

The Essential Guide to CSS and HTML Web Design

Craig Grannell



The Essential Guide to CSS and HTML Web Design

Copyright © 2007 by Craig Grannell

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-907-5

ISBN-10 (pbk): 1-59059-907-1

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

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

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

Lead Editors Chris Mills,
Tom Welsh

Assistant Production Director
Kari Brooks-Copony

Technical Reviewer
David Anderson

Production Editor
Ellie Fountain

Editorial Board
Steve Anglin, Ewan Buckingham,
Gary Cornell, Jonathan Gennick,
Jason Gilmore, Jonathan Hassell,
Matthew Moodie, Jeffrey Pepper,
Ben Renow-Clarke, Dominic Shakeshaft,
Matt Wade, Tom Welsh

Composer
Dina Quan

Artist
April Milne

Proofreader
Nancy Sixsmith

Project Manager
Kylie Johnston

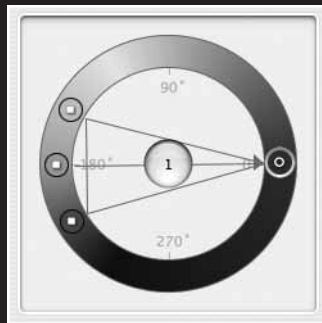
Indexer
Julie Grady

Copy Editor
Damon Larson

Interior and Cover Designer
Kurt Krames

Manufacturing Director
Tom Debolski

4 WORKING WITH IMAGES



In this chapter:

- Understanding color theory
- Choosing the best image format
- Avoiding common mistakes
- Working with images in XHTML
- Using alt text to improve accessibility
- Using CSS when working with images
- Displaying a random image from a selection

Introduction

Although text makes up the bulk of the Web's content, it's inevitable that you'll end up working with images at some point—that is, unless you favor terribly basic websites akin to those last seen in 1995. Images are rife online, comprising the bulk of interfaces, the navigation of millions of sites, and a considerable amount of actual content, too. As the Web continues to barge its way into every facet of life, this trend can only continue; visitors to sites now expect a certain amount of visual interest, just as readers of a magazine expect illustrations or photographs.

Like anything else, use and misuse of images can make or break a website—so, like elsewhere in this book, this chapter covers more than the essentials of working with HTML and CSS. Along with providing an overview of color theory, I've compiled a brief list of common mistakes that people make when working with images for the Web—after all, even the most dedicated web designers pick up bad habits without realizing it. Finally, at the end of the chapter, I'll introduce your first piece of JavaScript, providing you with a handy cut-out-and-keep script to randomize images on a web page.

Color theory

Color plays a massively important role in any field of design, and web design is no exception. Therefore, it seems appropriate to include in this chapter a brief primer on color theory and working with colors on the Web.

Color wheels

Circular color diagrams—commonly referred to as **color wheels**—were invented by Newton and remain a common starting point for creative types wanting to understand the relationship between colors and also for creating color schemes. On any standard color wheel, the three **primary colors** are each placed one-third of the way around the wheel, with **secondary colors** equally spaced between them—secondary colors being a mix of two primary colors. Between secondary and primary colors are **tertiary colors**, the result of mixing primary and secondary colors. Some color wheels blend the colors together, creating a continuous shift from one color to another, while others have rather more defined blocks of color; however, in all cases, the positioning is the same.

Additive and subtractive color systems

4

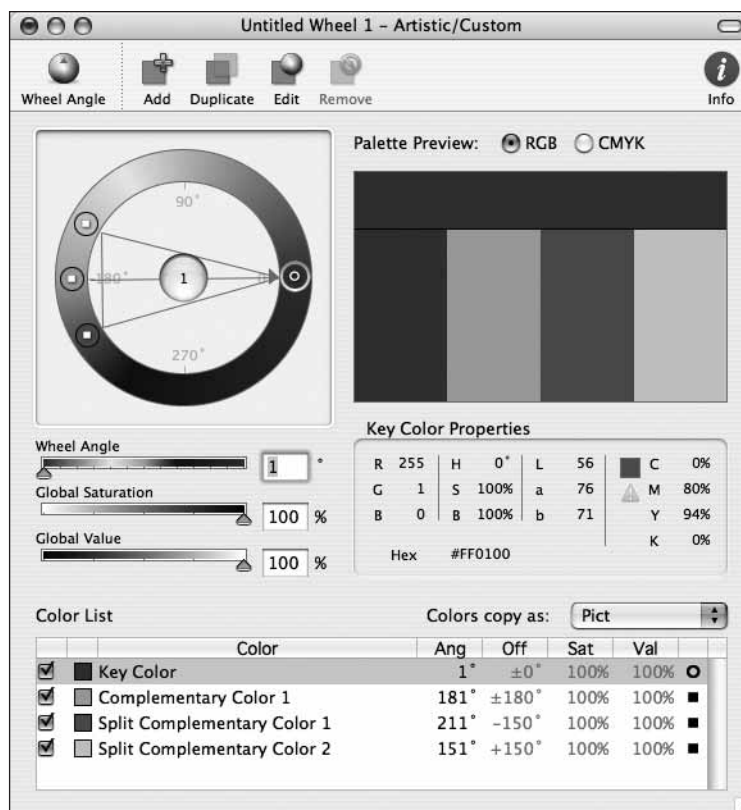
Onscreen colors use what's referred to as an **additive system**, which is the color system used by light—where black is the absence of color, and colored light is added together to create color mixes. The additive primaries are red, green, and blue (hence the commonly heard **RGB** when referring to definition of screen colors). Mix equal amounts of red, green, and blue light and you end up with white; mix secondaries from the primaries and you end up with magenta, yellow, and cyan.

In print, a **subtractive system** is used, similar to that used in the natural world. This works by absorbing colors before they reach the eye—if an object reflects all light it appears white, and if it absorbs all light, it appears black. Inks for print are transparent, acting as filters to enable light to pass through, reflect off the print base (such as paper), and produce unabsorbed light. Typically, the print process uses cyan, magenta, and yellow as primaries, along with a key color—black—since equal combination of three print inks tends to produce a muddy color rather than the black that it should produce in theory.

Although the technology within computers works via an additive system to display colors, digital-based designers still tend to work with subtractive palettes when working on designs (using red, yellow, and blue primaries), because that results in natural color combinations and palettes.

Creating a color scheme using a color wheel

Even if you have a great eye for color and can instinctively create great schemes for websites, it pays to have a color wheel handy. These days, you don't have to rely on reproductions in books or hastily created painted paper wheels. There are now digital color wheels that enable you to experiment with schemes, including Color Consultant Pro for the Mac (www.code-line.com/software/colorconsultantpro.html), shown in the following screenshot, and Color Wheel Pro (www.color-wheel-pro.com) and ColorImpact (www.tigercolor.com/Default.htm), both for Windows.



When working on color schemes and creating a palette for a website, there are various schemes available for you. The simplest is a **monochromatic** scheme, which involves variations in the saturation (effectively the intensity or strength) of a single hue. Such schemes can be soothing—notably when based on green or blue—but also have a tendency to be bland, unless used with striking design and black and white. A slightly richer scheme can be created by using colors adjacent on the color wheel—this is referred to as an **analogous scheme**, and is also typically considered harmonious and pleasing to the eye.

For more impact, a **complementary scheme** can be used, which uses colors on opposite sides of the color wheel (such as red/green, orange/blue, and yellow/purple); this scheme is often seen in art, such as a pointillist using orange dots in areas of blue to add depth. Complementary schemes work well due to a subconscious desire for visual harmony—an equal mix of complementary colors results in a neutral gray. Such effects are apparent in human color vision: if you look at a solid plane of color, you'll see its complementary color when you close your eyes.

A problem with a straight complementary scheme is that overuse of its colors can result in garish, tense design. A subtler but still attention-grabbing scheme can be created by using a color and the hues adjacent to the complementary color. This kind of scheme (which happens to be the one shown in the previous screenshot) is referred to as **split-complementary**.

Another scheme that offers impact—and one often favored by artists—is the **triadic scheme**, which essentially works with primary colors or shifted primaries—that is, colors equally spaced around the color wheel. The scheme provides plenty of visual contrast and, when used with care, can result in a balanced, harmonious result.

How colors “feel” also plays a part in how someone reacts to them—for example, people often talk of “warm” and “cool” colors. Traditionally, cooler colors are said to be passive, blending into backgrounds, while warmer colors are cheerier and welcoming. However, complexity is added by color intensity—a strong blue will appear more prominent than a pale orange. A color’s temperature is also relative, largely defined by what is placed around it. On its own, green is cool, yet it becomes warm when surrounded by blues and purples.

Against black and white, a color’s appearance can also vary. Against white, yellow appears warm, but against black, yellow has an aggressive brilliance. However, blue appears dark on white, but luminescent on black.

The human condition also adds a further wrench in the works. Many colors have cultural significance, whether from language (*cowardly* yellow) or advertising and branding. One person may consider a color one thing (green equals fresh), and another may have different ideas entirely (green equals moldy). There’s also the problem of color blindness, which affects a significant (although primarily male) portion of the population, meaning you should never rely entirely on color to get a message across. Ultimately, stick to the following rules, and you’ll likely have some luck when working on color schemes:

- Work with a color wheel, and be mindful of how different schemes work.
- Use tints and shades of a hue, but generally avoid entirely monochromatic schemes—inject an adjacent color for added interest.
- Create contrast by adding a complementary color.
- Keep saturation levels and value levels the same throughout the scheme (a color’s **value** increases the closer it is to white).
- Keep things simple—using too many colors results in garish schemes.
- Don’t rely on color to get a message across—if in doubt about the effects of color blindness, test your design with a color blindness simulator application such as Color Oracle (<http://colororacle.cartography.ch/>).
- Go with your gut reaction—feelings play an important part when creating color schemes. What feels right is often a good starting point.

Working with hex

The CSS specifications support just 17 color names: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow. All other colors must be written in another format, such as RGB numbers or percentages—`rgb(255.0.0)` or `rgb(100%,0%,0%)`—or hexadecimal format, which tends to be most popular in online design. Note that to keep things consistent, it actually makes sense to write all colors—even the 17 with supported names—in hex. Colors written in hex

comprise a hash sign followed by six digits. The six digits are comprised of pairs, representing the red, green, and blue color values, respectively:

- `#XXxxxx`: Red color value
- `#xxXXxx`: Green color value
- `#xxxxXX`: Blue color value

Because the hexadecimal system is used, the digits can range in value from 0 to f, with 0 being the lowest value (nothing) and f being the highest. Therefore, if we set the first two digits to full (ff) and the others to 0, we get `#ff0000`, which is the hex color value for red. Likewise, `#00ff00` is green and `#0000ff` is blue.

Of course, there are plenty of potential combinations—16.7 million of them, in fact. Luckily, any half-decent graphics application will do the calculations for you, so you won't have to work out for yourself that black is `#000000` and white is `#ffffff`—just use an application's color picker/eyedropper tool, and it should provide you with the relevant hex value.

When a hex value is made up of three pairs, the values can be abbreviated. For example, the value `#ffaa77` can be written `#fa7`. Some designers swear by this abbreviated form. I tend to use the full six-figure hex value because it keeps things consistent.

Web-safe colors

Modern PCs and Macs come with some reasonable graphics clout, but this wasn't always the case. In fact, many computers still in common use cannot display millions of colors. Back in the 1990s, palette restrictions were even more ferocious, with many computers limited to a paltry 256 colors (8-bit). Microsoft and Apple couldn't agree on which colors to use, hence the creation of the web-safe palette, which comprises just 216 colors that are supposed to work accurately on both platforms without dithering. (For more information about dithering, see the "GIF" section later in this chapter.) Applications such as Photoshop have built-in web-safe palettes, and variations on the palette can be seen at www.visibone.com.

Colors in the web-safe palette are made up of combinations of RGB in 20% increments, and as you might expect, the palette is limited. Also discouraging, in the article "Death of the Websafe Color Palette?" on Webmonkey (www.webmonkey.com/00/37/index2a.html; posted September 6, 2000), David Lehn and Hadley Stern reported that all but 22 of these colors were incorrectly shifted in some way when tested on a variety of platforms and color displays—in other words, only 22 of the web-safe colors are actually totally web-safe.

While the rise of PDAs means that the web-safe palette may make a comeback in specialist circles (although PDAs and even cell phones are increasingly powerful when it comes to graphics), most designers these days ignore it. The majority of people using the Web have displays capable of millions of colors, and almost everyone else can view at least

thousands of colors. Unless you're designing for a very specific audience with known restricted hardware, stick with sRGB (the default color space of the Web—see www.w3.org/Graphics/Color/sRGB) and design in millions of colors. And consider yourself lucky that it's not 1995.

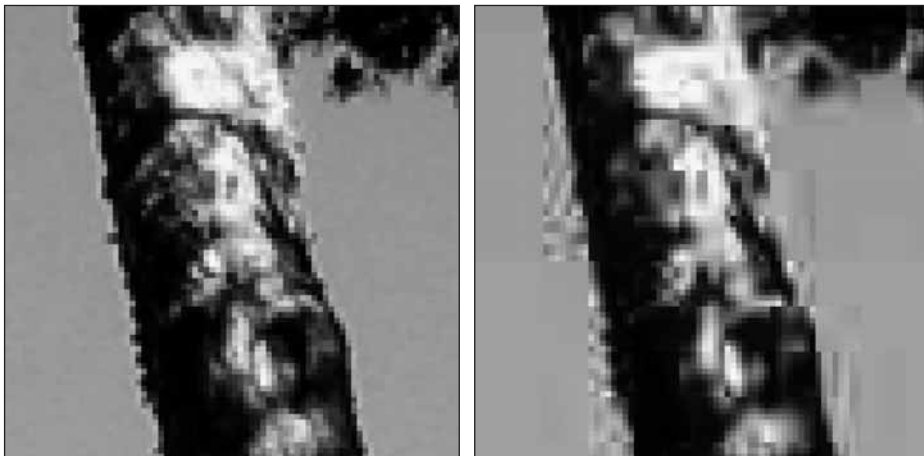
Choosing formats for images

In order to present images online in the best possible way, it's essential to choose the best file format when exporting and saving them. Although the save dialogs in most graphics editors present a bewildering list of possible formats, the Web typically uses just two: JPEG and GIF (along with the GIF89, or transparent GIF, variant), although a third, PNG, is finally gaining popularity, largely due to Internet Explorer 7 finally offering full support for it.

4

JPEG

The JPEG (Joint Photographic Experts Group) format is used primarily for images that require smooth color transitions and continuous tones, such as photographs. JPEG supports millions of colors, and relatively little image detail is lost—at least when compression settings aren't too high. This is because the format uses **lossy compression**, which removes information that the eye doesn't need. As the compression level increases, this information loss becomes increasingly obvious, as shown in the following images. As you can see from the image on the right, which is much more compressed than the one on the left, nasty artifacts become increasingly dominant as the compression level increases. At extreme levels of compression, an image will appear to be composed of linked blocks (see the following two images, the originals of which are in the chapter 4 folder as `tree.jpg` and `tree-compressed.jpg`).



Although it's tricky to define a cutoff point, it's safe to say that for photographic work where it's important to retain quality and detail, 50 to 60% compression (40 to 50% quality) is the highest you should go for. Higher compression is sometimes OK in specific circumstances, such as for very small image thumbnails, but even then, it's best not to go over 70% compression.

If the download time for an image is unacceptably high, you could always try reducing the dimensions rather than the quality—a small, detailed image usually looks better than a large, heavily compressed image. Also, bear in mind that common elements—that is, images that appear on every page of a website, perhaps as part of the interface—will be cached and therefore only need to be downloaded once. Because of this, you can get away with less compression and higher file sizes.

Be aware that applications have different means of referring to compression levels. Some, such as Adobe applications, use a quality scale, in which 100 is uncompressed and 0 is completely compressed. Others, such as Paint Shop Pro, use compression values, in which higher numbers indicate increased compression. Always be sure you know which scale you're using.

Some applications have the option to save progressive JPEGs. Typically, this format results in larger file sizes, but it's useful because it enables your image to download in multiple passes. This means that a low-resolution version will display rapidly and gradually progress to the quality you saved it at, allowing viewers to get a look at a simplified version of the image without having to wait for it to load completely.

GIF

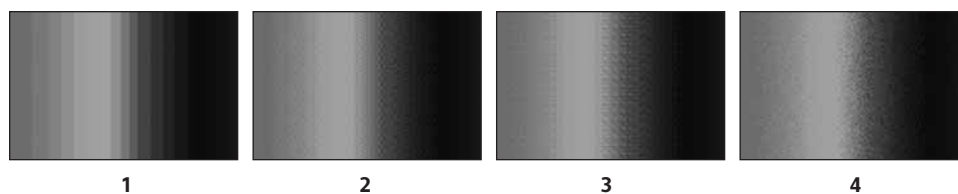
GIF (Graphics Interchange Format) is in many ways the polar opposite of JPEG—it's **lossless**, meaning that there's no color degradation when images are compressed. However, the format is restricted to a maximum of 256 colors, thereby rendering it ineffective for color photographic images. Using GIF for such images tends to produce banding, in which colors are reduced to the nearest equivalent. A fairly extreme example of this is shown in the following illustration.



GIF is useful for displaying images with large areas of flat color, such as logos, line art, and type. As I mentioned in the previous chapter, you should generally avoid using graphics for text on your web pages, but if you do, GIF is the best choice of format.

Although GIF is restricted to 256 colors, it's worth noting that you don't have to use the *same* 256 colors every time. Most graphics applications provide a number of palette options, such as **perceptual**, **selective**, and **Web**. The first of those, perceptual, tends to prioritize colors that the human eye is most sensitive to, thereby providing the best color integrity. Selective works in a similar fashion, but balances its color choices with web-safe colors, thereby creating results more likely to be safe across platforms. Web refers to the 216-color web-safe palette discussed earlier. Additionally, you often have the option to lock colors, which forces your graphics application to use only the colors within the palette you choose.

Images can also be dithered, which prevents continuous tones from becoming bands of color. Dithering simulates continuous tones, using the available (restricted) palette. Most graphics editors allow for three different types of dithering: **diffusion**, **pattern**, and **noise**—all of which have markedly different effects on an image. Diffusion applies a random pattern across adjacent pixels, whereas pattern applies a half-tone pattern rather like that seen in low-quality print publications. Noise works rather like diffusion, but without diffusing the pattern across adjacent pixels. Following are four examples of the effects of dithering on an image that began life as a smooth gradient. The first image (1) has no dither, and the gradient has been turned into a series of solid, vertical stripes. The second image (2) shows the effects of diffusion dithering; the third (3), pattern; and the fourth (4), noise.



GIF89: The transparent GIF

The GIF89 file format is identical to GIF, with one important exception: you can remove colors, which provides a very basic means of transparency and enables the background to show through. Because this is not alpha transparency (a type of transparency that enables a smooth transition from solid to transparent, allowing for many levels of opacity), it doesn't work in the way many graphic designers expect. You cannot, for instance, fade an image's background from color to transparent and expect the web page's background to show through—instead, GIF89's transparency is akin to cutting a hole with a pair of scissors: the background shows through the removed colors only. This is fine when the "hole" has flat horizontal or vertical edges. But if you try this with irregular shapes—such as in the following image of the cloud with drop shadow—you'll end up with ragged edges. In the example, the idea was to have the cloud casting a shadow onto the gray background. However, because GIFs can't deal with alpha transparency, we instead end up with an unwanted white outline. (One way around this is to export the image with the same background color as that of the web page, but this is only possible if the web page's background is a plain, flat color.)

Because of these restrictions, GIF89s are not used all that much these days. They do cling on in one area of web design, though: as spacers for stretching table cells, in order to lay out a page. However, in these enlightened times, that type of technique should be avoided, since you can lay out precisely spaced pages much more easily using CSS.



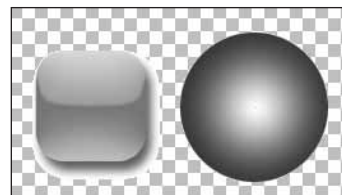
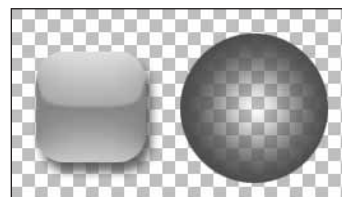
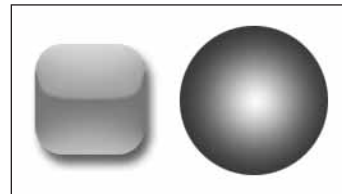
PNG

For years, PNG (pronounced *ping*, and short for Portable Network Graphics) lurked in the wilderness as a capable yet unloved and unused format for web design. Designed primarily as a replacement for GIF, the format has plenty to offer, including a far more flexible palette than GIF and true alpha transparency. Some have mooted PNG as a JPEG replacement, too, but this isn't recommended—PNGs tend to be much larger than JPEGs for photographic images. For imagery with sharp lines, areas of flat color, or where alpha transparency is required, it is, however, a good choice.

The reason PNG is still less common than GIF or JPEG primarily has to do with Internet Explorer. Prior to version 7, Microsoft's browser didn't offer support for PNG alpha transparency, instead replacing transparent areas with white or gray. Although a proprietary workaround exists (see Chapter 9's "Dealing with Internet Explorer bugs" section), it isn't intuitive, and it requires extra code. With post-version 6 releases of Internet Explorer finally supporting alpha transparency (and Internet Explorer's share of the market decreasing somewhat, primarily due to competition from Firefox), it's worth looking into PNG when creating layouts.

The three adjacent images highlight the benefit of PNG over GIF, as shown in a web browser. The first illustration shows two PNGs on a white background. The second illustration shows this background replaced by a grid. Note how the button's drop shadow is partially see-through, while the circle's center is revealed as being partially transparent, increasing in opacity toward its edge. The third illustration shows the closest equivalent when using GIFs—the drop shadow is surrounded by an ugly cutout, and the circle's central area loses its transparency. Upon closer inspection, the circle is also surrounded by a jagged edge, and the colors are far less smooth than those of the PNG.

For more information about this format, check out the PNG website at www.libpng.org/pub/png.



Other image formats

You may have worked on pages in the past and added the odd BMP or TIFF file, or seen another site do the same. These are not standard formats for the Web, though, and while they may work fine in some cases, they require additional software in order to render in some browsers (in many cases, they won't render at all, or they'll render inconsistently across browsers). Furthermore, JPEG, GIF, and PNG are well-suited to web design because

they enable you to present a lot of visual information in a fairly small file. Presenting the same in a TIFF or BMP won't massively increase the image's quality (when taking into account the low resolution of the Web), but it will almost certainly increase download times. Therefore, quite simply, don't use any formats other than JPEG, GIF, or PNG for your web images (and if you decide to use PNG transparency, be sure that your target audience will be able to see the images).

Common web image gaffes

The same mistakes tend to crop up again and again when designers start working with images. In order to avoid making them, read on to find out about some common ones (and how to avoid them).

Using graphics for body copy

Some sites out there use graphics for body copy on web pages, in order to get more typographical control than CSS allows. However, using graphics for body copy causes text to print poorly—much worse than HTML-based text. Additionally, it means the text can't be read by search engines, can't be copied and pasted, and can't be enlarged, unless you're using a browser (or operating system) that can zoom—and even then it will be pixilated. If graphical text needs to be updated, it means reworking the original image (which could include messing with line wraps, if words need to be added or removed), re-exporting it, and reuploading it.

As mentioned in the “Image-replacement techniques” section of Chapter 3, the argument is a little less clear-cut for headings (although I recommend using styled HTML-based text for those, too), but for body copy, you should always avoid using images.

Not working from original images

If it turns out an image on a website is too large or needs editing in some way, the original should be sourced to make any changes if the online version has been in any way compressed. This is because continually saving a compressed image reduces its quality each time. Also, under no circumstances should you increase the dimensions of a compressed JPEG. Doing so leads to abysmal results every time.

Overwriting original documents

The previous problem gets worse if you've deleted your originals. Therefore, be sure that you never overwrite the original files you're using. If resampling JPEGs from a digital camera for the Web, work with copies so you don't accidentally overwrite your only copy of that great photo you've taken with a much smaller, heavily compressed version. More important, if you're using an application that enables layers, save copies of the layered documents prior to flattening them for export—otherwise you'll regret it when having to make that all-important change and having to start from scratch.

Busy backgrounds

When used well, backgrounds can improve a website, adding visual interest and atmosphere—see the following image, showing the top of a version of the Snub Communications homepage. However, if backgrounds are too busy, in terms of complicated artwork and color, they'll distract from the page's content. If placed under text, they may even make your site's text-based content impossible to read. With that in mind, keep any backgrounds behind content subtle—near-transparent single-color watermarks tend to work best.

For backgrounds outside of the content area (as per the “Watermarks” section in Chapter 2), you must take care, too. Find a balance in your design and ensure that the background doesn't distract from the content, which is the most important aspect of the site.

4



Lack of contrast

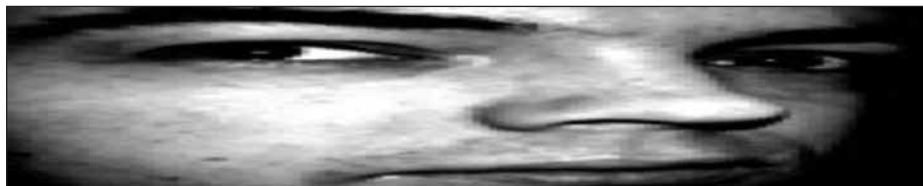
It's common to see websites that don't provide enough contrast between text content and the background—for example, (very) light gray text on a white background, or pale text on an only slightly darker background. Sometimes this lack of contrast finds its way into other elements of the site, such as imagery comprising interface elements. This isn't always a major problem—in some cases, designs look stylish if a subtle scheme is used with care. You should, however, ensure that usability isn't affected—it's all very well to have a subtle color scheme, but not if it stops visitors from being able to easily find things like navigation elements, or from being able to read the text.

Using the wrong image format

Exporting photographs as GIFs, using BMPs or TIFFs online, rendering soft and blotchy line art and text as a result of using the JPEG format—these are all things to avoid in the world of creating images for websites. See the section “Choosing formats for images” earlier in this chapter for an in-depth discussion of formats.

Resizing in HTML

When designers work in WYSIWYG editing tools, relying on a drag-and-drop interface, it's sometimes tempting to resize all elements in this manner (and this can sometimes also be done by accident), thereby compromising the underlying code of a web page. Where images are concerned, this has a detrimental effect, because the pixel dimensions of the image no longer tally with its width and height values. In some cases, this may lead to distorted imagery (as shown in the rather extreme example that follows); it may also lead to visually small images that have ridiculously large files sizes by comparison. In most cases, distortion of detail will still occur, even when proportion is maintained.



There are exceptions to this rule, however, although they are rare. For instance, if you work with pixel art saved as a GIF, you can proportionately enlarge an image, making it large on the screen. Despite the image being large, the file size will be tiny.

Not balancing quality and file size

Bandwidth can be a problem in image-heavy sites—both in terms of the host getting hammered when visitor numbers increase, and in terms of the visitors—many of whom may be stuck with slower connections than you—having to download the images. Therefore, you should always be sure that your images are highly optimized, in order to save on hosting costs and ensure that your website's visitors don't have to suffer massive downloads. (In fact, they probably won't—they'll more than likely go elsewhere.)

But this doesn't mean that you should compress every image on your website into a slushy mess (and I've seen plenty of sites where the creator has exported JPEGs at what looks like 90% compression—"just in case").

Err on the side of caution, but remember: common interface elements are cached, so you can afford to save them at a slightly higher quality. Any image that someone requests (such as via a thumbnail on a portfolio site) is something they *want* to see, so these too can be saved at a higher quality because the person is likely to wait. Also, there is no such thing as an optimum size for web images. If you've read in the past that no web image should ever be larger than 50 KB, it's hogwash. The size of your images depends entirely on context, the type of site you're creating, and the audience you're creating it for.

Text overlays and splitting images

Some designers use various means to stop people from stealing images from their site and reusing them. The most common are including a copyright statement on the image itself, splitting the image into a number of separate images to make it harder to download, and adding an invisible transparent GIF overlay.

The main problem with copyright statements is that they are often poorly realized (see the following example), ruining the image with a garish text overlay. Ultimately, while anyone can download images from your website to their hard drive, you need to remember that if someone uses your images, they're infringing your copyright, and you can deal with them accordingly (and, if they link directly to images on your server, try changing the affected images to something text-based, like "The scumbag whose site you're visiting stole images from me").

4



As for splitting images into several separate files or placing invisible GIFs over images to try to stop people from downloading them, don't do this—there are simple workarounds in either case, and you just end up making things harder for yourself when updating your site. Sometimes you even risk compromising the structural integrity of your site when using such methods.

Stealing images and designs

Too many people appear to think that the Internet is a free-for-all, outside of the usual copyright restrictions, but this isn't the case: copyright exists on the Web just like everywhere else. Unless you have permission to reuse an image you've found online, you shouldn't do so. If discovered, you may get the digital equivalent of a slap on the wrist, but you could also be sued for copyright infringement.

Although it's all right to be influenced by someone else's design, you should also ensure you don't simply rip off a creation found on the Web—otherwise you could end up in legal trouble, or the subject of ridicule as a feature on Tim Murtaugh's Pirated Sites forum (see www.pirated-sites.com/vanilla/).

Working with images in XHTML

The `img` element is used to add images to a web page. It's an empty tag, so it takes the combined start and end tag form with a trailing slash, as outlined in Chapter 1. The following code block shows an example of an image element, complete with relevant attributes:

```

```

Perhaps surprisingly, the height and width attributes are actually optional, although I recommend including them because they assist the browser in determining the size of the image before it downloads (thereby speeding up the process of laying out the page). The only two image element attributes required in XHTML are `src` and `alt`. The first, `src`, is the path to the image file to be displayed; and the second, `alt`, provides some **alternative text** for when the image is not displayed.

Note that this chapter's section on images largely concerns itself with inline images—the addition of images to the content of a web page. For an overview of using images as backgrounds, see the “Web page backgrounds” section of Chapter 2; for an overview of working with images within web navigation and with links in general, see much of Chapter 5.

Using alt text for accessibility benefits

Alternate text—usually referred to as “alt text,” after its attribute—is often ignored or used poorly by designers, but it's essential for improving the accessibility of web pages. Visitors using screen readers rely on the `alt` attribute's value to determine what an image shows. Therefore, always include a *succinct* description of the image's content and avoid using the image's file name, because that's often of little help. Ignoring the `alt` attribute not only renders your page invalid according to the W3C recommendations, but it also means that screen readers (and browsers that cannot display images) end up with something like this for output: `[IMAGE][IMAGE][IMAGE]`—not very helpful, to say the least.

Descriptive alt text for link-based images

Images often take on dual roles, being used for navigation purposes as well as additional visual impact. In such cases, the fact that the image is a navigation aid is likely to be of

more significance than its visual appearance. For instance, many companies use logos as links to a homepage—in such cases, some designers would suggest using “Company X homepage” for the alt text, as it’s more useful than “Company X.”

Alternatively, stick with using the alt attribute for describing the image, and add a title attribute to the link, using that to describe the target. Depending on user settings, the link’s title attribute will be read out in the absence of any link text.

If you don’t have access to screen-reading software for testing alt text and various other accessibility aspects of a website, either install the text-based browser Lynx, or run Opera in User mode, which can emulate a text browser.

Null alt attributes for interface images

In some cases, images have no meaning at all (e.g., if they’re a part of an interface), and there is some debate regarding the best course of action with regard to such images’ alt values. Definitely never type something like `spacer` or `interface element`, otherwise screen readers and text browsers will drive their users crazy relaying these values back to them. Instead, it’s recommended that you use a **null alt attribute**, which takes the form `alt=""`.

Null alt attributes are unfortunately not interpreted correctly by all screen readers; some, upon discovering a null alt attribute, go on to read the image’s `src` value. A common workaround is to use empty alt attributes, which just have blank space for the value (`alt=" "`). However, the null alt attribute has valid semantics, so it should be used despite some screen readers not being able to deal with it correctly.

Alternatively, try reworking your design so that images without meaning are applied as background images to div elements, rather than placed inline.

Using alt and title text for tooltips

Although the W3C specifically states that alt text shouldn’t be visible if the image can be seen, Internet Explorer ignores this, displaying alt text as a tooltip when the mouse cursor hovers over an image, as shown in the adjacent example.

Internet Explorer users are most likely accustomed to this by now, and, indeed, you may have used alt text to create tooltips in your own work. If so, it’s time to stop. This behavior is not recommended by the W3C and it’s also not common across all browsers and platforms.



If an image requires a tooltip, most browsers display the value of a `title` attribute as one. In spite of this, if the text you're intending for a pop-up is important, you should instead place it within the standard text of your web page, rather than hiding it where most users won't see it. This is especially important when you consider that Firefox crops the values after around 80 characters, unlike some browsers, which happily show multiline tooltips.

Another alternative for extended descriptions for images is the `longdesc` attribute. It's not fully supported, but Firefox, SeaMonkey, and Netscape display the attribute's contents as a description field when you view image properties. It's also fully supported in the JAWS screen reader, thereby warranting its use should your image descriptions be lengthy.

Using CSS when working with images

In the following section, we're going to look at relevant CSS for web page images. You'll see how best to apply borders to images and wrap text around them, as well as define spacing between images and other page elements.

Applying CSS borders to images

You may have noticed earlier that I didn't mention the `border` attribute when working through the `img` element. This is because the `border` attribute is deprecated; adding borders to images is best achieved and controlled by using CSS. (Also, because of the flexibility of CSS, this means that if you only want a simple surrounding border composed of flat color, you no longer have to add borders directly to your image files.) Should you want to add a border to every image on your website, you could do so with the following CSS:

```
img {
  border: 1px solid #000000;
}
```

In this case, a 1-pixel solid border, colored black (#000000 in hex), would surround every image on the site. Using contextual selectors, this can be further refined. For instance, should you only want the images within a content area (marked up as a `div` with an `id` value of `content`) to be displayed with a border, you could write the following CSS:

```
div#content img {
  border: 1px solid #000000;
}
```

Alternatively, you could set borders to be on by default, and override them in specific areas of the website via a rule using grouped contextual selectors:

```
img {
  border: 1px solid #000000;
}

#masthead img, #footer img, #sidebar img {
  border: 0;
}
```

Finally, you could override a global border setting by creating a `noBorder` class and then assigning it to relevant images. In CSS, you'd write the following:

```
.noBorder {
  border: 0;
}
```

And in HTML, you'd add the `noBorder` class to any image that you didn't want to have a border:

```

```

Clearly, this could be reversed (turning off borders by default and overriding this with, say, an `addBorder` style that could be used to add borders to specific images). Obviously, you should go for whichever system provides you with the greatest flexibility when it comes to rapidly updating styles across the site and keeping things consistent when any changes occur. Generally, the contextual method is superior for achieving this.

Although it's most common to apply borders using the shorthand shown earlier, it's possible to define borders on a per-side basis, as demonstrated in the "Using classes and CSS overrides to create an alternate pull quote" exercise in Chapter 3. If you wanted to style a specific image to resemble a Polaroid photograph, you could set equal borders on the top, left, and right, and a larger one on the bottom. In HTML, you would add a `class` attribute to the relevant image:

```

```

In CSS, you would write the following:

```
.photo {
  border-width: 8px 8px 20px;
  border-style: solid;
  border-color: #ffffff;
}
```

The results of this are shown in the image to the right. (Obviously, the white border only shows if you have a contrasting background—you wouldn't see a white border on a white background!)

Should you want to, you can also reduce the declaration's size by amalgamating the `border-style` and `border-color` definitions:

```
.photo {
  border: solid #ffffff;
  border-width : 8px 8px 20px;
}
```



Note that when you've used a contextual selector with an id value to style a bunch of elements in context, overriding this often requires the contextual selector to again be included in the override rule. In other words, a class value of `.override` would not necessarily override values set in `#box img`, even if applied to an image in the box div. In such cases, you'd need to add the id to the selector: `#box .override`.

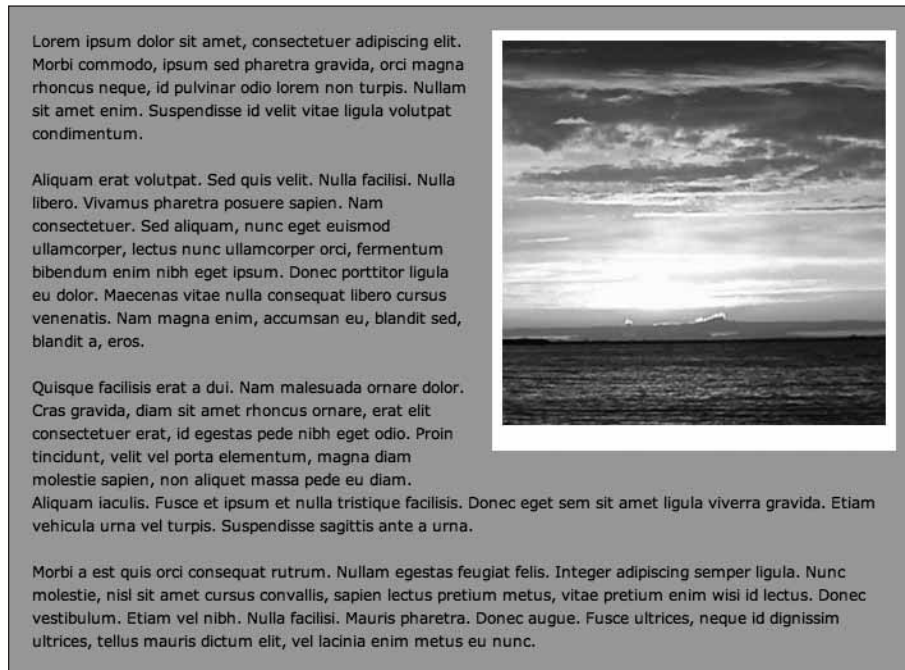
There are other `border-style` values that can be used with images, as well. Examples include dashed and dotted—see the `border-style` entry in Appendix D (CSS Reference) for a full list. However, overdone decoration can distract from the image, so always ensure that your borders don't overpower your imagery.

Using CSS to wrap text around images

You can use the `float` and `margin` properties to enable body copy to wrap around an image. The method is similar to the pull quote example in the previous chapter, so we won't dwell too much on this. Suffice to say that images can be floated left or right, and margins can be set around edges facing body copy in order to provide some whitespace. For example, expanding on the previous example, you could add the following rules to ensure that the surrounding body copy doesn't hug the image:

```
.photo {
  border-width: 8px 8px 20px 8px;
  border-style: solid;
  border-color: #ffffff;
  float: right;
  margin-left: 20px;
  margin-bottom: 20px;
}
```

This results in the following effect shown in the following image.



4

See [using-css-to-wrap-around-images.html](#), [using-css-to-wrap-around-images.css](#), and [sunset.jpg](#) in the chapter 4 folder for a working example of this page.

Displaying random images

This final section of the chapter looks at creating a simple system for displaying a random image from a selection. This has several potential uses, such as randomizing banners on a commercial website, or giving the impression that a site is updated more often than it is by showing visitors some new content each time they arrive. Also, for portfolios, it's useful to present a random piece of work from a selection.

Prior to starting work, you need to prepare your images. Unless you're prepared for subsequent layout elements to shift upon each visit to the page, aim to export all your images with equal dimensions. Should this not be an option, try to keep the same height setting. Note, however, that you can use different file formats for the various images. It's good housekeeping to keep these images in their own folder, too; for this exercise, the images are placed within `assets/random-images`.

Creating a JavaScript-based image randomizer

Required files	The image-randomizer-starting-point folder from the chapter 4 folder.
What you'll learn	How to create an image randomizer using JavaScript.
Completed files	The image-randomizer-javascript folder in the chapter 4 folder.

1. Edit the HTML. Open `randomizer.html`. In the body of the web page, add the following `img` element. The `src` value is for the default image, and this is what's shown if JavaScript is unavailable. The `id` value is important—this is a hook for both the JavaScript function written in steps 4 through 6 and a CSS rule to add a border to the image.

```

```

Next, add an `onload` attribute to the body start tag, as shown in the following code block. Note that the value of this attribute will be the name of the JavaScript function.

```
<body onload="randomImage()">
```

2. In `randomizer.js`, create arrays for image file names and alt attribute values. For the former, only the image file names are needed—not the path to them (that will be added later). Note that the order of the items in the arrays must match—in other words, the text in the first item of the `chosenAltCopy` array should be for the first image in the `chosenImage` array.

```
var chosenImage=new Array();
chosenImage[0]="stream.jpg";
chosenImage[1]="river.jpg";
chosenImage[2]="road.jpg";

var chosenAltCopy=new Array();
chosenAltCopy[0]="A stream in Iceland";
chosenAltCopy[1]="A river in Skaftafell, Iceland";
chosenAltCopy[2]="A near-deserted road in Iceland";
```

3. Create a random value. The following JavaScript provides a random value:

```
var getRan=Math.floor(Math.random()*chosenImage.length);
```

4. Create a function. Add the following text to start writing the JavaScript function, which was earlier dubbed `randomImage` (see step 1's `onload` value). If you're not familiar with JavaScript, then note that content from subsequent steps must be inserted into the space between the curly brackets.

```
function randomImage()
{
}
```

5. Add JavaScript to set the image. By manipulating the **Document Object Model (DOM)**, we can assign values to an element via its id value. Here, the line states to set the `src` attribute value of the element with the id value `randomImage` (i.e., the image added in step 1) to the stated path value plus a random item from the `chosenImage` array (as defined via `getRan`, a variable created in step 3).

```
document.getElementById('randomImage').setAttribute
↳('src', 'assets/random-images/'+chosenImage[getRan]);
```

6. Add JavaScript to set the alt text. Setting the alt text works in a similar way to step 5, but the line is slightly simpler, due to the lack of a path value for the alt text:

```
document.getElementById('randomImage').setAttribute
↳('alt', chosenAltCopy[getRan]);
```

7. Style the image. In CSS, add the following two rules. The first removes borders by default from images that are links. The second defines a border for the image added in step 1, which has an id value of `randomImage`.

```
a img {
  border: 0;
}
#randomImage {
  border: solid 1px #000000;
}
```

Upon testing the completed files in a browser, each refresh should show a random image from the selection, as shown in the following screenshot. (Note that in this image, the padding value for `body` was set to `20px 0 0 20px`, to avoid the random image hugging the top left of the browser window.)



There are a couple of things to note regarding the script. To add further images/alt text, copy the previous items in each array, increment the number in square brackets by one and then amend the values—for example:

```
var chosenImage=new Array();
chosenImage[0]="stream.jpg";
chosenImage[1]="river.jpg";
chosenImage[2]="road.jpg";
chosenImage[3]="harbor.jpg";

var chosenAltCopy=new Array();
chosenAltCopy[0]="A stream in Iceland";
chosenAltCopy[1]="A river in Skaftafell, Iceland";
chosenAltCopy[2]="A near-deserted road in Iceland";
chosenAltCopy[3]="The harbor in Reykjavík ";
```

You'll also note that in this example, the height and widths of the images is identical. However, these can also be changed by editing the script. For example, to set a separate height for each image, you'd first add the following array:

```
var chosenHeight=new Array();
chosenHeight[0]="200";
chosenHeight[1]="500";
chosenHeight[2]="400";
```

And you'd next add the following line to the function:

```
document.getElementById('randomImage').setAttribute
↳('height',chosenHeight[getRan]);
```

Remember, however, the advice earlier about the page reflowing if the image dimensions vary—if you have images of differing sizes, your design will need to take this into account.

Creating a PHP-based image randomizer

Required files	The image-randomizer-starting-point folder from the chapter 4 folder.
What you'll learn	How to create an image randomizer using PHP.
Completed files	The image-randomizer-php folder in the chapter 4 folder.

If you have access to web space that enables you to work with PHP, it's simple to create an equivalent to the JavaScript exercise using PHP. The main benefit is that users who disable JavaScript will still see a random image, rather than just the default. Note that you need some method of running PHP files to work on this exercise, such as a local install of Apache. Note also that prior to working through the steps, you should remove the HTML document's script element, and you should also amend the title element's value, changing it to something more appropriate.

1. Define the CSS rules. In CSS, define a border style, as per step 7 of the previous exercise, but also edit the existing paragraph rule with a font property/value pair, because in this example, you're going to add a caption based on the alt text value.

```
a img {
    border: 0;
}
#randomImage {
    border: solid 1px #000000;
}
p {
    font: 1.2em/1.5em Verdana, sans-serif;
    margin-bottom: 1.5em;
}
```

4

2. Set up the PHP tag. Change the file name of randomizer.html to randomizer.php to make it a PHP document. Then, place the following on the page, in the location where you want the randomized image to go. Subsequent code should be placed within the PHP tags.

```
<?php
?>
```

3. Define the array. One array can be used to hold the information for the file names and alt text. In each case, the alt text should follow its associated image.

```
$picarray = array("stream" => "A photo of a stream", "river" => "A
➤ photo of a river", "road" => "A photo of a road");
$randomkey = array_rand($picarray);
```

4. Print information to the web page. Add the following lines to write the img and p elements to the web page, using a random item set from the array for the relevant attributes. Note that the paragraph content is as per the alt text. Aside from the caption, the resulting web page looks identical to the JavaScript example.

```
echo '';
```

```
echo '<p>' . $picarray[$randomkey] . '</p>';
```

5. Use an include. This is an extra step of sorts. If you want to make your PHP more modular, you can copy everything within the PHP tags to an external document, save it (e.g., as random-image.php) and then cut it into the web page as an include:

```
<?php
@include($_SERVER['DOCUMENT_ROOT'] . "/random-image.php");
?>
```

For more on working with PHP, see PHP Solutions: Dynamic Web Design Made Easy, by David Powers.

Hopefully you've found this chapter of interest and now feel you have a good grounding in working with images on the Web. It's amazing to think how devoid of visual interest the Web used to be in contrast to today, now that images are essential to the vast majority of sites. As I've mentioned before, the importance of images on the Web lies not only in content, but in interface elements as well, such as navigation—a topic we're covering in the next chapter.

