.

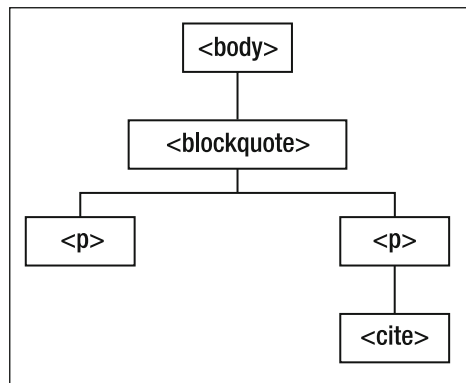
In Figure 9-1 we’ve illustrated a simplified version of the document tree for the King Kong web page (for the purposes of simplicity, we haven’t included *all* of the elements descended from the body).



**Figure 9-1.** The relationship between the body and its descendants

Looking at Figure 9-1, you’ll also notice that in addition to all of the elements on the King Kong page being descended from the body element, there are other elements with child elements. In our simplified example, both the `p` and `ul` elements have children, the former an `em`, the latter an `li`.

The King Kong page we’ve created also has another element descended from the body which itself has a number of child elements. The `blockquote` features two children: two `p`s, one of which has a descendant, namely a `cite` element, as shown in Figure 9-2.



**Figure 9-2.** Our `cite` element is a child of a `p`, itself a child of the `blockquote` which, in turn, is a child of the body. Think of this as a family tree.

Given that all of these elements are descended from the body element, they *inherit* any rules applied to the body. What this means is that we don’t need to write rules for each of the elements we’d like to style; we can instead rely on inheritance to take care of this for us, as you briefly saw in the last chapter when we applied some style to our body element.

Bravo! This is going to result in a lot less work down the line. But you might be thinking that this all looks just a little bit too easy . . .

## Meet specificity

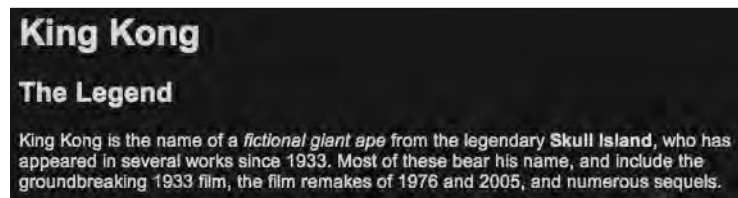
Now we've ascertained that all the CSS declarations we apply to the body element are inherited by its descendants, let's take a look at this in action. We've created a simplified version of our King Kong web page, removing most of the content so we can focus on the h1, h2, and p elements. We've created a single CSS rule to look at an example of inheritance in action. Our simplified web page looks like this:

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

    body
    {
    font-family: Arial;
    font-size: 14px;
    color: #E0FDA;
    background-color: #26201C;
    }

</style>
</head>
<body>
  <h1>King Kong</h1>
  <h2>The Legend</h2>
  <p>King Kong is the name of a <em>fictional giant ape</em> from
  the legendary <strong>Skull Island</strong>, who has appeared in
  several works since 1933. Most of these bear his name, and include
  the groundbreaking 1933 film, the film remakes of 1976 and 2005,
  and numerous sequels.</p>
</body>
```

According to the rules of inheritance we introduced earlier, we might imagine that everything on our simple web page would display in Arial at a size of 14 pixels, in cream type against a dark brown background. Let's take a look at Figure 9-3, which shows how this displays in the browser.



**Figure 9-3.** Our CSS specified a size of 14px for the body, so why aren't the h1 and h2 inheriting the 14px setting?

We'd hoped *everything* would display in Arial at a size of 14px. Our h1, h2, and p elements are displaying in Arial as we expected; however, the type is clearly not all set to 14px. Why? To answer that question we need to delve a little bit deeper into the complexities of **specificity**.

To explain this riddle, we return to our previous example where we've specified a `font-size` on the body. This rule should, according to the principles of inheritance, be applied to all of its descendants: the h1, h2, and p.

However, the examples we covered in the previous chapter showed us that if we wanted to change the `font-size` of a child element, the h1 for example, we could do so by targeting this element with a rule that would be *more specific* and therefore override the less specific rule we applied to its parent, the body.

But in this case there's no rule targeting the h1 in our style sheet—we only applied style to the body. So why is the `font-size` not 14px? The answer is buried deep inside your browser, in something known as the browser's *default style sheet*.

*We've mentioned default styles and the browser's default style sheet previously, but what is the browser's default style sheet? The CSS rules and declarations that apply style to your web pages can in fact come from a number of sources:*

**Author Styles:** *The CSS rules we have been writing for our King Kong page are known as Author Styles, as they are style sheets provided by the author of the web page. When we mention style sheets, rules, and declarations throughout this book, we're in fact referring to Author Styles.*

**User Styles:** *The user of your web page can also create CSS rules, usually through options in their web browser. These style sheets are applied to all web pages and may override the Author Styles. User Styles might be created to enlarge text or to increase contrast, by a visually impaired user, for example.*

**User Agent Styles:** *Finally, User Agent Styles, also referred to as the browser's default style sheet, control the browser's default presentation of XHTML elements. The "unstyled" XHTML pages we worked on in the first six chapters of this book have in fact been styled all along, using the browser's default style sheet. Although these styles vary slightly between browsers, they do share common characteristics.*

In the example in Figure 9-3, we used the body selector to assign some style to our page; however, it didn't affect the size of headings. That's because the browser's default style sheet is also assigning style to our page, in particular, controlling the size of the headings.

The browser's default style sheet contains a more specific rule, in this case targeting the h1 and h2 elements directly. In other words, the browser's more specific h1 and h2 rules take precedence over our less-specific body rule.

In CSS, when two or more conflicting rules are controlling the same element, the browser needs to determine which rule to follow. It does this by following a number of basic (but rather complex) rules, as we'll see in Chapter 10. In this case, the browser's default style sheet is winning and overriding the size we set for the headings.

Now that we’ve covered the tricky topics of inheritance and specificity, let’s get back to the focus of this chapter: styling text.

## Specifying type on the Web

In the last chapter we specified a typeface for our body element using CSS. We deliberately kept things simple, specifying Arial, a typeface most people would be familiar with. However, what if we’d like to use a different typeface? Something a little more exotic perhaps?

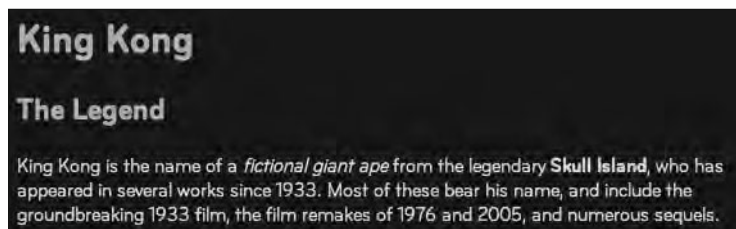
As anyone who has ever used Microsoft Word knows, the list of typefaces available is often very large. You might be forgiven for asking, “Why can’t I specify one of these other typefaces instead?” The answer is that the display of the typeface you choose depends upon the typefaces installed on your end user’s computer, not yours.

When you specify a typeface using CSS, it will only display in that typeface if it is installed on the end user’s computer. With so many different operating systems—Windows Vista, Windows XP, Mac OS X, and the various flavors of Linux—all with different preinstalled typefaces, it’s difficult to predict with any degree of confidence what typefaces will be on your users’ computers.

So, what happens if we specify a font and the end user doesn’t have it? Let’s take a look. When creating the brand for our Famous Primates web site, we used a typeface called *Bryant*, a lovingly crafted sans serif typeface inspired by mechanical lettering kits used by draftsmen and amateur sign makers, created by the talented Eric Olson of Process Type Foundry ([www.processtypefoundry.com](http://www.processtypefoundry.com)). We’ve rewritten the body rule that specified Arial at the end of the last chapter, to specify Bryant instead, as follows:

```
body
{
font-family: Bryant;
...
}
```

Being conscientious Web Standardistas, we check it in our browser, and it displays exactly as we wanted, as in Figure 9-4.



**Figure 9-4.** Change Arial to Bryant in the CSS rule and . . . voilà! Everything changes.



Good news. The principle of inheritance is simplifying things considerably. Change the typeface on the body rule and inheritance takes care of the rest, changing the type on the whole page. Not so fast!

Being truly conscientious Web Standardistas, we know better than to check the web page only in our own browsers. We ask a friend to take a look. Disaster. Our friend sends us back a screenshot as in Figure 9-5. No Bryant, just Times New Roman. It turns out he doesn't have Bryant installed on his computer, so the type has reverted to his browser's default text style.

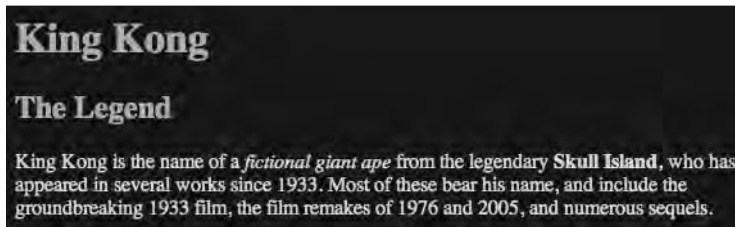


Figure 9-5. No Bryant. No joy.

Needless to say, Figure 9-5 wasn't how we intended this page to display at all. We were happy with it the way it displayed in Figure 9-4. Why has this happened? The answer is simple: we have the typeface Bryant installed, our friend doesn't. Without the typeface we've specified installed on his machine, the font reverts to what is defined by his browser's default style sheet, with the default typeface usually being Times New Roman.

So how can we be sure that a typeface we specify for a web page is available in browsers other than our own? In short, we can't 100%, but all is not lost. Why? Unusually—thanks goes to Microsoft . . .

## Core Web Fonts

In 1996, Microsoft began a project to establish a standard suite of fonts for the Internet. Released under a generous end-user license, the fonts quickly became established as a cross-platform core font set that could reasonably be relied upon to be installed on most users' computers.

Although this project was discontinued by Microsoft in 2002, the generosity of the original license still allows for distribution and use of the fonts today, with the result being that they still remain prevalent. The full list of fonts available is as follows:

- Andale Mono
- Arial
- Arial Black
- Comic Sans MS
- Courier New
- Georgia

- Impact
- Times New Roman
- Trebuchet MS
- Verdana
- Webdings

As these fonts were optimized for display and legibility on screen, the best way to see how they render is to look at them in your browser. We’ve created a page displaying all of them here:

[www.webstandardistas.com/09/core\\_fonts.html](http://www.webstandardistas.com/09/core_fonts.html)

Good news: specifying one of the preceding fonts is generally considered reliable as most users have them installed. Better news: we can use the preceding fonts to create a *fallback option* for anyone, like our friend in the example earlier, who doesn’t have our obscure, but beloved, typefaces installed. So, how do we establish a fallback font when writing CSS?

## Writing more reliable CSS rules to specify fonts

We now know that the fonts we specify in CSS will only display if the user has the same font installed on their computer. We also know that this isn’t always the case.

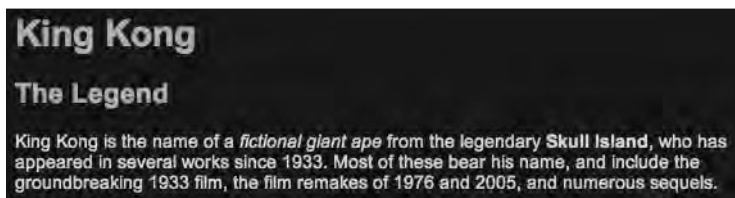
CSS provides a solution to this problem by allowing us to specify more than one font in a CSS rule so that we can provide a list of alternatives or fallback options in the event that the typeface we’d really like is not available on the user’s computer.

In the last rule, where we specified Bryant, we supplied it on its own with no fallback option. In the following rule, we’d really like Bryant to display in the first instance; however, we know not everyone has that typeface installed so we’ve provided a number of alternatives as follows:

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

The browser interprets this list of fonts in order of preference, with the first being your preferred choice. In the preceding example everything in the body will display in Bryant if the user has that typeface installed; if not it will display in Arial, and finally, as a last resort, it will display in the browser’s generic sans serif font (unless a different sans serif font has been explicitly set as a *User Style* preference, as mentioned earlier in the chapter).

Let’s return to our friend who doesn’t have the typeface Bryant; in his browser the revised page now displays as in Figure 9-6.



**Figure 9-6.** The page displayed in our friend’s browser. He doesn’t have Bryant, so the page displays using our fallback typeface, Arial.

Both Bryant and Arial are sans serif typefaces, and although we may prefer the look of the page displayed using Bryant, the fallback typefaces share the basic characteristics of our first choice. As you’ve seen from the list of Core Web Fonts, you can also be reasonably assured that Arial will be widely installed. Selecting it as a fallback font has resulted in a page that retains many of the typographic characteristics we were aiming for.

*Microsoft’s Core Web Fonts were specifically designed for use onscreen and are consequently very readable on most computers. Modern operating systems, Macintosh OS X for instance, however, have made the display of fonts originally designed for print much more viable for onscreen use.*

*When considering legibility and readability, it’s worth considering the best typeface for the job. The Guardian, which we highlighted in Chapter 3, decided to use Arial as a body font instead of Helvetica because Arial—being custom-designed for the Web—looked better in a wider range of environments.*

So, selecting a carefully considered list of fonts when specifying your font-family can help when a user doesn’t have the *exact* typeface you’d like.

Let’s take a look at this CSS rule again:

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

You may be forgiven for asking, “What typeface is sans-serif?” Unlike Bryant or Arial, sans serif is not a specific typeface, but a *generic font family*. There are five generic font families built into the CSS language. These are often used as “an option of last resort” when specifying a list of alternative font values.

The five generic font families are as follows.

## Serif

Serif fonts are characterized by decorative serifs, or accents, at the ends of various letter strokes. Used widely on the Web, serif fonts are sometimes considered to have a classic, formal style. Examples include Times New Roman, Georgia, and Garamond.

## Sans serif

Sans serif translates literally as *without serif*. Unsurprisingly sans serif fonts have simpler forms than serif fonts. Also used widely on the Web, sans serif fonts can be seen as having a clean, modern style. Examples include Helvetica, Arial, and Trebuchet.

## Monospace

As the name implies, the width of the characters in a monospace font—a, b, c . . . , A, B, C . . . , 1, 2, 3 . . .—are all the same. They are most often specified for displaying examples of computer code, where monospaced characters make the code easier to read. Examples include Andale Mono, Courier New, and Monaco.

## Cursive

Cursive fonts emulate handwritten letterforms and have a scriptlike appearance. Their characteristics can vary widely, and most cursive fonts are unlikely to be present on a majority of computers. They should therefore be used with caution. Examples include Bello, Caflisch Script, and Ex Ponto.

## Fantasy

Fantasy fonts (who, we ask, came up with the term *fantasy fonts*?) are primarily decorative and usually intended for headings. The junk-drawer of generic families, fantasy fonts don’t necessarily share many characteristics, and most are unlikely to be present on a majority of computers. Specifying fantasy fonts is an unpredictable affair, perhaps one reason why these fonts aren’t widely used on the Web. Examples include Impact, Critter, and Cottonwood.

Now that we’ve covered the font-family affair in some detail, let’s move swiftly on and talk about another important issue for many: size.

## Size matters

In the last chapter we set a font-size for our paragraphs as follows:

```
p
{
  font-size: 14px;
}
```

thus setting all instances of p at 14px (or 14 pixels), deliberately keeping things simple. When it comes to setting font-size, however, we have a number of options available, including pixels, ems, and keywords. As we mentioned in the introduction, all have strengths; all also have weaknesses. In this section we focus on two units of measure: pixels and ems. Before we get to those, however, first a word on **keywords**.

CSS includes seven font-size keywords, ranging from xx-small to xx-large, which are relative to the browser’s medium setting (where the medium setting is usually interpreted as 16px). In addition to a number of issues in older browsers that require workarounds, the

problem with keywords is their lack of precision. Imagine you'd like to buy a T-shirt, `xx-large` is a little too big, but `x-large` is just a little too small. Herein lies the problem. That said, if you don't care exactly what size your T-shirt is, maybe keywords are for you . . .

## Sizing text with pixels

Sizing your type using pixels is perhaps the easiest method, which explains why we've used it so far; however, it too has its limitations. If you want your paragraphs to appear at a size of 14px, you simply write a CSS rule as we did in the last chapter:

```
p
{
  font-size: 14px;
}
```

What if you'd prefer them at 18px? Simply rewrite the value:

```
p
{
  font-size: 18px;
}
```

Sizing text in pixels allows you to get consistent font sizing in your web pages without trouble. It's also quite easy to grasp the relationship between different font sizes on your web pages: 10px will be half the size of 20px and so on. Great news, but there's a catch.

The main drawback with sizing text in pixels lies with the issue of accessibility. Internet Explorer 6—rapidly dropping in popularity, but still a widely installed browser—is not capable of resizing text specified in pixels. Where other browsers allow the user to resize text set in pixels, IE 6 has no method of doing so. For people who like their text to be larger (or indeed smaller) than the designer specified in the style sheet, IE 6 and pixels are not a winning combination.

IE 7 goes some way toward resolving this with a feature called **Page Zoom**, which enlarges the entire web page, including images. However, this behavior quickly becomes unwieldy, leading to horizontal scrollbars as your page zooms up. This is better than nothing, but it's not ideal.

If accessibility and allowing the visitors of your web site to resize the text to a size that suits them is important to you, consider ems or keywords instead.

## Sizing text with ems

So by now you're really hoping that ems are the Holy Grail. Not quite. (Honestly, are you surprised?) While it's true that ems have a lot going for them, they also have some drawbacks (not least the need to get your head around some, at times complicated, math).

An em, not to be confused with the HTML element `em`, is a relative measure used when sizing type. Ems are *calculated based on the font size of the parent element*. What does that mean in English? Let's look at an example:

```
<body>
  <p><strong>King Kong</strong> is the name of a fictional giant ape
    from the legendary Skull Island.</p>
</body>
```

Imagine you'd like the paragraph in this example to appear at a size equivalent to 14 pixels. You first need to determine the font size of the *parent* element of the `p`. Now you're thinking, "The parent element? What's that again?"

There's no need to panic; when we introduced the concept of inheritance earlier in this chapter, you met parent and child elements. The parent element of the `p` in the preceding example is the `body`, which means we need to establish a size for the `body` and the `p` will then be sized relative to that.

If you have a degree in astrophysics, great. If not, brace yourself for a little mathematics.

Most modern browsers have a default paragraph text size that is 16px, that is, unstyled `p` elements will display at 16px using the browser's default style sheet.

By adding the following rule to a style sheet, we can change the default font size of the page from 16px to anything we'd like by changing the percentage. In the following example we set the default font-size on the `body` to 62.5%.

```
body
{
  font-size: 62.5%;
}
```

Why 62.5%?

62.5% of 16 pixels is 10 pixels. 10 is a nice round number. Setting the base font-size of the `body` to 10px makes working out relative sizes considerably easier. Using the declaration `font-size: 62.5%`; sets 1em at 10px. We'd like our paragraphs to display at 14 px. We now size them *in relation to* the `body` element (1em = 10px, therefore 1.4em = 14px) using the following rules:

```
body
{
  font-size: 62.5%;
}

p
{
  font-size: 1.4em;
}
```

Let's recap because it's a little convoluted! We've reduced the font size of the body element to 10px in our first rule using a percentage of 62.5% (remember, most browsers have a default font-size of 16px and  $62.5\% \times 16\text{px} = 10\text{px}$ ). Everything will be sized relative to the body, which is the *parent* element of everything on the page. The second rule is setting the paragraph to 1.4 times the size of the parent element, or 14px. *Voilà!*

Although this is not as straightforward as setting sizes directly in pixels, this method has the advantage of allowing users to resize their text in any relatively modern browser, including IE 6. For accessibility purposes, this is a good thing to strive for.

*If you're wondering why the font-size in the body element is specified using a percentage instead of using 0.625em, which **should** have the same effect, congratulations, you've just earned five extra nerd points! The answer is that using a percentage works around a bug in IE where the text would resize too much or too little, resulting in super-large or super-tiny text. Redeem your nerd points at the Web Standardistas web site.*

To make the examples in the rest of this chapter easier to follow, we're using pixels from this point on. We'll leave experimenting with the use of ems or keywords as an exercise for the reader.

## Writing more efficient rules

Consider the following example, which sets the font-family, respective font-size for our h1–h4 elements, and colors them #9CC4E5 (a light shade of powder blue). We could write the h1–h4 declarations like this:

```
h1
{
font-family: Arial, sans-serif;
font-size: 36px;
color: #9CC4E5;
}

h2
{
font-family: Arial, sans-serif;
font-size: 24px;
color: #9CC4E5;
}

h3
{
font-family: Arial, sans-serif;
font-size: 18px;
color: #9CC4E5;
}
```

```

h4
{
font-family: Arial, sans-serif;
font-size: 14px;
color: #9CC4E5;
}

```

In this example each of the rules specifies *identical* font-family and color declarations. If we wanted to change these details down the line, we would have to edit each individual rule. This repetitive task seems somewhat inefficient—surely there's a better solution.

The good news is that CSS allows us a way to group selectors to keep our style sheets leaner and meaner. If we replace the preceding rules with the following, the result will be *exactly the same*:

```

h1, h2, h3, h4
{
font-family: Arial, sans-serif;
color: #9CC4E5;
}

h1
{
font-size: 36px;
}

h2
{
font-size: 24px;
}

h3
{
font-size: 18px;
}

h4
{
font-size: 14px;
}

```

Let's take a look at what's happening here. The first rule is instructing the browser to style our h1–h4 headings in Arial (or a generic sans serif fallback font) and in the color #9CC4E5. We've then written rules for each of the h1–h4 headings that set their size, an additional rule specific to each heading.

The benefits of this are twofold. First, the second set of rules is considerably smaller than the first, and we know that the smaller the file, the faster the download. Second, if we'd like to change the color or the font of our h1, h2, h3, and h4 elements down the line, we can do it in just one location, a more efficient method of making changes.



An understanding of grouped selectors coupled with a knowledge of the principles of inheritance, outlined earlier in this chapter, can go a long way to reducing the size and complexity of your style sheets.

## Show and tell: Adding a few more rules

By now you should be quite familiar with adding CSS rules to your web pages, and you should be beginning to grasp some of the principles of CSS. We restricted ourselves in the last chapter to a limited number of CSS properties. We'll introduce a few more in this section to ensure you're getting your money's worth. Value for money is, after all, what it's all about.

Now that we've introduced you to some of the practicalities of handling typography on the Web, we'll finish the process of styling the remainder of the typographic elements on our King Kong page. The process of doing this will introduce you to a variety of CSS properties that you can use to control the look and feel of your web pages' typography.

You can see the effect of all of this chapter's changes combined in our King Kong page, which you'll be referring to as you embark on this chapter's homework: adding additional style to Gordo's page. You can access this page here:

[www.webstandardistas.com/09/king\\_kong.html](http://www.webstandardistas.com/09/king_kong.html)

Without further ado, let's get started on our typographic journey.

## Specifying a typeface

You now know that you can specify a set of typefaces to ensure your typeface selections are more reliable, using alternatives and generic font family names to create a controlled fallback situation in the event that your users don't have your first choice of typeface.

We'd like to display all of the type on our King Kong page in Lucida Grande. A humanist sans serif typeface included with Apple's Mac OS X operating system, we know this won't be available on computers running Windows or Linux by default, so we'll need to specify some fallback fonts for users on other platforms.

We add the following rules to our CSS body declaration, setting the typeface and establishing a base type size for all of the typographic elements on the page. (Remember, elements like our headings will need to be given explicit sizes to stop the browser's default style sheet from overriding our 14px declaration.) We amend the following declarations to our body rule:

```
body
{
  font-family: 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;
  font-size: 14px;
  ...
}
```

This sets all of the type on the page to display in Lucida Grande. Users who don’t have that typeface installed will get Lucida Sans if it’s installed, failing that Arial (a Core Web Font). If that’s not available, they’ll get their browser’s default sans serif.

*Astute readers will notice that both Lucida Grande and Lucida Sans have quotation marks around them. Why is this? The reason is simple: both typefaces’ names consist of more than one word and contain a space. Any font with more than one word and a space—for example, Times New Roman or Courier New—should be enclosed in straight quotes. Note also that the comma needs to follow the closing quotation mark.*

*Quotation marks must not be used to enclose generic font-family names (serif, sans-serif, monospace, cursive, and fantasy).*

You can see the effect of this rule on our King Kong page by following the URL we listed at the start of this section.

## Let’s lose some weight

As you briefly saw earlier in the chapter when we looked at the issue of specificity, when we mark up headings, the browser’s built-in style sheet sets headings to display in bold by default. If we’d like our headings to display in something other than the default weight, we need to explicitly set it using a rule in CSS. We do this using the `font-weight` property.

We’d like to style all of our headings from h1 to h4 to display in a normal weight (or, if you were a typographer, a *roman* weight). We can do this using one rule, a grouped selector, which takes care of all of our headings by explicitly setting a `font-weight`. We add the following rule to our style sheet:

```
h1, h2, h3, h4
{
  font-weight: normal;
  ...
}
```

The result of this one rule is that *all* of our headings display in a roman weight. In Figure 9-7 we show the headings before and after the change for comparison.

Using the `font-weight` property we can set a variety of weights for our type either numerically (100, 200, 300 . . .) or using keywords. The numeric value 400 is equivalent to the keyword `normal`; 700 is equivalent to `bold`.



**Figure 9-7.** Our single rule, using a grouped selector, takes care of all of our headings. On the left our headings as displayed using the browser's default style sheet's font-weight, on the right as they are now styled with our new rule.

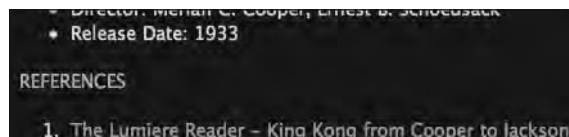
## Text transform

CSS offers us a variety of powerful methods of altering the text that's included in our markup, enabling us to transform the case in which it's set, regardless of how it appears in our markup. We can use the `text-transform` property in CSS to visually transform the display of text into uppercase or lowercase, or even to capitalize words using the value `capitalize`.

In this section we'll use a rule to transform the `h4`, which sits above our list of references, into uppercase to differentiate it from the other headings on the page. We add the following declaration to our `h4` rule:

```
h4
{
text-transform: uppercase;
...
}
```

The result of this declaration is shown in Figure 9-8.



**Figure 9-8.** Using the `text-transform` property in CSS, we can magically transform the visual display of our `h4` text into uppercase.

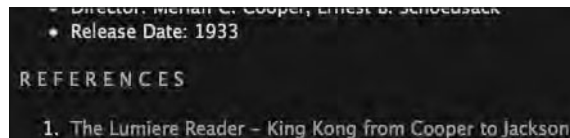
## Letter spacing

Looking at Figure 9-8, the spacing between the uppercase letters is a little tight; they would look better with a little space inserted between them (a property known to typographers as **kerning**). No problem! CSS provides us with a means of controlling kerning using the letter-spacing property.

We increase the letter spacing of our h4 heading by adding the following letter-spacing declaration to our h4 rule:

```
h4
{
text-transform: uppercase;
letter-spacing: 0.4em;
...
}
```

The result of this is shown in Figure 9-9.



**Figure 9-9.** Our h4 type is now letter-spaced a little more generously.

It’s worth noting that we’ve specified our letter-spacing in ems to ensure that the amount of kerning is applied in proportion to the text size of the h4.

A word of warning: letter spacing—while useful when styling headings in uppercase letters—is not recommended for lowercase letters. Letter spacing lowercase letters can impede legibility and, unless absolutely necessary, should be avoided.

*The prolific American type designer Frederick W. Goudy went so far as to state, “A man who would letterspace lower case would steal sheep.” Trust us, at the time, this was quite an insult. Stealing sheep? Who would even countenance the thought . . . This quote went on to become the title of noted typographer Erik Spiekermann’s excellent book Stop Stealing Sheep & Find Out How Type Works (Adobe Press, 1993), which is well worth reading for a comprehensive introduction to the wonderful world of typography.*

When setting letter-spacing using CSS, you’re not restricted to positive letter spacing values; you can also set negative values, for example, letter-spacing: -0.5em;, although why you might want all of your letterforms to overlap each other to create an illegible mess is beyond us.

CSS also allows you to control the space between words using the word-spacing property as in the following example:

```
p
{
  word-spacing: 2em;
  ...
}
```

We leave it as an exercise for you to add this declaration to your Gordo web page and witness why word-spacing should best be avoided like the plague.

## Styling paragraphs

Our page is beginning to take shape. In this section we're going to look at some CSS rules to give our paragraphs some style. We'll introduce some vertical space between the lines within our paragraphs, using a property known as `line-height` to improve the legibility of our paragraphs.

*In CSS the vertical space between lines of text is called `line-height`; in traditional typographic terms this would have been known as **leading**. In the early days of typography, strips of lead would have been inserted between lines of text to space them apart, which is where the term leading originates.*

Once we've set a `line-height`, we'll look at the `text-indent` property to create indents for the first lines of each of our paragraphs. Finally, we'll look at the `text-align` selector to align our text to both the left and right, to center it, and to justify it.

### Setting a line height

Before we embark on adding some line height, let's regroup. The `font-size` of our `p` elements is 14px (remember, we set it on the body element, and the `p` element has inherited this size). However, at present the `line-height` of all the elements on the page is being dictated by the browser's default style sheet as shown in Figure 9-10.

King Kong is the name of a *fictional giant ape* from the legendary Skull Island, who has appeared in several works since 1933. Most of these bear his name, and include the groundbreaking 1933 film, the film remakes of 1976 and 2005, and numerous sequels.

In the original film, the character's name is Kong – a name given to him by the inhabitants of Skull Island, where Kong lived along with a number of dinosaurs which miraculously escaped extinction. 'King' is an appellation added by an American film crew led by Carl Denham, who captures Kong and takes him to New York City to be exhibited as the 'Eighth Wonder of the World'.

**Figure 9-10.** The `line-height` of our paragraphs as it currently stands, styled using the browser's defaults

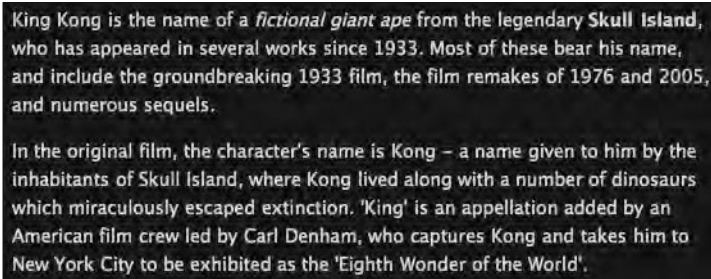
We'd like to give our `p` elements some additional `line-height`, however, to space the lines out a little bit more and improve their legibility. We do this by adding a declaration to the rule targeting our `p` elements as follows:

```

p
{
  line-height: 1.6;
  ...
}

```

The result of this is shown in Figure 9-11.



**Figure 9-11.** The line-height of our paragraphs set to a value of 1.6— $1.6 \times 14$  px (the paragraph text size)

As you can see, adding the line-height declaration adds a little more vertical space between the lines of our paragraphs, improving their legibility. We can set as much line-height (or as little) as we'd like; however, it's worth bearing in mind that too much (or too little) space between the lines of paragraphs can impede legibility. While we recommend the addition of a little line-height to “loosen the text up a little” when setting large paragraphs of type, we suggest—like extra hot chili powder—it be used sparingly.

When specifying a line-height, we're not using a unit of measure (like px or em); instead we're leaving the line-height value unitless. This way the line-height remains consistent throughout your page. Let's explain this by looking at a simple example:

```

body
{
  font-size: 10px;
  line-height: 1.5em;
}

p
{
  font-size: 20px;
}

```

To make the calculations easier to follow, we've set a font-size of 10px on the body and 20px on our p element. We've set the line-height to  $1.5em \times 10px$ , or 15px for the body. The descendent p element will inherit this calculated line-height of 15px.

However, most of the time we'd like the `line-height` to be consistent across all of our elements. In our example we'd like the `line-height` of the `p` element to be  $1.5 \times 20\text{px}$ , or `30px`. The simplest way to achieve this is to remove the unit of measure from our `line-height` declaration in the body rule as follows:

```
body
{
  font-size: 10px;
  line-height: 1.5;
}

p
{
  font-size: 20px;
}
```

This will achieve our goal of creating a `line-height` that is *1.5 times* the `font-size` of each descendant element on our page. The rule of thumb is this: when specifying `line-height`, leave the unit of measure off, and your CSS will work as expected.

## Adding paragraph indents

Now that we've increased the `line-height` of our paragraphs, let's take a look at another property that CSS allows us to control: `text-indent`.

As you discovered in Chapter 3, paragraphs are block-level elements, separated by line breaks. As such, the beginnings and endings of paragraphs are easily identified thanks to the vertical space added beneath `p` elements by default in most web browsers.

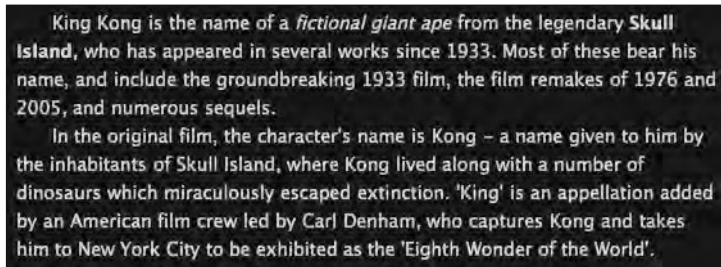
In Chapter 10 we'll show you how to reset the vertical space (a property known as `margin`) that sits between paragraphs, removing the blank lines that separate the paragraphs as they stand on our current King Kong page. However, we'll need to replace this vertical space with something to indicate to the reader that a new paragraph is beginning.

Most novels use indentation to indicate the beginning of each new paragraph within a block of continuous text. We can achieve the same visual effect using the `text-indent` property in CSS.

Adding the following declaration to a rule styling paragraphs indents the first line of every paragraph by 2 ems:

```
p
{
  text-indent: 2em;
  ...
}
```

The result of this rule can be seen in Figure 9-12.



King Kong is the name of a *fictional giant ape* from the legendary Skull Island, who has appeared in several works since 1933. Most of these bear his name, and include the groundbreaking 1933 film, the film remakes of 1976 and 2005, and numerous sequels.

In the original film, the character's name is Kong – a name given to him by the inhabitants of Skull Island, where Kong lived along with a number of dinosaurs which miraculously escaped extinction. 'King' is an appellation added by an American film crew led by Carl Denham, who captures Kong and takes him to New York City to be exhibited as the 'Eighth Wonder of the World'.

**Figure 9-12.** Using text-indent allows us to indent the initial lines of our paragraphs.

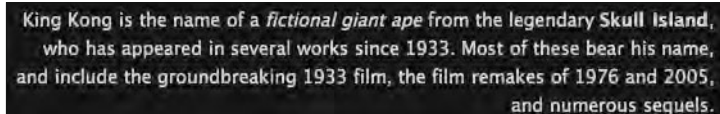
## Aligning text using text-align

Our last focus for this section is the question of aligning type. In CSS type can be aligned using the following four principle settings: left, right, center, and justify. The first three are fairly self-explanatory; the fourth comes with a note of caution.

You've seen left-aligned text already on the King Kong page we've been building, as browsers align text by default to the left. With no text-align specified, the browser's default (in the Western world) is text-align: left; and the preceding paragraph examples are all left-aligned. However, we can align text to the right by using a text-align declaration, as added in the following example:

```
p
{
text-align: right;
...
}
```

You can see the result of this declaration in Figure 9-13.



King Kong is the name of a *fictional giant ape* from the legendary Skull Island, who has appeared in several works since 1933. Most of these bear his name, and include the groundbreaking 1933 film, the film remakes of 1976 and 2005, and numerous sequels.

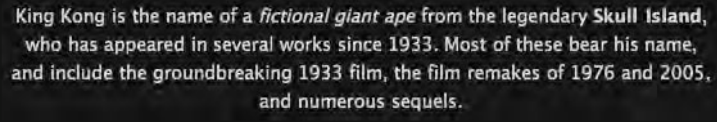
**Figure 9-13.** Our paragraph is now aligned to the right.

We can also center our text using the center value as in the following example:

```
p
{
text-align: center;
...
}
```

You can see the result of this declaration in Figure 9-14.





King Kong is the name of a *fictional giant ape* from the legendary Skull Island, who has appeared in several works since 1933. Most of these bear his name, and include the groundbreaking 1933 film, the film remakes of 1976 and 2005, and numerous sequels.

**Figure 9-14.** Our paragraph is now centered. This can be useful to differentiate type in headings or for centering footer text, for example.

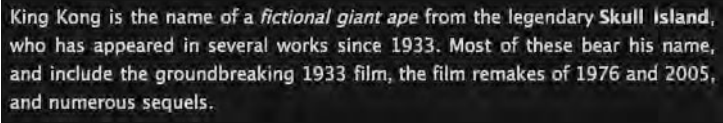
The last text-align value we'll consider is the justify value as in the following example:

```

p
{
  text-align: justify;
  ...
}

```

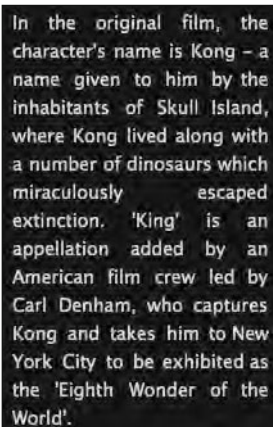
You can see the result of this declaration in Figure 9-15.



King Kong is the name of a *fictional giant ape* from the legendary Skull Island, who has appeared in several works since 1933. Most of these bear his name, and include the groundbreaking 1933 film, the film remakes of 1976 and 2005, and numerous sequels.

**Figure 9-15.** Our paragraph is now justified. The browser adjusts both letter spacing and word spacing to justify the text.

Danger, Will Robinson! In certain situations `text-align: justify;` can really add to a design. However, a note of caution: when text is justified in short paragraphs, it can result in large gaps or “rivers” of space between words, for example, between the words *miraculously* and *escaped* in Figure 9-16.



In the original film, the character's name is Kong – a name given to him by the inhabitants of Skull Island, where Kong lived along with a number of dinosaurs which miraculously escaped extinction. 'King' is an appellation added by an American film crew led by Carl Denham, who captures Kong and takes him to New York City to be exhibited as the 'Eighth Wonder of the World'.

**Figure 9-16.**

When used on paragraphs with a narrow measure, justification results in unsightly rivers running through the text.

Justifying text in print is considerably easier than justifying it online. Justifying text in print often involves the use of hyphenation to even out the different lines in a block of text as much as possible, reducing the possibility of rivers of white space. Online, however, where text can reflow as a user increases and decreases the size of their type, this isn't possible.

## Styling links

The last typographic element on our King Kong page we need to style are its links. Before we get down to some examples, first a word on styling links in general. How should links be styled? Specifically, *to underline or not to underline?*

There are many different opinions on this, and many designers have created great-looking links without an underline in sight. Be aware though that users usually expect links to be underlined (and expect underlined text to be links). However, this isn't to say that links *have* to be underlined. Should you choose to style your links differently, it's a good idea to ensure that they stand out from the body text.

CSS allows us to style links so that they react to a user's interaction with the link itself. We do this by targeting a number of *pseudo-classes*, which relate to the different states a link can be in. These pseudo-classes are as follows:

- **link**: This is the default state for all unvisited links; left unstyled, this is usually blue and underlined.
- **visited**: This is the state for all visited links; left unstyled, this is usually purple and underlined.
- **hover**: This is used to identify when a user is *hovering over* a link (i.e., the user's mouse is positioned over the link).
- **active**: This is used to identify when a user is activating a link or actually in the process of clicking it.
- **focus**: This is used to identify when a link has focus, for instance, when a user tabs to the link using the keyboard.

We can style these pseudo-classes to give our users more visual feedback when they interact with links, a topic we cover in the next section.

## Using pseudo-classes to style links

We'll introduce `classes` in a following chapter; for now we'll look at styling your links using the different link pseudo-classes to give your users a richer experience as they interact with your web pages.

Links can be styled at the most basic level through the inclusion of a simple rule in your style sheet, as in the following example:

```
a
{
  color: red;
}
```

This rule targets all instances of the `a` element (our anchors) and overrides the browser's default blue for links, replacing it with a bright red. So far, so good, but not exactly super-exciting. Let's look at links in a little more detail by styling the links in the references section of our King Kong page.

Our page currently stands as shown in Figure 9-17; the links, in the default blue, are very hard to see due to a lack of contrast between the link color and the page's background color.



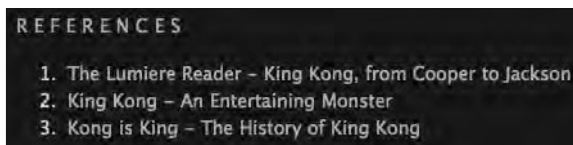
**Figure 9-17.** Unstyled, the default dark blue color of our links is hard to read, lacking contrast with the page's background.

We resolve this by setting the links to display in the same powder blue color as our headings, which, being higher contrast, improves legibility. We also switch off the browser's default underlines by setting the `text-decoration` property of our links to `none`, removing the underline (we'll add this later using a border). Although this change might seem counterintuitive—switching off an underline to replace it with a border—the border will allow us much more scope for styling as you'll see shortly. Lastly, we give the links their own `line-height` to space them out a little more.

We add the following CSS rule:

```
a
{
  color: #9CC4E5;
  text-decoration: none;
  line-height: 1.5;
}
```

The result of this rule can be seen in Figure 9-18.



**Figure 9-18.** Our links' color is now set to match the powder blue of our headings. We've also switched off the underlining.

Now our links are styled so they pick up the powder blue theme we've been developing, improving the consistency of our design. However, we switched off the underline in the last stage by setting the `text-decoration` property to `none`. This could be confusing to our users who might not realize the references are, in fact, links.

We add a border to our `:link` pseudo-class to switch the underline back on. Why do this? Simply because the border gives us much more flexibility. We've set the border to be `1px`, `solid`, and powder blue (`#9CC4E5`). By simply adjusting this declaration, we could increase the weight of our underline to, for example, `5px`; we could also use `dotted` or `dashed` instead of `solid` to create, you guessed it, dotted and dashed underlines. We add the following rule to our style sheet:

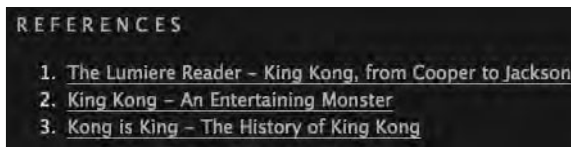
```

a
{
color: #9CC4E5;
text-decoration: none;
line-height: 1.5;
}

a:link
{
border-bottom: 1px solid #9CC4E5;
}

```

The result of this rule is shown in Figure 9-19.



**Figure 9-19.** Our links are now underlined, this time using border-bottom.

By using a border we have much more control over our link's underlines, allowing us to set the border, for example, to dotted, dashed, solid, double, or groove. We can also accurately control the position of the underline in relation to the link text using padding, as you'll see in Chapter 10.

Now that we've styled our `:link` state, we'll look at styling the `:visited` state. Giving it a different style to our `:link` state will visually inform the user which links they've visited and which they haven't. We add the following rule:

```

a
{
color: #9CC4E5;
text-decoration: none;
line-height: 1.5;
}

a:link
{
border-bottom: 1px solid #9CC4E5;
}

a:visited
{
color: #E0DFDA;
border-bottom: 1px dotted #E0DFDA;
}

```

We've used a dotted border so you can see its effect; we've also set the type and the underline to the same color as the body text on the page, differentiating our visited links from our unvisited links. The result of this can be seen in Figure 9-20.



**Figure 9-20.** Our visited links are now differentiated from our unvisited links.

Lastly, we style our `:hover` and `:active` states to provide some interactivity when the user hovers over the links or clicks them. We do this by adding two further rules to our style sheet:

```
a
{
color: #9CC4E5;
text-decoration: none;
line-height: 1.5;
}

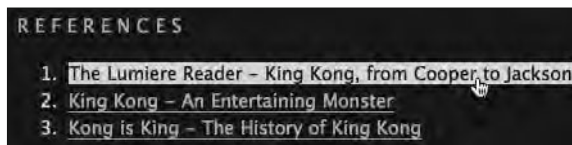
a:link
{
border-bottom: 1px solid #9CC4E5;
}

a:visited
{
color: #E0DFDA;
border-bottom: 1px dotted #E0DFDA;
}

a:hover
{
color: #26201C;
background-color: #E0DFDA;
border-bottom: none;
}

a:active
{
color: #26201C;
background-color: #9CC4E5;
border-bottom: none;
}
```

As you can see in Figure 9-21, when the user mouses over a link, the background color changes, giving a clear visual indication that the text is a link.



**Figure 9-21.** Setting a `:hover` state provides useful visual feedback to the user.

You can see everything we've covered in this section in action at the King Kong web page at the book's companion web site:

[www.webstandardistas.com/09/king\\_kong.html](http://www.webstandardistas.com/09/king_kong.html)

*To ensure your links are more accessible, it's advisable to include another, lesser-known pseudo-class: `a:focus`. This pseudo-class is useful for people who might not use a mouse, instead perhaps using their keyboard to navigate through the links on a page. In most instances `a:focus` can be grouped with the `a:hover` pseudo-class, using a grouped selector, to easily increase the accessibility of your web page.*

We've only scratched the surface of what's possible when styling links. The best way to get a feel for what's possible is to experiment by adding rules to your own XHTML pages and experimenting with changing your own link styles. Trial and error along with some good, old-fashioned use of View Source to look at how links are styled at other web sites will give you lots of inspiration.

## LoVe HAte your links

In the last section we added a full set of rules for our links in all of their states (link, visited, hover, and active). It's important to point out that these pseudo-classes need to be written in a particular order in our CSS to behave properly so that one link pseudo-class doesn't override another. This order is as follows:

- `a:link` (L)
- `a:visited` (V)
- `a:hover` (H)
- `a:active` (A)

An easy way to remember the right order—LVHA—is with the mnemonic: **LoVe HAte**. (Or you could use noted Standardista Dan Cederholm's mnemonic **Love Vegetables? Have Asparagus!**)

## Summary

So what have we covered? This chapter has been a bit of a rollercoaster. We've discussed a great deal, and we'd strongly recommend you read it again, trying out some of the examples as you read along.

We covered some more complicated, but important, aspects of CSS including inheritance and specificity. We also looked at the use of grouped selectors in CSS and how their use can keep our code lean and mean and, as a consequence, easier to maintain as we move forward.

Along the way we looked at typography and how you can use CSS to control it within your documents. In our walkthrough we showed you a wide variety of ways to create great-looking text online. Finally, we explored using pseudo-classes to give a degree of interactivity and user feedback to our links. As with the preceding chapters, follow along using your homework files, comparing them to our examples, and your grasp of styling text with CSS should improve considerably.

In the next chapter we'll introduce the fundamental principles of CSS layout, enabling you to control the position of the elements on your web pages.

## Homework: Improving Gordo's typography

In this chapter we introduced you to a variety of methods of styling text using CSS, showcasing the power and flexibility of style sheets when used for handling typography. The topics we covered will enable you to take your Gordo page's typography and raise it to the next level.

Our focus throughout the chapter was on type, and we introduced you to a number of different properties perfect for styling text using CSS, including font-weight, text-transform, letter-spacing and word-spacing, line-height, text-indent, and text-align.

We also introduced the concept of inheritance and specificity, explaining how an understanding of these two principles will enable you to write lean and easy-to-maintain style sheets.

In this chapter's "show and tell" we applied all of the preceding properties to our King Kong page to show them in action. Your homework for this chapter will be to apply some of these properties to your Gordo page, enhancing its typography and improving the styling of its text.

The process of adding these new properties will further underline your knowledge and awareness of CSS and, as a byproduct, introduce you to the power of inheritance and how it can be used to improve your style sheets by creating more efficient rules.

### 1. Change your font-family

In Chapter 8 we showed you how to set a font-family on the body and p elements. In this chapter we'd like you to remove the font-family declaration from the p element of your Gordo page and take care of all of the page's typography by setting a declaration on the body and allowing inheritance to take care of the rest.

It's also time to revisit the choice of typeface you're using. In the last chapter we deliberately kept things simple by specifying two widely installed typefaces: Arial and Verdana. In this chapter, however, we introduced you to the concept of fallback fonts and used them to specify values for our King Kong page's font-family property as follows:

```
font-family: 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;
```

We've left it to your discretion to select a typeface of your own choosing for your Gordo page; however, we'd like you to specify a number of fallback fonts to ensure that the page's typography is considered in the event of your first choice of typeface being unavailable.

Once you've made these changes, save your Gordo page and test their effect in a number of browsers.

### 2. Use grouped selectors

In Chapter 8 we specified the same color for two of our headings (the h1 and h2), writing an identical color declaration for each rule. In this chapter we introduced you to the concept of grouped selectors, which allow you to reduce the number of declarations in your style sheets by targeting more than one element with each CSS rule.

In this chapter we'd like you to style all of your Gordo page's headings from h1 to h4. Set the font-weight of all of your Gordo page's headings to display in normal (overriding the browser's default style sheet bold styling) and set the color of all of your headings to display in the light shade of blue we've been using (#9CC4E5).

Instead of writing duplicate declarations for each heading's rules, use a grouped selector to take care of the common styles that the headings share. You can refer to the example of a grouped selector in this chapter to see how we styled the common declarations on our King Kong page's h1–h4 elements.

### 3. Style the <h4>

Referring to the examples provided in this chapter, style your h4 element—the “References” heading—using the text-transform and letter-spacing properties.

Experiment with some of the values we covered in the chapter; for example, try setting the text-transform value to uppercase, lowercase, and capitalize and testing the effect of these rules in the browser.

We'd also like you to explore the effect of adjusting your letter-spacing values; try 0.4em, 4em, and 40em. Again, test the effect of changing these values in the browser. (Remember our note of caution: use letter-spacing in moderation, especially when letter-spacing lowercase letters. 40em! What were we thinking?)



#### 4. Style the paragraphs

In this chapter we introduced `line-height`, `text-indent`, and `text-align`—three properties that can be used to great effect to style your paragraphs and improve their legibility. Referring to the examples in this chapter, experiment by setting a variety of values for each of these properties and testing their effects in the browser.

#### 5. Style your links

The last set of rules we'd like you to create is targeted at your Gordo page's links. You might recall that, at the end of the last chapter, the links on your Gordo page suffered from a lack of contrast with the `background-color` of the page, displaying dark blue on a dark brown background. Good news, you'll resolve this here.

Taking a look at our examples in this chapter, style your Gordo page's links, improving their usability. Using the pseudo-classes we introduced, experiment with setting styles for the following pseudo-classes: `a:link`, `a:visited`, `a:hover`, and `a:active`.

As usual, to help you with the different stages of this chapter's homework, we've created our own, similarly styled, page about King Kong featuring new, improved typography. You can refer to this, using your browser's *View Source* menu command to see how we've updated our CSS, here:

[www.webstandardistas.com/09/king\\_kong.html](http://www.webstandardistas.com/09/king_kong.html)

Once you've completed Gordo's typographic upgrade, put the kettle on and enjoy a cup of *Ceylon Dimbula Inverness* as you prepare yourself for the next chapter.