



# Web Designing

## Hand Book

## Contents

Introduction to Web Designing .....	4
CSS (Cascading Style Sheets) .....	4
Server-side programming.....	4
HTML documents .....	6
The Nature of “WEB DESIGN” .....	7
Alternative Browsing Environments.....	8
Users with disabilities .....	8
Browser Window Size and Monitor.....	10
HTML Overview.....	10
Introducing... the HTML element .....	10
Adding Link .....	12
Adding Images .....	14
Lists .....	17
Table Markup .....	18
Forms .....	21
Mailing List Signup .....	21
Form Elements .....	23
Cascading Style Sheet Orientation.....	28
Colors and Backgrounds Specifying Color Values .....	37
Floating and Positioning .....	45
Positioning Basics .....	47
Page layout with css.....	50
Dealing with line lengths .....	52
Generic Elements (div and span) .....	53
Element identifiers .....	54
CSS 3 .....	56
What is CSS 3 ? .....	56
CSS3 Rounded Corners .....	57
CSS3 Border Images.....	57
CSS3 Backgrounds .....	57
Multiple Background Images.....	58
Background Gradients .....	59
Shadows effects .....	59
Transitions.....	60
Transforms .....	60
Animation .....	61
What is Responsive Web Design? .....	62
HTML 5 .....	63
What is HTML 5 ? .....	63
Introduction.....	63
Tools.....	63
New Features .....	64
New tags of HTML 5 .....	64
HTML5 form tag.....	65
HTML5 Input type form tag .....	66
Javascript.....	67
How it Works? .....	67
Add Javascript to HTML Page.....	68
Get Html Element in Javascript .....	70
Javascript with CSS Style .....	71
Javascript Validation .....	71
Jquery.....	75
About Browser Compatibility.....	75
Including the library.....	75

All Selector (“*”).....	77
animated Selector.....	78
Class Selector (“.class”).....	78
Element Selector (“element”).....	79
Empty Selector.....	80
jQuery(‘:empty’).....	80
.addClass().....	80
addClass( className ).....	80
removeClass().....	81
toggleClass().....	82
Effects.....	83
.hide( ).....	83
Custom Effects.....	85
Fading.....	87
Bootstrap.....	91
What is Twitter Bootstrap.....	91
Precompiled Bootstrap.....	92
Basic template.....	92
Grid System.....	93
Old concept of grid.....	93
New concept of grid.....	93
Grid System Rules.....	93
Containers.....	94
Grid Options.....	95
Introduction.....	95
Responsive.....	96
Responsive utility classes.....	96
Typography.....	97
Table.....	98
Image Shapes.....	98
Button.....	98
Glyphicons.....	99
Progress Bar With Label.....	100
Basic Pagination.....	100
List Groups.....	101
Form Inputs.....	101
Navbar.....	105
The Modal Plugin.....	106
Tooltip.....	107
Alert messages alert.js.....	108
Checkbox / Radio.....	109
Collapse collapse.js.....	109
Customize CSS in BootStrap.....	110
Flash.....	112
Flash Embedded in HTML.....	112
Tweening.....	113
Flash Guide Tweening.....	114
Flash Tint Tweening.....	114
Flash Shape Tweening.....	115
Steps for built Flash Button1.....	116
Flash Animation.....	117
Interview Question.....	126

## Introduction to Web Designing

**Web design** include many different skills and disciplines in the production and maintenance of websites. The different areas of web design include web graphic design; interface design; authoring, including standardized code and proprietary software; user experience design; and search engine optimization. Often many individuals will work in teams covering different aspects of the design process, although some designers will cover them all. The term web design is normally used to describe the design process relating to the front-end (client side) design of a website including writing mark up. Web design partially overlaps web engineering in the broader scope of web development. Web designers are expected to have an awareness of usability and if their role involves creating markup then they are also expected to be up to date with web accessibility guidelines.

## Information design

One easily overlooked aspect of web design is information design, the organization of content and how you get to it.

### Interface design

The interface design focuses on how the page works. The concept of usability, how easily visitors can accomplish their goals on the site, as well as the general experience of using the site, is a function of the interface design.

It includes the methods for doing things on a site: buttons, links, navigation devices, etc.

### Document production

The process of writing HTML and style sheet documents is commonly referred to as authoring.

The people who handle production need to have the knowledge of HTML (the markup language used to make web documents) and style sheets, and often additional scripting or programming skills.

### Multimedia

You can add multimedia elements to a site, including sound, video, animation, and Flash movies for interactivity.

### HTML/XHTML

HTML (Hypertext Markup Language) is the language used to create web page documents. The updated version, XHTML (extensible HTML) is essentially the same language with stricter syntax rules.

You don't need programming skills—only patience and common sense—to write (X) HTML.

(X)HTML is not a programming language; it is a markup language, which means it is a system for identifying and describing the various components of a document such as headings, paragraphs, and lists. You don't need programming skills—only patience and common sense—to write (X)HTML.

Everyone involved with the Web needs a basic understanding of how HTML works. The best way to learn is to write

## CSS (Cascading Style Sheets)

While (X) HTML is used to describe the content in a web page, it is Cascading Style Sheets (CSS) that describe how you want that content to *look*.

CSS is now the official and standard mechanism for formatting text and page layouts.

Style sheets are also a great tool for automating production, because you can make changes to all the pages in your site by editing a single style sheet document. Style sheets are supported to some degree by all modern browsers.

## Server-side programming

Some web sites are collections of static (X)HTML documents and image files, but most commercial sites have more advanced functionality such as forms handling, dynamically generated pages, shopping carts, content management

Systems, databases, and so on. These functions are handled by special web applications running on the server. There are a number of scripting and programming languages that are used to create web applications, including: CGI Scripts (written in C+, Perl, Python, or others)

- ▲ Java Server Pages (JSPs)
- ▲ PHP
- ▲ VB.NET
- ▲ ASP.NET

## XML

If you hang around the web design world at all, you're sure to hear the acronym XML (which stands for extensible Markup Language). XML is not a specific language in itself, but rather a robust set of rules for creating other markup languages. To use a simplified example, if you were publishing recipes, you might use XML to create a custom markup language that includes the elements <ingredient>, <instructions>, and <servings> that accurately describe the types of information in your recipe documents. Once labeled correctly, that information can be treated as data. In fact, XML has proven to be a powerful tool for sharing data between applications.

Still, there are a number of XML languages that are used on the Web. The most prevalent is XHTML, which is HTML rewritten according to the stricter rules of XML.

## Software

There's no shortage of software available for creating web pages. In the early days, we just made do with tools originally designed for print. Today, there are wonderful tools created specifically with web design in mind that make the process more efficient.

### Web page authoring

Web-authoring tools are similar to desktop publishing tools, but the end product is a web page (an (X)HTML file and its related style sheet and image files). These tools provide a visual "WYSIWYG" (What You See Is What You

Get; pronounced "whizzy-wig") interface and shortcuts that save you from typing repetitive (X)HTML and CSS. The following are some popular web authoring programs:

**Adobe (previously Macromedia) Dreamweaver.** This is the industry standard due to its clean code and advanced features.

### HTML editors

HTML editors (as opposed to authoring tools) are designed to speed up the process of writing HTML by hand. They do not allow you edit the page visually as WYSIWYG authoring tools (listed previously) do.

### Graphics software

You'll probably want to add pictures to your pages, so you will need an image editing program.

**Adobe Photoshop.** Photoshop is undeniably the industry standard for image creation in both the print and web worlds. If you want to be a professional designer, you'll need to know Photoshop thoroughly.

### Multimedia tools

Because this is a book for beginners, I won't focus on advanced multimedia elements; however, it is still useful to be aware of the software that is available to you should you choose to follow that specialty:

**Adobe (Macromedia) Flash.** This is the hands-down favorite for adding animation, sound, and interactive effects to web pages due to the small file size of Flash movies.

### Internet tools

Because you will be dealing with the Internet, you need to have some tools specifically for viewing and moving files over the network:

**A variety of browsers.** Because browsers render pages differently, you'll want to test your pages on as many browsers as possible. There are hundreds of browsers on the market

## Web

### A Word About Browsers

We now know that the server does the serving, but what about the other half of the equation? The software that does the requesting is called the client. On the Web, the browser is the client software that makes requests for documents.

The server returns the documents for the browser to display. The requests and responses are handled via the HTTP protocol, mentioned earlier. Although we've been talking about "documents," HTTP can be used to transfer images, movies, audio files, and all the other web resources that commonly make up web sites or are shared over the Web.

When we think of a browser, we usually think of a window on a computer monitor with a web page displayed in it. These are known as graphical browsers or desktop browsers. The most popular graphical browser is Internet Explorer for Windows, with over 80% of web traffic as of this writing. However, there are many other popular browsers, including Firefox, Safari, Opera, and Netscape.

Although it's true that the Web is most often viewed on traditional graphical browsers, it is important to keep in mind that there are all sorts of browsing experiences. Users with sight disabilities may be listening to a web page read by a screen reader. Some browsers are small enough to fit into cell phones or PDAs. The sites we build must be readable in all of these environments.

Bear in mind also that your web pages may look and work differently even on up-to-date graphical browsers. This is due to varying support for web technologies and users' ability to set their own browsing preferences.

## Web Page Addresses (URLs)

Each document has its own special address called a URL (Uniform Resource Locator).

### The parts of a URL

A complete URL is generally made up of three components: the protocol, the site name, and the absolute path to the document or resource.

## HTML documents

Web pages use a markup language called the HyperText Markup Language, or HTML for short that was created especially for documents with hypertext links. HTML defines dozens of text elements that make up documents such as headings, paragraphs, emphasized text, and of course, links. There are also HTML elements that add information about the document (such as its title) and that add media such as images, videos, Flash movies, or applets to the page.

### A quick introduction to HTML

It's easy to see how the elements marked up with HTML tags in the source document correspond to what displays in the browser window.

#### **Example,**

```
<html>
<head><title>Jen's Kitchen</title></head>
<body>

<h1>Welcome to the future home of Jen's Kitchen</h1>
<p>If you love to read about <strong>cooking and eating</strong>, would like to learn of some
of the best restaurants in the world, or just want a few choice recipes to add to your
collection, <em>this is the site for you!</em></p>
<p>We're busy putting the site together.
Please check back soon.</p>
<hr />
<p>Copyright 2006, Jennifer Robbins</p>
</body>
</html>
```

First, notice that the text within brackets (for example, **<body>**) does not display in the final page. The browser only displays the content of the element; the markup is hidden. The tags provide the name of the HTML element— usually an abbreviation such as “h1” for “heading level 1,” or “em” for “emphasized text.”

Second, that most of the HTML tags appear in pairs surrounding the content of the element. In our HTML document, **<h1>** indicates that the following text should be a level-1 heading; **</h1>** indicates the end of the heading. Some elements, called empty elements, do not have content. In our sample, the **<hr />** tag indicates an empty element that tells the browser to “draw a horizontal rule (line) here.” when the browser encounters an open bracket (<) it assumes all of the following characters are part of

the markup until it finds the closing bracket (>). Similarly, it assumes all of the content following an opening <h1>tag is a heading until it encounters the closing </h1>tag. This is the manner in which the browser parses the HTML document. Understanding the browser's method can be helpful when troubleshooting a misbehaving HTML document.

## **But where are the pictures?**

Obviously, there are no pictures in the HTML file itself, so how do they get there when you view the final page?

Each image is a separate graphic file. The graphics are placed in the flow of the text with the HTML image element (**img**) that tells the browser where to find the graphic (its URL). When the browser sees the **img** element, it makes another request to the server for the image file, and then places it in the content flow. The browser software brings the separate pieces together into the final page.

The assembly of the page generally happens in an instant, so it appears as though the whole page loads all at once. Over slow connections or on slower computers, or if the page includes huge graphics, the assembly process may be more apparent as images lag behind the text. The page may even need to be redrawn as new images arrive (although you can construct your pages in a way to prevent that from happening).

## **The Nature of “WEB DESIGN”**

As a web designer, you spend a lot of time creating pages and tweaking them until they look good in your browser. Before you grow too attached to the way your page looks on your screen, you should know that it is likely to look different to other people. That's just the nature of web design—you can't guarantee that everyone will see your page the way you do. The way your site looks and performs is at the mercy of a number of variables such as browser version, platform, monitor size, and the preferences or special needs of each individual user. Your page may also be viewed on a mobile device like a cell phone, or using an assistive device like a screen magnifier or a screen reader.

This unpredictable nature of the Web is particularly challenging if you have experience designing for print, where what you design stays put. As a print designer who made the transition to web design, I found I needed to let go of controlling things such as page size, typography, and precise color. Having a solid understanding of the web environment allows you to anticipate and plan for these shifting variables. Eventually, you'll develop a feel for it.

## **Browser Versions**

One of the biggest challenges in designing for the Web is dealing with the multitude of browsers in current use.

In the no-so-distant past, browsers were so incompatible that web authors were forced to create two separate sites, one for Internet Explorer and one for Netscape (the only two players at the time). Fortunately, things have improved dramatically now that browsers have better support for web standards established by the World Wide Web Consortium (W3C for short). The situation will continue to improve as older, problematic browser versions such as Internet Explorer 5 and Netscape 4 fade out of existence. Fortunately, nearly all browsers in use today support HTML 4.01 and XHTML standards, with only a few exceptions.

That doesn't mean that an (X)HTML document will look identical on all browsers—there may still be slight differences in the default rendering of text and form elements. That's because browsers have their own internal style sheets that determine how each element looks by default. Instead, the new challenge for cross-browser consistency comes in the varying

Support of certain aspects of Cascading Style Sheets (CSS). Although most of the basic style sheet properties can be used reliably, there are still some bugs and inconsistencies that may cause unexpected results.

<b>Firefox 1.5</b>	<b>Internet Explorer 5 (Windows 2000)</b>
This page appears as the author intended. Because of IE5Win's implementation of CSS, centering	is broken, columns overlap, and the tabs run together.

How do professional web designers and developers cope with the multitude of browsers and their varying capabilities? Here are a few guidelines.



**Don't sweat the small stuff.** As a web designer, you must allow a certain amount of variation. It's the nature of the medium. What is important isn't that form input boxes are all precisely 15 pixels tall, but that they work. The first lesson you'll learn is that you have to let go.

**Stick with the standards.** Following web standards—(X)HTML for document structure and CSS for presentation—as documented by the W3C is your primary tool for ensuring your site is as consistent as possible on all standards-compliant browsers (that's approximately 99% of browsers in current use).

**Start with good markup.** When an (X)HTML document is written in logical order and its elements are marked up in a meaningful way, it will be usable on the widest range of browsing environments, including the oldest browsers, future browsers, and mobile and assistive devices. It may not look exactly the same, but the important thing is that you're content is available.

**Don't use browser-specific (X)HTML elements.** There are markup elements and attributes out there that work only with one browser or another, a remnant from the browser wars of old. Don't use them! (You won't learn them here.)

**Become familiar with the aspects of CSS that are likely to cause problems.** Using style sheets effectively takes some practice, but experienced developers know which properties are "safe," and which require some extra tweaks to get consistent results on all current browsers.

## Alternative Browsing Environments

The previous section focused on issues relevant to graphical browsers used on desktop or laptop computers. It is critical to keep in mind, however, that people access content on the Web in many different ways. Web designers must build pages in a manner that creates as few barriers as possible to getting to information, regardless of the user's ability and the device used to access the Web. In other words, you must design for accessibility.

Accessibility is a major topic of discussion in the web design world, and a priority for all web designers. While intended for users with disabilities such as poor vision or limited mobility, the techniques and strategies developed for accessibility also benefit other users with less-than-optimum browsing experiences, such as handheld devices, or traditional browsers over slow modem connections or with the images and JavaScript turned off. Accessible sites are also more effectively indexed by search engines such as Google. The extra effort in making your site accessible is well worth the effort.

## Users with disabilities

There are four broad categories of disabilities that affect how people interact with their computers and the information on them:

**Vision impairment.** People with low or no vision may use an assistive device such as a screen reader, Braille display, or a screen magnifier to get content from the screen. They may also simply use the browser's text zoom function to make the text large enough to read.

**Mobility impairment.** Users with limited or no use of their hands may use special devices such as modified mice and keyboards, foot pedals, or joysticks to navigate the Web and enter information.

**Auditory impairment.** Users with limited or no hearing will miss out on audio aspects of multimedia, so it is necessary to provide alternatives, such as transcripts for audio tracks or captions for video.

**Cognitive impairment.** Users with memory, reading comprehension, problem solving, and attention limitations benefit when sites are design simply and clearly. These qualities are helpful to anyone using your site.

The lesson here is that you shouldn't make assumptions about how your users are accessing your information. They may be hearing it read aloud. They may be pushing a button to jump from link to link on the page. The goal is to make sure your content is accessible, and the site is as easy to use as possible.

## User Preferences

At the heart of the original web concept lies the belief that the end user should have ultimate control over the presentation of information. For that reason, browsers are built with features that enable users to set the default appearance of the pages they view.

Simply by changing preference settings in the browser, anyone can affect the appearance and functionality of web pages (including yours) in the following ways.



**Change the font face and size.** The text zoom feature in modern browsers makes it easy to make text larger or smaller on the fly. Users might also change the font face in addition to the size using font settings in the browser Preferences.

**Change the background and text colors.** These days, users are less likely to alter the color settings in their browsers just for fun as they did when all web pages were comprised of black text on gray backgrounds. However, some users with impaired vision may use the browser preferences to ensure that all text is dark on a light background with plenty of contrast.

**Ignore style sheets or apply their own.** Savvy users with specific needs may create their own style sheets that apply to all the sites they view. Others may choose to simply turn style sheets off, for whatever reason.

**Turn images off.** Users can opt to turn off the graphics completely. You'd be surprised at how many people do this to alleviate the wait for bandwidth- hogging graphics over slow modem connections. Make sure your pages are at least functional with the graphics turned off. Although adding alternative text for each image helps (and it's required in HTML 4.01 and XHTML), it is not visible to 100% of your users.

**Turn off Java and JavaScript.** Your visitors can turn off technologies such as Java or JavaScript with the push of a button. With Java turned off, Java applets will not function. It is actually fairly common for users to turn off JavaScript due to security issues. The lesson is to avoid relying on technology that can be turned on and off for critical content.

**Turn off pop-up windows.** Because pop-up ads have become such a nuisance, some browsers make it easy to prevent pop-up windows from opening.

**Design for flexibility.** Whether for good reason or on a whim, the user has the final say on how pages look in the browser. The trend in contemporary web design is to build flexibility into the page. Techniques include using CSS layout techniques that specifically allow text size to change or providing multiple style sheets.

**Make sure your content is accessible without images, scripts, applets, and plug-ins.** Be prepared for the fact that some users opt to turn these features off in their browsers. It is a good idea to test your site under minimal

Conditions to make sure content is not lost and that there are no dead ends. Always provide alternative text for images and alternative means of accessing your important information or media.

## **Different Platforms**

Another variable that affects how users see your pages is the platform, or operating system, of their computers. Although most web users have personal computers running some version of the Windows operating system, a significant portion view the Web from Macintosh computers and Unix/Linux systems. The user's platform affects:

**Font availability and display.** Operating systems come with different fonts installed, so you can't assume that a font that comes installed on Windows will be available for everyone else. In addition, text tends to have a different look from platform to platform due to the methods used for sizing and rendering.

**The rendering of form elements.** Form elements such as scrolling lists and pull-down menus tend to take on the general appearance of the operating system, and therefore appear quite differently from one platform to another. They may also be sized differently, which comes into play if you are attempting to fit form elements into a space of a specific size.

**Availability of plug-in media players.** Browsers use plug-ins (or ActiveX controls on Windows) to play media such as streaming video, audio, or Flash movies that have been embedded on a web page. Fortunately, very popular players like the Flash Player are available for all platforms. Be aware, however, that some plug-in releases for Macintosh and Unix lag

Behind the Windows versions (the Windows Media Player, for example) or are not supported at all.

These are a few strategies for dealing with the fact that your page will be viewed on different platforms.

**Allow some variation.** You've heard this tip before in the previous section. As long as your content is available and functional, the small details don't matter. You'll get the hang of designing for flexibility to

allow for changing font and form control sizes.

**Specify common fonts and provide alternatives.** There are a handful of fonts that are available cross-platform, and you should always provide a list back-up fonts should your specified font not be found.

**Be sure media players are available for all platforms.** Before you commit to a particular media format, make sure that it will be accessible for all platforms. If the necessary plug-in isn't available for everyone, provide an alternative format, if possible. It has become common for media sites to offer a choice between QuickTime, Windows Media, and Real Media and let the user pick the format they prefer.

**Keep your files as small as possible.** It should be fairly intuitive that larger amounts of data will require more time to arrive. One of the worst culprits for hogging bandwidth is graphics files, so it is especially important that you spend time optimizing them for the Web. (X)HTML files, although generally just a few kilobytes (KB) in size, can be optimized as well by removing redundant markup and extra spaces. Audio, video, and multimedia content also consume lots of bandwidth and should be compressed appropriately. Because you know a web page is designed to travel, do your best to see that it travels light.

## Browser Window Size and Monitor

### Resolution

Although you may prefer the way your page looks when the window is just larger than the masthead you designed, the fact is users can set the window as wide or narrow as they please. You really have no idea how big your page will be: as large as the user's monitor will allow, or smaller according to personal preference or to accommodate several open windows at once.

But don't worry. Not only will you become familiar with how our content behaves at different window sizes, there are also design techniques that can make the page layout more predictable.

### Web page dimensions

Because browser windows can only be opened as large as the monitors displaying them, standard monitor resolution (the total number of pixels available on the screen) is useful in anticipating the likely dimensions of your page. This is particularly true on Windows machines, because the browser window is typically maximized to fill the monitor. The sidebar, Common Monitor Resolutions, lists the most popular resolutions as well as how much space that leaves for your content.

As of this writing, most commercial web sites are designed to fit in an 800 x 600 monitor, the smallest monitor that is still in significant use. Allowing for the browser chrome and operating system menus, that leaves a canvas area of approximately 775 x 425 pixels for your web content.

## HTML Overview

This section shows how to open new documents in Notepad and Text Edit. Even if you've used these programs before, skim through for some special settings that will make the exercises go more smoothly. We'll start with Notepad; Mac users can jump ahead.

## Creating a new document in Notepad (Windows users)

These are the steps to creating a new document in Notepad on Windows XP.

- Open the Start menu and navigate to Notepad (in Accessories)
- Clicking on Notepad will open a new document window, and you're ready to start typing.
- Next, we'll make the extensions visible. This step is not required to make (X)HTML documents, but it will help make the file types more clear at a glance.
- In any Explorer window, select "Folder Options..." from the Tools menu and select the "View" tab.
- Find "Hide extensions for known file types" and uncheck that option.
- Click OK to save the preference and the file extensions will now be visible.

## Introducing... the HTML element

Back in Chapter 2, How the Web Works, you saw examples of (X)HTML elements with an opening tag (<p> for a paragraph, for example) and closing tag (</p>). Before we start adding tags to our document, let's look at the structure of an HTML element and firm up some important terminology. A generic (X)HTML element:

Elements are identified by tags in the text source. A tag consists of the element name (usually an abbreviation of a longer descriptive name) within angle brackets (<>). The browser knows that any text within brackets is hidden and not displayed in the browser window.

The element name appears in the opening tag (also called a start tag) and again in the closing (or end) tag preceded by a slash (/). The closing tag works something like an “off” switch for the element. Be careful not to use the similar backslash character in end tags (see the tip, Slash vs. Backslash).

The tags added around content are referred to as the markup. It is important to note that an element consists of both the content *and* its markup (the start and end tags). Not all elements have content, however. Some are empty by definition, such as the **img** element used to add an image to the page.

## Basic document structure

Much like you and me, (X)HTML documents have a head and a body. The

head of the document (also sometimes called the header) contains descriptive information about the document itself, such as its title, the style sheet it uses, scripts, and other types of “meta” information. The body contains the actual content that displays in the browser window.

First, the entire document is contained within an **html** element. The **html** element is called the root element because it contains all the elements in the document, and it may not be contained within any other element. It is used for both HTML and XHTML documents.

The **head** comes next and contains the **title** element. According to the (X)HTML specifications, every document must contain a descriptive title. The **body** element comes after the **head** and contains everything that we want to show up in the browser window. The document structure elements do not affect how the content looks in the browser (as you’ll see in a moment), but they are required to make the document valid (that is, to properly abide by the (X)HTML standards).

## Block and inline elements

The heading and paragraph elements start on new lines and do not run together as they did before. That is because they are examples of block-level elements. Browsers treat block-level elements as though they are in little rectangular boxes, stacked up in the page. Each block-level element begins on a new line, and some space is also usually added above and below the entire element by default.

By contrast, look at the text we marked up as emphasized (**em**). It does not start a new line, but rather stays in the flow of the paragraph. That is because the **em** element is an inline element. Inline elements do not start new lines; they just go with the flow.

The distinction between block-level and inline elements is important. In (X)HTML markup, whether an element is block-level or inline restricts what other elements it may contain. For example, you can’t put a block-level element within an inline element (such as a paragraph within a link). Block level and inline elements also behave differently when it comes to applying Cascading Style Sheets.

## Empty elements

A handful of elements, however, do not have text content because they are used to provide a simple directive. These elements are said to be empty. The image element (**img**) is an example of such an element; it tells the browser to get an graphic file from the server and insert it into the flow of the text at that spot in the document. Other empty elements include the line break (**br**),

Horizontal rule (**hr**), and elements that provide information about a document but don’t affect its displayed content, such as the **meta** element.

The syntax for empty elements is slightly different for HTML and XHTML. In HTML, empty elements don’t use closing tags—they are indicated by a single tag (<**img**>, <**br**>, or <**hr**>, for example) inserted into the text, as shown in this example that uses the **br** element to insert a line break.

```
<p>1005 Gravenstein Highway North <br>Sebastopol, CA 95472</p>
```

In XHTML, all elements, including empty elements, must be closed. Empty elements are terminated by adding a trailing slash preceded by a space before the closing bracket, like so:

<**img** />, <**br** />, and <**hr** />. Here is that example again, this time using XHTML syntax.

```
<p>1005 Gravenstein Highway North <br />Sebastopol, CA 95472</p>
```

## Attributes

Obviously, an `<img />` tag is not very useful by itself... there's no way to know which image to use. That's where attributes come in. Attributes are instructions that clarify or modify an element. For the **img** element, the **src** (short for "source") attribute is required, and provides the location of the image file via its URL.

The syntax for attributes is as follows:

```
<element attribute-name="value">Content</element>
```

or for empty elements:

```
<element attribute-name="value" />
```

For another way to look at it, the attribute structure of an **img** element:

Here's what you need to know about attributes:

- Attributes go after the element name in the opening tag only, never in the end tag.
- There may be several attributes applied to an element, separated by spaces in the opening tag. Their order is not important.
- Attributes take values, which follow an equals sign (=).
- A value might be a number, a word, a string of text, a URL, or a measurement depending on the purpose of the attribute.
- Always put values within quotation marks. Although quotation marks aren't required around all values in HTML, they *are* required in XHTML. You might as well do it the more future-compatible way from the start. Either single or double quotation marks are acceptable as long as they are used consistently, however, double quotation marks are the convention.
- Some attributes are required, such as the **src** and **alt** attributes in the **img** element.
- The attribute names available for each element are defined in the (X)HTML specifications; in other words, you can't make up an attribute for an element.

## Adding Link

If you're creating a page for the Web, chances are you'll want it to point to other web pages, whether to another section of your own site or to someone else's. You can even link to another spot on the same page. Linking, after all, is what the Web is all about. In this chapter, we'll look at the markup that makes links work: to other sites, to your own site, and within a page. If you've used the Web at all, you should be familiar with the highlighted text and graphics that indicate "click here." There is one element that makes linking possible: the anchor (**a**).

```
<a>...</a>
```

### Anchor element (hypertext link)

The content of the anchor element becomes the hypertext link. Simply wrap a selection of text in opening and closing `<a>...</a>` tags and use the href attribute to provide the URL of the linked page. Here is an example that creates a link to the O'Reilly Media web site:

```
<a href="http://www.oreilly.com">Go to O'Reilly.com</a>
```

To make an image a link, simply put the **img** element in the anchor element:

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

The only restriction is that because anchors are inline elements, they may only contain text and other inline elements. You may not put a paragraph, heading, or other block element between anchor tags. Most browsers display linked text as blue and underlined, and linked images with a blue border. Visited links generally display in purple. Users can change these colors in their browser preferences, and, of course, you can change the appearance of links for your sites using style sheets.

### The href Attribute

You'll need to tell the browser which document to link to, right? The href (hypertext reference) attribute provides the address of the page (its URL) to the browser. The URL must always appear in quotation marks. Most of the time you'll point to other (X)HTML documents; however, you can also point to other web resources, such as images, audio, and video files.

Because there's not much to slapping anchor tags around some content, the real trick to linking comes in getting the URL correct.

There are two ways to specify the URL:

Absolute URLs provide the full URL for the document, including the protocol (<http://>), the domain name, and the pathname as necessary. You need to use an absolute URL when pointing to a document out on the Web.

**Example:** `href="http://www.oreilly.com/"`

Sometimes, when the page you're linking to has a long URL pathname, the link can end up looking pretty confusing. Just keep in mind that the structure is still a simple container element with one attribute. Don't let the pathname intimidate you.

Relative URLs describe the pathname to the linked file relative to the current document. It doesn't require the protocol or domain name—just the pathname. Relative URLs can be used when you are linking to another document on your own site (i.e., on the same server).

**Example:** `href="recipes/index.html"`

## Linking to Pages on the Web

Many times, you'll want to create a link to a page that you've found on the Web. This is known as an "external" link because it is going to a page outside of your own server or site. To make an external link, you need to provide the

Absolute URL, beginning with <http://> (the protocol). This tells the browser, "Go out on the Web and get the following document."

I want to add some external links to the Jen's Kitchen home page. First, I'll link the list item "The Food Network" to the site [www.foodtv.com](http://www.foodtv.com). I marked up the link text in an anchor element by adding opening and closing anchor tags. Notice that I've added the anchor tags inside the list item (li) element. That's because block-level elements, such as li, may not go inside the inline anchor element.

```
<li><a>The Food Network</a></li>
```

Next, I add the href attribute with the complete URL for the site.

```
<li><a href="http://www.foodtv.com">The Food Network</a></li>
```

And voila! That's all there is to it. Now "The Food Network" will appear as a link, and will take my visitors to that site when they click it.

## Linking Within Your Own Site

A large portion of the linking you'll do will be between pages of your own site: from the home page to section pages, from section pages to content pages, and so on. In these cases, you can use a relative URL—one that calls for a page on your own server.

### Linking within a directory

The most straightforward relative URL to write is to another file within the same directory. When you are linking to a file in the same directory, you only need to provide the name of the file (its filename). When the URL is a single file name, the server looks in the current directory (that is, the directory that contains the (X)HTML document with the link) for the file.

In this example, I want to make a link from my home page (index.html) to a general information page (about.html). Both files are in the same directory (jenskitchen). So from my home page, I can make a link to the information page by simply providing its filename in the URL : `<a href="about.html">About the site...</a>`

### Linking to a lower directory

But what if the files aren't in the same directory? You have to give the browser directions by including the pathname in the URL. Let's see how this works.

Getting back to our example, my recipe files are stored in a subdirectory called recipes. I want to make a link from index.html to a file in the recipes directory called salmon.html. The pathname in the URL tells the browser to look in the current directory for a directory called recipes, and then look for the file salmon.html:

```
<li><a href="recipes/salmon.html">Garlic Salmon</a></li>
```



## Linking to a higher directory

So far, so good, right? Here comes the tricky part. This time we're going to go in the other direction and make a link from the salmon recipe page back to the home page, which is one directory level up.

In Unix, there is a pathname convention just for this purpose, the "dot-dot-slash" (../). When you begin a pathname with a ../, it's the same as telling the browser "back up one directory level" and then follow the path to the specified file. If you are familiar with browsing files on your desktop with, it is helpful to know that a "../" has the same effect as clicking the "Up" button in Windows Explorer or the left-arrow button in the Finder on Mac OS X.

Let's start by making a link back to the home page (index.html) from salmon.html. Because salmon.html is in the recipes subdirectory, we need to back up a level to jenskitchen to find index.html. This pathname tells the browser to "go up one level," then look in that directory for index.html:

```
<p><a href="../index.html">[Back to home page]</a></p>
```

Note that we don't need to write out the name of the higher directory (jenskitchen) in the pathname. The ../ stands in for it.

A link on the couscous.html page back to the home page (index.html) would look like this:

```
<p><a href="../../index.html">[Back to home page]</a></p>
```

The first ../ backs up to the recipes directory; the second ../ backs up to the top-level directory where index.html can be found. Again, there is no need to write out the directory names; the ../ does it all.

## Site root relative pathnames

All web sites have a root directory, which is the directory that contains all the directories and files for the site. So far, all of the pathnames we've looked at are relative to the document with the link. Another way to write a pathname is to start at the root directory and list the sub-directory names until you get to the file you want to link to. This kind of pathname is known as site root relative.

In the Unix pathname convention, the root directory is referred to with a forward slash (/) at the start of the pathname. The site root relative pathname in the following link reads, "Go to the very top-level directory for this site, open the recipes directory, then find the salmon.html file" :

```
<a href="/recipes/salmon.html">Garlic Salmon</a>
```

Note that you don't need to write the name of the root directory (jenskitchen) in the URL—the forward slash (/) stands in for it and takes the browser to the top level. From there, it's a matter of specifying the directories the browser should look in.

Because this link starts at the root to describe the pathname, it will work from any document on the server, regardless of which sub-directory it may be located in. Site root relative links are useful for content that might not always be in the same directory, or for dynamically generated material. They also make it easy to copy and paste links between documents. On the down-side, however, the links won't work on your local machine because they will be relative to your hard drive. You'll have to wait until the site is on the final server to check that links are working.

## Adding Images

Images appear on web pages in two ways: as part of the inline content or as tiling background images. Background images are added using Cascading Style Sheets. In this chapter, we'll focus on adding image content to the document using the inline img element.

Inline images may be used on web pages in several ways:

As a simple image. An image can be used on a web page much as it is used in print, as a static image that adds information, such as a company logo or an illustration.

As a link. As we saw in the previous chapter, an image can be used as a link to another document by placing it in the anchor element.

As an imagemap. An imagemap is a single image that contains multiple links ("hotspots") that link to other documents. We'll look at the markup used to add clickable areas to images in this chapter as well.



## First, a Word on Image Formats

We'll get to the `img` element and markup examples in a moment, but first it's important to know that you can't put just any image on a web page. In order to be displayed inline, images must be in the GIF, JPEG, or PNG file format.

## The `img` Element

`<img />` (XHTML)

`<img>` (HTML)

Adds an inline image The `img` element tells the browser, "Place an image here." You add it in the flow of text at the point where you want the image to appear, as in this example. Because it is an inline element, it does not cause any line breaks,

When the browser sees the `img` element, it makes a request to the server and retrieves the image file before displaying it on the page. Even though it makes a separate request for each image file, the speed of networks and computers

usually makes it appear to happen instantaneously (unless you are dialing in on a slow modem connection).

The `src` and `alt` attributes shown in the sample are required. The `src` attribute tells the browser the location of the image file. The `alt` attribute provides alternative text that displays if the image is not available. We'll talk about `src` and `alt` a little more in upcoming sections.

There are a few other things of note about the `img` element:

- It is an empty element, which means it doesn't have any content. You just place it in the flow of text where the image should go.
- In XHTML, empty elements need to be terminated, so the `img` element is written `<img />`. In HTML, it's simply `<img>`.
- It is an inline element, so it behaves like any other inline element in the text flow.
- The `img` element is what's known as a replaced element because it is replaced by an external file when the page is displayed. This makes it different from text elements that have their content right there in the (X)HTML source (and thus are non-replaced).
- By default, the bottom edge of an image aligns with the baseline of text. Using Cascading Style Sheets, you can float the image to the right or left margin and allow text to flow around it, control the space and borders around the image, and change its vertical alignment.

## Providing the location with `src`

`src="URL"`

Source (location) of the image

The value of the `src` attribute is the URL of the image file. In most cases, the images you use on your pages will reside on your server, so you will use relative URLs to point to them. If the image is in the same directory as the (X)HTML document, you can just refer to the image by name in the `src` attribute:

``

Developers usually organize the images for a site into a directory called `images` or `graphics`. There may even be separate image directories for each section of the site. If an image is not in the same directory as the document, you need to provide the relative pathname to the image file.

``

Of course, you can place images from other web sites as well (just be sure that you have permission to do so). Just use an absolute URL, like this:

``

## Providing alternate text with `alt`

`alt="text"`

Alternative text

Every img element must also contain an alt attribute that is used to provide a brief description of the image for those who are not able to see it, such as users with screen readers, Braille, or even small mobile devices. Alternate text (also referred to as alt text) should serve as a substitute for the image content—serving the same purpose and presenting the same information.

```
<p>If you're  and you know it clap your hands.</p>
```

A screen reader might indicate the image and its alt value this way:

“If you’re image happy and you know it clap your hands.”

If an image is purely decorative, or does not add anything meaningful to the text content of the page, it is recommended that you leave the value of the alt attribute empty, as shown in this example and other examples in this chapter. Note that there is no character space between the quotation marks.

```

```

## **Providing width and height dimensions**

width="number"

Image width in pixels

height="number"

Image height in pixels

The width and height attributes indicate the dimensions of the image in number of pixels. Sounds mundane, but these attributes can speed up the time it takes to display the final page.

When the browser knows the dimensions of the images on the page, it can busy itself laying out the page while the image files themselves are download-ing. Without width and height values, the page is laid out immediately, and then reassembled each time an image file arrives from the server. Telling the browser how much space to hold for each image can speed up the final page display by seconds for some users.

## **Image maps**

In your web travels, I’m sure you’ve run across a single image that has multiple “hotspots,” or links, within it. These images are called image maps.

Putting several links in one image has nothing to do with the image itself; it’s just an ordinary image file placed with an img element. Rather, the image merely serves as the frontend to the mechanisms that match particular mouse-click coordinates to a URL.

The real work is done by a map in the source document that matches sets of pixel coordinates to their respective link information. When the user clicks somewhere within the image, the browser passes the pixel coordinates of the pointer to the map, which in turn generates the appropriate link. When the cursor passes over a hotspot, the cursor changes to let the user know that the area is a link. The URL may also appear in the browser’s status bar. Because the browser does the matching of mouse coordinates to URLs, this type of image map is called a client-side imagemap.

## **The parts of an imagemap**

Client-side imagemaps have three components: An ordinary image file (.gif, .jpg/.jpeg, or .png) placed with the img element. The usemap attribute within that img element that identifies which map to use (each map is given a name) A map element that is a container for some number of area elements. Each area element corresponds to a clickable area in the imagemap and contains the pixel coordinate and URL information for that area. We’ll look at a map in detail in a moment.

## **Creating the map**

Fortunately, there are tools that generate maps so you don’t have to write out the map by hand. Nearly all web-authoring and web graphics tools currently on the market (Adobe’s Dreamweaver, Fireworks, and Photoshop/ImageReady being the most popular) have built-in imagemap generators. You could also download shareware imagemap programs (see the sidebar Imagemap Tools).

The process for creating the map is essentially the same for all imagemap tools:

- Open the image in the imagemap program (or place it on the page in a web-authoring tool).
- Define an area that will be “clickable” by using the appropriate shape tools: rectangle, circle, or polygon (for tracing irregular shapes).
- While the shape is still highlighted, enter a URL for that area in the text entry field provided A. Enter alternative text for the area as well C.

- Continue adding shapes and their respective URLs for each clickable area in the image.
- Select the type of imagemap you want to create—client-side is the only practical option.
- Give the map a name in the provided map name field D.
- Add the map to the (X)HTML document. Web-authoring tools, such as Dreamweaver, insert the map automatically. If you are using ImageReady or another tool, you need to export or save the map code, then copy and paste it into the (X)HTML file. The map can go at the top or the bottom of the document; just make sure to keep it together. Then make sure that the img element points to the correct map name.
- Save the (X)HTML document and open it in your browser.

## It's the same for images

The src attribute in the img element works the same as the href attribute in anchors when it comes to specifying URLs. Since you'll most likely be using images from your own server, the src attributes within your image elements will be set to relative URLs.

Let's look at a few examples from the Jen's Kitchen site. First, to add an image to the index.html page, the markup would be:

```

```

The URL says, "Look in the current directory (jenskitchen) for the images directory; in there you will find jenskitchen.gif."

Now for the piece de résistance. Let's add an image to the file couscous.html:

```

```

This is a little more complicated than what we've seen so far. This pathname tells the browser to go up two directory levels to the top-level directory and, once there, look in the images directory for a image called spoon.gif. Whew! Of course, you could simplify that path by going the site root relative route, in which case, the pathname to spoon.gif (and any other file in the images directory) could be accessed like this: 

```

```

The trade-off is that you won't see the image in place until the site is uploaded to the server, but it does make maintenance easier once it's there.

## Lists

Sometimes it is necessary to itemize information instead of breaking it into paragraphs. There are three main types of lists in (X)HTML: Unordered lists. Collections of items that appear in no particular order. Ordered lists. Lists in which the sequence of the items is important. Definition lists. Lists that consist of terms and definitions.

All list elements—the lists themselves and the items that go in them—are block-level elements, which means that they always start on a new line by default. In this section, we'll look at each list type in detail.

### Unordered lists

By default, unordered lists display with a bullet before each list item, but you can change that with a style sheet, as you'll see in a moment. To identify an unordered list, mark it up as a ul element. The opening <ul> tag goes before the first list item and the closing tag </ul> goes after the last item. Then, each item in the list gets marked up as a list item (li) by enclosing

it in opening and closing li tags as shown in this example.

Notice that there are no bullets in the source document. They are added automatically by the browser.

```
<ul>
<li>Serif</li>
<li>Sans-serif</li>
<li>Script</li>
<li>Display</li>
<li>Dingbats</li>
</ul>
```

#### Output:-

- Serif
- Sans-serif
- Script
- Display
- Dingbats

We can take that same unordered list markup, and radically change its appearance by applying different style sheets.

## Ordered lists

Ordered lists are for items that occur in a particular order, such as step-by-step instructions or driving directions. They work just like unordered lists described earlier, except they are defined with the `ol` element (for ordered list, naturally). Instead of bullets, the browser automatically inserts numbers before ordered list items, so you don't need to number them in the source

Document. This makes it easy to rearrange list items without renumbering them.

Ordered list elements must contain one or more list item elements.

```
<ol>
<li>Gutenberg develops moveable type (1450s)</li>
<li>Linotype is introduced (1890s)</li>
<li>Photocomposition catches on (1950s)</li>
<li>Type goes digital (1980s)</li>
</ol>
```

Output:-

1. Gutenberg develops moveable type (1450s)
2. Linotype is introduced (1890s)
3. Photocomposition catches on (1950s)
4. Type goes digital (1980s)

If you want a numbered list to start at a number other than "1," you can use the `start` attribute in the `ol` element to specify another starting number, as shown here:

```
<ol start="17">
<li>Highlight the text with the text tool.</li>
<li>Select the Character tab.</li>
<li>Choose a typeface from the pop-up menu.</li>
</ol>
```

Output:-

17. Highlight the text with the text tool.
18. Select the Character tab
19. Choose a typeface from the pop-up menu

The resulting list items would be numbered 17, 18, and 19, consecutively.

## Definition lists

Definition (or dictionary) lists are used for lists of terms with their respective definitions. They are a bit different from the other two list types in format. The whole list is marked up as a definition list (`dl`) element. The content of a `dl` is some number of terms (indicated by the `dt` element) and definitions (indicated by the `dd` element).

```
<dl>
  <dt>Linotype</dt>
  <dd>Line-casting allowed type to be selected, used, then recirculated into the machine automatically. This advance increased the speed of typesetting and printing dramatically.</dd>
  <dt>Photocomposition</dt>
  <dd>Typefaces are stored on film then projected onto photo-sensitive paper. Lenses adjust the size of the type.</dd>
  <dt>Digital type</dt>
  <dd><p>Digital typefaces store the outline of the font shape in a format such as Postscript. The outline may be scaled to any size for output.</p></dd>
</dl>
```

The `dl` element is only allowed to contain `dt` and `dd` elements. It is okay to have multiple definitions with one term and vice versa. You cannot put block-level elements (like headings or paragraphs) in terms (`dt`), but the definition (`dd`) can contain any type of content (inline or block-level elements).

## Table Markup

HTML tables were created for instances when you need to add tabular material (data arranged into rows and columns) to a web page. Tables may be used to organize calendars, schedules, statistics, or other types of information. Note that "data" doesn't necessarily mean numbers. A table cell may

contain any sort of information, including numbers, text elements, even images and multimedia objects.

In visual browsers, the arrangement of data in rows and columns gives readers an instant understanding of the relationships between data cells and their respective header labels. Bear in mind when you are creating tables, however, that some readers will be hearing your data read aloud with a screen reader or reading Braille output. Later in this chapter, we'll discuss measures you can take to make table content accessible to users who don't have the benefit of visual presentation.

## Minimal Table Structure

Let's take a look at a simple table to see what it's made of. Here is a small table with three rows and three columns that lists nutritional information.

The elements that identify the table (table), rows (tr, for "table row"), and cells (th, for "table headers," and td, for "table data"). Cells are the heart of the table, because that's where the actual content goes. The other elements just hold things together.

What we don't see are column elements (see note). The number of columns in a table is determined by the number of cells in each row. This is one of the things that make (X)HTML tables potentially tricky. Rows are easy—if you want the table to have three rows, just use three tr elements. Columns are different. For a table with four columns, you need to make sure that every row has four td or th elements; the columns are implied.

```
<table border="1">
<tr>
<th>Menu item</th><th>Calories</th>
<th>Fat (g)</th>
</tr>
<tr>
<td>Chicken noodle soup</td>
<td>120</td><td>2</td>
</tr>
<tr>
<td>Caesar salad</td><td>400</td><td>26</td>
</tr>
</table>
```

Output:-

Menu item	Calories	Fat (g)
Chicken Noodle shoup	120	2
Caesar Salad	400	26

Remember, all the content for a table must go in cells; that is, within td or th elements. You can put any content in a cell: text, a graphic, even another table.

Start and end table tags are used to identify the beginning and end of the table. The table element may only directly contain some number of tr (row) elements. The only thing that can go in the tr element is some number of td or th elements. In other words, there may be no text content within the table and tr elements that isn't contained within a td or th.

## Spanning Cells

One fundamental feature of table structure is cell spanning, which is the stretching of a cell to cover several rows or columns. Spanning cells allows you to create complex table structures, but it has the side effect of making the markup a little more difficult to keep track of. You make a header or data cell span by adding the colspan or rowspan attributes, as we'll discuss next.

## Column spans

Column spans, created with the colspan attribute in the td or the element, stretch a cell to the right to span over the subsequent columns. Here a column span is used to make a header apply to two columns. (I've added a border around cells to reveal the table structure in the screenshot.)

```
<TABLE border="1">
<TR>
<TD colspan="2">my first table</TD>
</TR>
<TR>
<TD>my first table</TD>
<TD>my first table</TD>
</TR>
</TABLE>
```

Output:-

my first table	
my first table	my first table

Notice in the first row (tr) that there is only one th element, while the second row has two td elements. The th for the column that was spanned over is no longer in the source; the cell with the colspan stands in for it. Every row should have the same number of cells or equivalent colspan values. For example, there are two td elements and the colspan value is 2, so the implied number of columns in each row is equal.

## Row spans

Row spans, created with the rowspan attribute, work just like column spans, except they cause the cell to span downward over several rows. In this example, the first cell in the table spans down three rows.

```
<TABLE BORDER=1>
<TR>
<TD rowspan=2>
  my first table
</TD>
<TD>
  my first table
</TD>
</TR>
<TR>
<TD>
  my first table
</TD>
</TR>
</TABLE>
```

Output:-

my first table	my first table
	my first table

## Cell Padding and Spacing

By default, cells are sized just large enough to fit their contents, but often, you'll want to add a little breathing room around tabular content. There are two kinds of space that can be added in and around table cells: cell padding and cell spacing, using the cellpadding and cellspacing attributes, respectively. These attributes may be used with the table element only. In other words, you can't apply them to tr, td, or th elements.

### Cell padding

Cell padding is the amount of space held between the contents of the cell and the cell border. If you don't specify any cell padding, the cells will have the default value of one pixel of padding.

```
<table cellpadding="15">
<tr><td>CELL 1</td>
<td>CELL 2</td></tr>
<tr><td>CELL 3</td>
<td>CELL 4</td></tr>
</table>
```

Output:-

CELL 1	CELL 2
CELL 3	CELL 4

Because the cellpadding attribute may be used with the table element only, the cellpadding value applies to all the cells in the table. In other words, you can't specify different amounts of padding for individual cells with this attribute. However, you can apply padding amounts on a cell-by-cell basis using the padding property in CSS.

### Cell spacing

Cell spacing is the amount of space held between cells, specified in number of pixels. If you don't specify anything, the browser will use the default value of two pixels of space between cells.

```
<table cellpadding="15" cellspacing="15">
<tr><td>CELL 1</td>
<td>CELL 2</td></tr>
<tr><td>CELL 3</td>
<td>CELL 4</td></tr>
</table>
```

Output:-

CELL 1	CELL 2
CELL 3	CELL 4

## Captions and Summaries

There are two methods for providing additional information about a table: captions and summaries.



The difference is that the caption is displayed with the table in visual browsers, while the summary is not displayed but may be used by assistive devices. Both captions and summaries are useful tools in improving table accessibility.

## The caption element

The caption element is used to give a table a title or brief description. The caption element must be the first thing within the table element, as shown in this example that adds a caption to the nutritional chart from earlier in the chapter.

```
<table>
<caption>Nutritional Information</caption>
<tr> <th>Menu item</th><th>Calories</th>
<th>Fat (g)</th></tr>
<tr><td>Chicken noodle soup</td><td>120</td><td>2</td>
</tr>
<tr> <td>Caesar salad</td>
<td>400</td><td>26</td></tr></table>
```

Output:-

Nutritional Information		
Menu item	Calories	Fat (g)
Chicken noodle soup	120	2
Caesar salad	400	26

## The summary attribute

Summaries are used to provide a more lengthy description of the table and its contents. They are added using the summary attribute in the table element, as shown here.

```
<table summary="A listing of the calorie and fat content for each of the most popular menu items">
<caption>Nutritional Information</caption> ...table continues...</table>
```

The summary is not rendered in visual browsers, but may be used by screen readers or other assistive devices to give visually impaired users a better understanding of the table's content, which sighted users could understand at a glance. This alleviates the need to listen to several rows of data before deciding whether to continue with the table data or skip it.

## Forms

There are two parts to a working form. The first part is the form that you see on the page itself. Forms are made up of buttons, text fields, and pull-down menus (collectively known as form controls) used to collect information from the user. Forms may also contain text and other elements. The other component of a web form is an application or script on the server that processes the information collected by the form and returns an appropriate response. It's what makes the form work.

### The form Element

Forms are added to web pages using the form element. The form element is a container for all the content of the form, including some number of form controls, such as text entry fields and buttons. It may also contain block elements, (h1, p, and lists, for example), however, it may not contain another form element.

```
<html><head><title>Mailing List Signup</title>
</head><body>
<h1>Mailing List Signup</h1>
<form action="/cgi-bin/maillinglist.pl" method="post">
<fieldset>
<legend>Join our email list</legend>
<p>Get news about the band such as tour dates and special
MP3
releases sent to your own in-box.</p>
<ol><li><label for="name">Name:</label>
<input type="text" name="name" id="name" /></li>
<li><label for="name">Email:</label>
<input type="text" name="email" id="email" /></li>
</ol>
<input type="submit" value="Submit" />
</fieldset>
</form></body>
</html>
```

Output:-

## Mailing List Signup

Join our email list

Get news about the band such as tour dates and special MP3 releases sent to your own in-box.

1. Name:

2. Email:

Submit

## **The action attribute**

The action attribute provides the location (URL) of the application or script (sometimes called the action page) that will be used to process the form. The action attribute in this example sends the data to a script called mailinglist.pl. The script is the cgi-bin directory on the same server as the HTML document (you can tell because the URL is site root relative).

```
<form action="/cgi-bin/ mailinglist.pl" method="post">...</form>
```

The .pl suffix indicates that this form is processed by a Perl script (Perl is a scripting language). It is also common to see web applications that end with the following:

- .php, indicating that a PHP program is doing the work. PHP is an open source scripting language most commonly used with the Apache web server.
- .asp, for Microsoft's ASP (Active Server Pages) programming environment for the Microsoft Internet Information Server (IIS).
- .jsp, for JavaServer Pages, a Java-based technology similar to ASP.

When you create a web form, you most likely will be working with a developer or server administrator who will provide the name and location of the program to be provided by the action attribute.

## **The method attribute**

The method attribute specifies how the information should be sent to the server.

name = Sally Strongarm

email = strongarm@example.com

When the browser encodes that information for its trip to the server, it looks like this (see the earlier sidebar if you need a refresher on encoding):

name=Sally%20Strongarm&email=strongarm%40example.com

There are only two methods for sending this encoded data to the server: POST or GET indicated using the method attribute in the form element. We'll look at the difference between the two methods in the following sections. Our example uses the POST method, as shown here:

```
<form action="/cgi-bin/ mailinglist.pl" method="post">...</form>
```

## **The POST method**

When the form's method is set to POST, the browser sends a separate server request containing some special headers followed by the data. Only the server sees the content of this request, thus it is the best method for sending secure information such as credit card or other personal information.

The POST method is also preferable for sending a lot of data, such as a lengthy text entry, because there is no character limit as there is for GET.

## **The GET method**

With the GET method, the encoded form data gets tacked right onto the URL sent to the server. A question mark character separates the URL from the following data, as shown here:

Gethttp://www.bandname.com/cgi-

bin/ mailinglist.pl?name=Sally%20Strongarm&email=strongarm%40example.com

The GET method is appropriate if you want users to be able to bookmark the results of a form submission (such as a list of search results). Because the content of the form is in plain sight, GET is not appropriate for forms with private personal or financial information. In addition, because there is a 256 character limit on what can be appended to a URL, GET may not be used for sending a lot of data or when the form is used to upload a file.

## **Variables and Content**

Web forms use a variety of controls (also sometimes called widgets) that allow users to enter information or choose options. Control types include various text entry fields, buttons, menus, and a few controls with special functions.

As a web designer, it is important to be familiar with control options to make your forms easy and intuitive to use. It is also useful to have an idea of what form controls are doing behind the scenes.

## The name attribute

The job of a form control is to collect one bit of information from a user. In the form example a few pages back, the text entry fields are used to collect the visitor's name and email address. To use the technical term, "name" and "email" are two variables collected by the form. The data entered by the user ("Sally Strongarm" and "strongarm@example.com") is the value or content of the variable.

The name attribute identifies the variable name for the control. In this example, the text gathered by a textarea element is identified as the "comment" variable:

```
<textarea name="comment" rows="4" cols="45">Would you like to add a comment?</textarea>
```

When a user enters a comment in the field ("This is the best band ever!"), it would be passed to the server as a name/value (variable/content) pair like this:

```
comment=This%20is%20the%20best%20band%20ever!
```

All form control elements must include a name attribute so the form-processing application can sort the information. The only exceptions are the submit and reset button elements because they have special functions (submitting or resetting the form) not related to data collection.

## Form Elements

### Labels

The label element is used to associate descriptive text with its respective form field. This provides important context for users with speech-based browsers.

Each label element is associated with exactly one form control. There are two ways to use it. One method, called implicit association, nests the control and its description within a label element, like this:

```
<label>Male: <input type="radio" name="gender" value="M" /></label>
```

```
<label>Female: <input type="radio" name="gender" value="F" /></label>
```

The other method, called explicit association, matches the label with the control's id reference. The for attribute says which control the label is for. This approach is useful when the control is not directly next to its descriptive text in the source. It also offers the potential advantage of keeping the label and the control as two distinct elements, which may come in handy when aligning them with style sheets.

```
<label for="form-login-username">Login account:</label>
```

```
<input type="text" name="login" id="form-login-username" />
```

```
<label for="form-login-password">Password:</label>
```

```
<input type="password" name="password" id="form-login-password" />
```

Output:-

Male: ☐ Female: ☐ The other method, called explicit association, matches the label with the control's id reference. The for attribute says which control the label is for. This approach is useful when the control is not directly next to its descriptive text in the source. It also offers the potential advantage of keeping the label and the control as two distinct elements, which may come in handy when aligning them with style sheets. Login account:  Password:

### fieldset and legend

it use to indicate a logical group of form controls. may also include a legend element that provides a caption for the enclosed fields.

```
<fieldset><legend>Customer Information</legend>
<ol><li><label>Full name: <input type="text" name="name" /></label></li>
<li><label>Email: <input type="text" name="email" /></label></li>
<li><label>State: <input type="text" name="state" /></label></li></ol>
</fieldset><fieldset>
<legend>Mailing List Sign-up</legend>
<ul><li><label>Add me to your mailing list <input type="radio"
name="list" value="yes" checked="checked" /></label></li>
<li><label>No thanks <input type="radio" name="list" value="no" />
</label></li></ul>
</fieldset>
```

Output:-

Customer Information

1. Full name:
2. Email:
3. State:

Mailing List Sign-up

- ☒ Add me to your mailing list
- ☐ No thanks

The Great Form Control Round-up This is the fun part—playing with the markup that adds form controls to the page. This section introduces the elements used to create:

- Text entry controls
- Submit and reset buttons
- Radio and checkbox buttons
- Pull-down and scrolling menus
- File selection and upload control
- Hidden controls

The majority of controls are added to a form using the input element. The functionality and appearance of the input element changes based on the type attribute.

## Text entry controls

There are three basic types of text entry fields in web forms: single-line text fields, password entry fields, and multiline text entry fields. Single-line text field

**<input type="text" />**

Single-line text entry control

One of the simplest types of form control is the text entry field used for entering a single word or line of text. It is added to the form using the input element with its type attribute set to text.

**<li><label for="form-city">City:</label><input type="text" name="city" value="Your Hometown" size="25" maxlength="50" id="form-city" /></li>**

The name attribute is required for identifying the variable name. The id attribute binds this control to its associated label (although it could also be referenced by style sheets and scripts). This example also includes a number of additional attributes:

- value: The value attribute specifies default text that appears in the field when the form is loaded. When you reset a form, it returns to this value.
- Size : By default, browsers display a text-entry box that is 20 characters wide but you can change the number of characters using the size attribute.
- Maxlength : By default, users can type an unlimited number of characters in a text field regardless of its size (the display scrolls to the right if the text exceeds the character width of the box). You can set a maximum character limit using the maxlength attribute if the forms processing program you are using requires it.

## Password text entry field

**<input type="password" />**

Password text control

A password field works just like a text entry field, except the characters are obscured from view using asterisk (\*) or bullet (•) characters, or another character determined by the browser.

It's important to note that although the characters entered in the password field are not visible to casual onlookers, the form does not encrypt the information, so it should not be considered a real security measure.

**<li><label for="form-pswd">Log in:</label><input type="password" name="pswd" size="8" maxlength="8" id="form-pswd" /></li>**

## Multiline text entry field

**<textarea>...</textarea>**

Multiline text entry control

At times, you'll want your users to be able enter more than just one line of text. For these instances, use the textarea element that is replaced by a multi-line, scrollable text entry box when displayed by the browser

**<li>  
<label for="form-entry">Official contest entry:</label><br />  
<textarea name="contest\_entry" rows="5" cols="100" id="form-entry">  
Tell us why you love the band in 50 words or less. Five winners will get  
backstage passes!</textarea></li>**

- Official contest entry:

```
Tell  
us why you love the band in 50 words or less. Five winners will get  
backstage passes!
```

Unlike the empty input element, the textarea element has content between its opening and closing tags. The content of the textarea element is the initial content of the text box when the form is displayed in the browser. In addition to the required name attribute, the textarea element uses the following attributes:

- ▲ rows : Specifies the number of lines of text the area should display. Scrollbars will be provided if the user types more text than fits in the allotted space.
- ▲ Cols : Specifies the width of the text area measured in number of characters

## **Submit and reset buttons**

There are a number of different kinds of buttons that can be added to web forms. The most fundamental is the submit button. When clicked, the submit button immediately sends the collected form data to the server for processing. The reset button returns the form controls to the state they were in when the form loaded.

Both submit and reset buttons are added using the input element. As mentioned earlier, because these buttons have specific functions that do not include the entry of data, they are the only form control elements that do not require the name attribute.

```
<input type="submit" />
```

Submits the form data to the server

```
<input type="reset" />
```

Resets the form controls to their default settings

Submit and reset buttons are straightforward to use. Just place them in the appropriate place in the form, in most cases, at the very end. By default, the submit button displays with the label “Submit” or “Submit Query” and the reset button is labeled “Reset.” Change the text on the button using the value attribute.

```
<p><input type="submit" /><input type="reset" value="Start over" /></p>
```

## **Radio and checkbox buttons**

Both checkbox and radio buttons make it simple for your visitors to choose from a number of provided options. They are similar in that they function like little on/off switches that can be toggled by the user and are added using the input element.

A form control made up of a collection of radio buttons is appropriate when only one option from the group is permitted, or, in other words, when the selections are mutually exclusive (such as Yes or No, or Male or Female). When one radio button is “on,” all of the others must be “off,” sort of the way buttons used to work on old radios—press one button in and the rest

pop out.

When checkboxes are grouped together, however, it is possible to select as many or as few from the group as desired. This makes them the right choice for lists in which more than one selection is okay.

## **Radio buttons**

```
<input type="radio" />
```

Radio button

Radio buttons are added to a form with the input element with the type attribute set to radio. The name attribute is required. Here is the syntax for a minimal radio button:

```
<input type="radio" name="variable" />
```

In this example, radio buttons are used as an interface for users to enter their age group.

```
<fieldset>
<legend>How old are you?</legend>
<ol>
<li><label><input type="radio" name="age" value="under24"
checked="checked" /> under 24</label></li>
<li><label><input type="radio" name="age" value="25-34" /> 25 to 34
</label></li>
<li><label><input type="radio" name="age" value="35-44" /> 35 to 44
</label></li>
<li><label><input type="radio" name="age" value="over45" /> 45+
</label></li>
</ol>
</fieldset>
```

## Output

How old are you? —

- ☒ under 24
- ☐ 25 to 34
- ☐ 35 to 44
- ☐ 45+

Notice that all of the input elements have the same variable name ("age"), but their values are different. Because these are radio buttons, only one button can be checked at a time, and therefore, only one value will be sent to the server for processing when the form is submitted. You can decide which button is checked when the form loads by adding the checked attribute to the input element.

## Checkbox buttons

```
<input type="checkbox" />
```

Checkbox button

Checkboxes are added using the input element with its type set to checkbox. As with radio buttons, you create groups of checkboxes by assigning them the same name value. The difference, as we've already noted, is that more than one checkbox may be checked at a time. The value of every checked button will be sent to the server when the form is submitted.

```
<fieldset>
<legend>What type of music do you listen to?</legend>
<ul>
<li><label><input type="checkbox" name="genre" value="punk"
checked="checked" /> Punk rock</label></li>
<li><label><input type="checkbox" name="genre" value="indie"
checked="checked" /> Indie rock</label></li>
<li><label><input type="checkbox" name="genre" value="techno" />
Techno </label></li>
<li><label><input type="checkbox" name="genre" value="rockabilly" />
Rockabilly</label></li>
</ul>
</fieldset>
```

## Output:-

What type of music do you listen to? —

- ☒ Punk rock
- ☒ Indie rock
- ☐ Techno
- ☐ Rockabilly

Checkboxes don't necessarily need to be used in groups, of course. In this example, a single checkbox is used to allow visitors to opt in for special promotions. The value of the control will only be passed along to the server if the user checks the box.

```
<p><input type="checkbox" name="optin" value="yes" /> Yes, send me news and special
promotions by email.</p>
```



## Menus

Another option for providing a list of choices is to put them in a pull-down or scrolling menu. Menus tend to be more compact than groups of buttons and checkboxes.

```
<select>...</select>
```

Menu control

```
<option>...</option>
```

An option within a menu

```
<optgroup>...</optgroup>
```

A logical grouping of options within a menu


You add both pull-down and scrolling menus to a form with the `select` element. Whether the menu pulls down or scrolls is the result of how you specify its size and whether you allow more than one option to be selected. Let's take a look at both menu types

### Pull-down menus

The `select` element displays as a pull-down menu by default when no size is specified or if the size attribute is set to 1. In pull-down menus, only one item may be selected. Here's an example:

```
<label for="form-fave">What is your favorite 80s band?</label><br />
<select name="EightiesFave" id="form-fave">
  <option>The Cure</option>
  <option>Cocteau Twins</option>
  <option>Tears for Fears</option>
  <option>Thompson Twins</option>
  <option value="EBTG">Everything But the Girl</option>
  <option>Depeche Mode</option>
  <option>The Smiths</option>
  <option>New Order</option>
</select>
```

Output



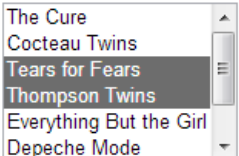
You can see that the `select` element is just a container for a number of option elements. The content of the chosen option element is what gets passed to the web application when the form is submitted. If for some reason you want to send a different value than what appears in the menu, use the `value` attribute to provide an overriding value.

### Scrolling menus

To make the menu display as a scrolling list, simply specify the number of lines you'd like to be visible using the `size` attribute. This example menu has the same options as the previous one, except it has been set to display as a scrolling list that is six lines tall.

```
<label for="EightiesBands">What 80s bands did you listen to?</label>
<select name="EightiesBands" size="6" multiple="multiple"
  for="EightiesBands">
  <option>The Cure</option>
  <option>Cocteau Twins</option>
  <option selected="selected">Tears for Fears</option>
  <option selected="selected">Thompson Twins</option>
  <option value="EBTG">Everything But the Girl</option>
  <option>Depeche Mode</option>
  <option>The Smiths</option>
  <option>New Order</option>
</select>
```

Output:-



You may notice a few new attributes tucked in there. The `multiple` attribute allows users to make more than one selection from the scrolling list. Note that pull-down menus do not allow multiple selections; when the browser detects the `multiple` attribute, it displays a small scrolling menu automatically by default.

Use the `selected` attribute in an option element to make it the default value for the menu control. Selected options are highlighted when the form loads. The `selected` attribute can be used with pull-down menus as well.

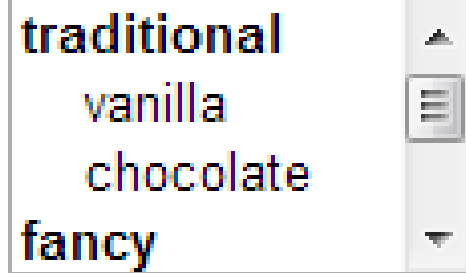
### Grouping menu options

You can use the `optgroup` element to create conceptual groups of options. The required label

attribute in the optgroup element provides the heading for the group.

```
<select name="icecream" multiple="multiple">
<optgroup label="traditional">
<option>vanilla</option>
<option>chocolate</option>
</optgroup>
<optgroup label="fancy">
<option>Super praline</option>
<option>Nut surprise</option>
<option>Candy corn</option>
</optgroup>
</select>
```

Output



## Cascading Style Sheet Orientation

Cascading Style Sheets (CSS) is the W3C standard for defining the presentation of documents written in HTML, XHTML, and, in fact, any XML language. Presentation, again, refers to the way the document is displayed or delivered to the user, whether on a computer screen, a cell phone display, or read aloud by a screen reader. With style sheets handling the presentation, (X)HTML can get back to the business of defining document structure and meaning, as intended.

CSS is a separate language with its own syntax.

### The Benefits of CSS

Not that you need further convincing that style sheets are the way to go, but here is a quick run-down of the benefits of using style sheets.

- Better type and layout controls. Presentational (X)HTML never gets close to offering the kind of control over type, backgrounds, and layout that is possible with CSS.
- Less work. You can change the appearance of an entire site by editing one style sheet. Making small tweaks and even entire site redesigns with style sheets is much easier than when presentation instructions are mixed in with the markup.
- Potentially smaller documents and faster downloads. Old school practices of using redundant font elements and nested tables make for bloated documents. Not only that, you can apply a single style sheet document to all the pages in a site for further byte savings.
- More accessible sites. When all matters of presentation are handled by CSS, you can mark up your content meaningfully, making it more accessible for no visual or mobile devices.
- Reliable browser support. Nearly every browser in current use supports all of CSS Level 1 and the majority of CSS Level 2. (See the sidebar, meet the Standards, for what is meant by CSS "levels.")

### How Style Sheets Work

It's as easy as 1-2-3!

- Start with a document that has been marked up in HTML or XHTML.
- Write style rules for how you'd like certain elements to look.
- Attach the style rules to the document. When the browser displays the document, it follows your rules for rendering elements (unless the user has applied some mandatory styles, but we'll get to that later).

### Writing the CSS rules

A style sheet is made up of one or more style instructions (called rules) that describe how an element or group of elements should be displayed. The first step in learning CSS is to get familiar with the parts of a rule. As you'll see, they're fairly intuitive to follow. Each rule selects an element and declares how it should look.

The following example contains two rules. The first makes all the h1 elements in the document green; the second

Specifies that the paragraphs should be in a small, sans serif font.

```
h1 { color: green; }
```

```
p { font-size: small; font-family: sans-serif; }
```

## Selectors

In the previous small style sheet example, the h1 and p elements are used as selectors. This most basic type of selector is called an element type selector. The properties defined for each will apply to every h1 and p element in the document, respectively.

## Declarations

The declaration is made up of a property/value pair. There can be more than one declaration in a single rule; for example, the rule for p element above has both the font-size and font-family properties. Each declaration must end with a semicolon to keep it separate from the following declaration (see note). If you omit the semicolon, the declaration and the one following it will be ignored. The curly brackets and the declarations they contain are often referred to as the declaration block.

Because CSS ignores whitespace and line returns within the declaration block, authors typically write each declaration in the block on its own line, as shown in the following example. This makes it easier to find the properties applied to the selector and to tell when the style rule ends.

```
p {  
  font-size: small;  
  font-family: sans-serif;  
}
```

Note that nothing has really changed here—there is still one set of curly brackets, semicolons after each declaration, etc.. The only difference is the insertion of line returns and some character spaces for alignment.

## Attaching the styles to the document

In the previous exercise, we embedded the style sheet right in the XHTML document using the style element. That is just one of three ways that style information can be applied to an (X)HTML document. You'll get to try each of these out soon, but it is helpful to have an overview of the methods and terminology up front.

**External** style sheets. An external style sheet is a separate, text-only document that contains a number of style rules. It must be named with the .css suffix. The .css document is then linked to or imported into one or more (X)HTML documents. In this way, all the files in a web site may share the same style sheet. This is the most powerful and preferred method for attaching style sheets to content.

**Embedded** style sheets. This is the type of style sheet we worked with in the exercise. It is placed in a document using the style element and its rules apply only to that document. The style element must be placed in the head of the document and it must contain a type attribute that identifies the content of the style element as "text/css" (currently the only available value).

```
<head>  
<title>Required document title here</title>  
<style type="text/css">  
  /* style rules go here */  
</style>  
</head>
```

The style element may also include the media attribute used to target specific media such as screen, print, or handheld devices.

**Inline** styles. You can apply properties and values to a single element using the style attribute in the element itself, as shown here:

```
<h1 style="color: red">Introduction</h1>
```

To add multiple properties, just separate them with semicolons, like this:

```
<h1 style="color: red; margin-top: 2em">Introduction</h1>
```

Inline styles apply only to the particular element in which they appear. Inline styles should be avoided, unless it is absolutely necessary to override styles from an embedded or external style sheet. Inline styles are problematic in that they intersperse presentation information into the structural markup. They also make it more difficult to make changes because every style attribute must be hunted down in the source.

## The Big Concepts

There are a few big ideas that you need to get your head around to be comfortable with how Cascading Style Sheets behave. I'm going to introduce you to these concepts now so we don't have to slow down for a lecture once we're rolling through the style properties. Each of these ideas will

certainly be revisited and illustrated in more detail in the upcoming chapters.

## Inheritance

just as parents pass down traits to their children, (X)HTML elements pass down certain style properties to the elements they contain. when we styled the p elements in a small, sans-serif font, the em element in the second paragraph became small and sans-serif as well, even though we didn't write a rule for it specifically.

## Document structure

### Parents and children

The document tree becomes a family tree when it comes to referring to the relationship between elements. All the elements contained within a given element are said to be its descendants. For example, all of the h1, h2, p, em, and img elements in the document in Figure are descendants of the body element.

An element that is directly contained within another element (with no intervening hierarchical levels), is said to be the child of that element. Conversely, the containing element is the parent. For example, the em element is the child of the p element, and the p element is its parent.

All of the elements higher than a particular element in the hierarchy are its ancestors. Two elements with the same parent are siblings

When you write a font-related style rule using the p element as a selector, the rule applies to all of the paragraphs in the document as well as the inline text elements they contain. Note that the img element is excluded because font-related properties do not apply to images.

## Conflicting styles: the cascade

Ever wonder why they are called "cascading" style sheets? CSS allows you to apply several style sheets to the same document, which means there are bound to be conflicts. For example, what should the browser do if a document's imported style sheet says that h1 elements should be red, but its embedded style sheet has a rule that makes h1s purple?

The folks who wrote the style sheet specification anticipated this problem and devised a hierarchical system that assigns different weights to the various sources of style information. The cascade refers to what happens when several sources of style information vie for control of the elements on a page: style information is passed down until it is overridden by a style command with more weight.

For example, if you don't apply any style information to a web page, it will be rendered according to the browser's internal style sheet (we've been calling this the default rendering). However, if the web page's author provides a style sheet for the document, that has more weight and overrides the browser's styles. Individual users can apply their own styles to documents as well, as discussed in the Reader Style Sheets sidebar.

As we've learned, there are three ways to attach style information to the source document, and they have a cascading order as well. Generally speaking, the closer the style sheet is to the content, the more weight it is given. Embedded style sheets that appear right in the document in the style element have more weight than external style sheets. In the example that started this section, the h1 elements would end up purple as specified in the embedded style sheet, not red as specified in the external .css file that has less weight. Inline styles have more weight than embedded style sheets because you can't get any closer to the content than a style right in the element's opening tag.

To prevent a specific rule from being overridden, you can assign it "importance" with the !important indicator, as explained in the Assigning Importance sidebar. The sidebar, Style Sheet Hierarchy , provides an overview of the cascading order from general to specific.

## Specificity

Once the applicable style sheet has been chosen, there may still be conflicts; therefore, the cascade continues at the rule level. When two rules in a single style sheet conflict, the type of selector is used to determine the winner. The more specific the selector, the more weight it is given to override conflicting declarations.

## Rule order

Finally, if there are conflicts within style rules of identical weight, whichever one comes last in the list "wins." Take these three rules, for example:

```
<style type="text/css">
  p { color: red; }
  p { color: blue; }
  p { color: green; }
```

</style>

In this scenario, paragraph text will be green because the last rule in the style sheet, that is, the one closest to the content in the document, overrides the earlier ones.

## The box model

As long as we're talking about "big CSS concepts," it is only appropriate to introduce the cornerstone of the CSS visual formatting system: the box model. The easiest way to think of the box model is that browsers see every element on the page (both block and inline) as being contained in a little rectangular box. You can apply properties such as borders, margins, padding, and backgrounds to these boxes, and even reposition them on the page.

To see the elements roughly the way the browser sees them, I've written style rules that add borders around every content element in our sample article.

```
h1 { border: 1px solid blue; }
h2 { border: 1px solid blue; }
p { border: 1px solid blue; }
em { border: 1px solid blue; }
img { border: 1px solid blue; }
```

The borders reveal the shape of each block element box. There are boxes around the inline elements (em and img), as well.

Notice that the block element boxes expand to fill the available width of the browser window, which is the nature of block elements in the normal document flow. Inline boxes encompass just the characters or image they contain.

### Grouped Selectors

Hey! This is a good opportunity to show you a handy style rule shortcut. If you ever need to apply the same style property to a number of elements, you can group the selectors into one rule by separating them with commas. This one rule has the same effect as the five rules listed previously. Grouping them makes future edits more efficient and results in a smaller file size.

```
h1, h2, p, em, img { border: 1px solid blue; }
```

Now you've got two selector types in your toolbox: a simple element selector, and grouped selectors.

## FORMATTING TEXT

### The Font Properties

When I design a text document (especially for print, but also for the Web), one of the first things I do is specify the font. In CSS, fonts are specified using a little bundle of font-related properties for typeface, size, weight, and font style. There is also a shortcut property that lets you specify all of the font attributes in one fell swoop.

The nature of the Web makes specifying type tricky, if not downright frustrating, particularly if you have experience designing for print (or even formatting text in a word processing program). Because you have no way of knowing which fonts are loaded on users' machines, you can't be sure that everyone will see text in the font you've chosen. And because the default font size varies by browser and user preferences, you can't be sure how large or small the type will appear, as well. We'll address the best practices for dealing with these font challenges as we go along.

### Specifying the font name

Choosing a typeface, or font family as it is called in CSS, is a good place to start. Let's begin with the easy part: using the property font-family and its values.

font-family

Values: one or more font or generic font family names, separated by commas | inherit

Default: depends on the browser

Applies to: all elements

Inherits: yes

Use the font-family property to specify a font or list of fonts by name as shown in these examples.

```
body { font-family: Arial; }
tt { font-family: Courier, monospace; }
p { font-family: "Trebuchet MS", Verdana, sans-serif; }
```



All font names, with the exception of generic font families, must be capitalized. For example, use “Arial” instead of “arial”. Notice that font names that contain a character space (such as Trebuchet MS in the third example) must appear within quotation marks. Use commas to separate multiple font names as shown in the second and third examples.

## Font limitations

Browsers are limited to displaying fonts that are already installed on the user’s machine. So, although you may want the text to appear in Futura, if Futura is not installed on the user’s computer, the browser’s default font will be used instead.

Fortunately, CSS allows you to provide a list of back-up fonts should your first choice not be available. In the third example above, if the browser does not find Trebuchet MS, it will use Verdana, and if Verdana is not available, it will substitute some other sans-serif font.

## Generic font families

That last option, “some other sans-serif font,” bears more discussion. “Sans-serif” is just one of five generic font families that you can specify with the font-family property. When you specify a generic font family, the browser chooses an available font from that stylistic category. Generic font family names do not need to be capitalized.

### serif

Examples: Times, Times New Roman, Georgia

Serif typefaces have decorative serifs, or slab-like appendages, on the ends of certain letter strokes.

### sans-serif

Examples: Arial, Arial Black, Verdana, Trebuchet MS, Helvetica, Geneva

Sans-serif typefaces have straight letter strokes that do not end in serifs. They are generally considered easier to read on computer monitors.

### monospace

Examples: Courier, Courier New, and Andale Mono

In monospace (also called constant width) typefaces, all characters take up the same amount of space on a line. For example, a capital W will be no wider than a lowercase i. Compare this to proportional typefaces (such as the one you’re reading now) that allot different widths to different characters.

### cursive

Examples: Apple Chancery, Zapf-Chancery, and Comic Sans

Cursive fonts emulate a script or handwritten appearance. These are rarely specified for professional web pages.

### fantasy

Examples: Impact, Western, or other decorative font

Fantasy fonts are purely decorative and would be appropriate for headlines and other display type. Fantasy fonts are rarely used for web text due to cross-platform availability and legibility.

## Font specifying strategies

The best practice for specifying fonts for web pages is to start with your first choice, provide some similar alternatives, then end with a generic font family that at least gets users in the right stylistic ballpark. Here’s another example of this strategy in action. With this style rule, I specify that I’d prefer that users see all the text in Verdana, but I’d settle for Arial, or Helvetica, or, if all else fails, I’ll let the browser choose an available sans-serif font for me.

**body { font-family: Verdana, Arial, Helvetica, sans-serif; }**

Because a font will only show up if it’s on a user’s hard drive, it makes sense to specify fonts that are the most commonly available. Although there are countless fonts out there, the fact is that because licensed copies of fonts cost big bucks, most users stick with the fonts that are installed by their operating system or other applications. Font copyright also prevents designers from just making cool fonts available for download. For these reasons, web designers tend to specify fonts from the Microsoft Core Web Fonts collection. These come installed with Windows, Internet Explorer, and Microsoft Office, so it is likely that they will find their way onto all Windows and even most Apple and Linux computers. Not only are they widely available, they were designed to be legible on low-resolution computer screens. Table lists the fonts in the collection.

## Specifying font size

Use the aptly-named font-size property to specify the size of the text.



font-size

Values: lengthunit,percentage,xx-small|x-small|small|medium|large|x-large|xx-large| smaller |larger  
|inherit

Default:medium

Applies to:allelements

Inherits:yes

You can specify text in a several ways:

At a specific size using one of the CSS length units (see the sidebar, CSS Units of Measurement, for a complete list), as shown here:

```
h1 { font-size: 1.5em; }
```

When specifying a number of units, be sure the unit abbreviation immediately follows the number, with no extra character space in between:

INCORRECT `h1 { font-size: 1.5 em; } /*space before the em*/`

- As a percentage value, sized up or down from the element's default or inherited font size:  
`h1 { font-size: 150%; }`
- Using one of the absolute keywords (xx-small, x-small, small, medium, large, x-large, xx-large). On most current browsers, medium corresponds to the default font size:  
`h1 { font-size: x-large; }`
- Using a relative keyword (larger or smaller) to nudge the text larger or smaller than the surrounding text:

```
strong { font-size: larger; }
```

I'm going to cut to the chase and tell you that, despite all these options, the only acceptable values for font-size in contemporary web design are em measurements, percentage values, and keywords. These are preferred because they allow users to resize text using the text-zoom feature on their browser. This means you can size the text as you prefer it (generally smaller than the most common default 16 pixel text), but still rest assured that users can make it larger if they have special needs or preferences. While it may be tempting to specify text in actual pixel measurements, Internet Explorer (all versions) does not allow text-zoom on type sized in pixels. That means users are stuck with your 10 or 11 pixel type, even if they are unable to read it. That's a big no-no in terms of accessibility. In addition, all of the absolute units such as pt, pc, in, mm, and cm are out because they are irrelevant on computer monitors (although they may be useful for print style sheets).

## Working with keywords

Many web designers like to specify type size using one of the predefined absolute keywords: xx-small, x-small, small, medium, large, x-large, xx-large. The keywords do not correspond to particular measurements, but rather are scaled consistently in relation to one another. The default size is medium in current browsers.

The benefit of keywords is that current browsers in Standards Mode will never let text sized in keywords render smaller than 9 pixels, so they protect against illegible type (although I would still opt for Verdana for better readability).

On the downside, the size keywords are imprecise and unpredictable. For example, while most browsers scale each level up by 120%, some browsers use a scaling factor of 150%. Another notable problem is that Internet Explorer 5 and 5.5 for Windows use small as the default (not medium), meaning your text will display a lot smaller for users with those browsers. Fortunately, with the introduction of IE 7, these old versions are slowly going away. The relative keywords, larger and maller, are used to shift the size of text relative to the size of the parent element text. The exact amount of the size

Change is determined by each browser, and is out of your control. Despite that limitation, it is an easy way to nudge type a bit larger or smaller if the exact proportions are not critical.

Shows the result of this simple bit of markup (note that the inline styles were used just to keep the example compact).

<p>There are two relative keywords:  
<span style="font-size: larger">larger</span> and  
<span style="font-size: smaller">smaller</span>. They are used to  
shift the size of text relative to the parent element.</p>

## Working with percentages and ems

By far the most popular way to specify font sizes for the Web is using measurements or percentage values, or a combination of the two. Both ems and percentages are relative measurements, which means they are based on another font size, namely, the font-size of the parent element. In this example, the font-size of the h1's parent element body) is 16 pixels, so the resulting size of the h1 would be 150% of that, or 24 pixels.

```
body { font-size: 16px; }  
h1 { font-size: 150%; } /* 150% of 16 = 24 */
```

If no font-size properties have been specified, relative measurements are based on the browser's base font size, which is 16 pixels in most browsers. Of course, users can resize their base font as small or as large as they like, so there is no guaranteed starting size, only a reasonable guess.

An em is a relative unit of measurement that, in traditional typography, is based on the width of the capital letter M (thus, the name "em"). In the CSS specification, an em is calculated as the distance between baselines when the font is set without any extra space between the lines (also known as leading).

## Font weight (boldness)

After font families and size, the remaining font properties are straightforward. For example, if you want a text element to appear in bold, use the font-weight property to adjust the boldness of type.

font-weight

Values: normal|bold|bolder |lighter|100|200|300|400 |500|600|700|800|900|inherit

Default: normal

Applies to: all elements

Inherits: yes

As you can see, the font-weight property has many predefined values, including descriptive terms (normal, bold, bolder, and lighter) and nine numeric values (100 to 900) for targeting various weights of a font if they are available. Because most fonts common on the Web have only two weights, normal (or roman) and bold, the only font weight value you will use in most cases is bold. You may also use normal to make text that would otherwise appear in bold (such as strong text or headlines) appear at a normal weight.

## Font style (italics)

The font-style property affects the posture of the text, that is, whether the letter shapes are vertical (normal) or slanted (italic and oblique).

font-style

Values: normal | italic | oblique | inherit

Default: normal

Applies to: all elements

Inherits: yes

Italic and oblique are both slanted versions of the font. The difference is that the italic version is usually a separate typeface design with curved letter forms, while oblique text takes the normal font design and just slants it. The

Truth is that in most browsers, they may look exactly the same. You'll probably only use the font-style property to make text italic or to make text that is italicized by default (such as emphasized text) display as normal.

## Font Variant (Small Caps)

Some typefaces come in a "small caps" variant. This is a separate font design that uses small uppercase-style letters in place of lowercase letter designs. The one-trick-pony font-variant property is intended to allow designers to

specify such a small-caps font for text elements.

font-variant

Values: normal |small-caps|inherit

Default: normal

Applies to: allelements

Inherits: yes

In most cases, a true small caps font is not available, so browsers simulate small caps by scaling down

uppercase letters in the current font, as you'll see when we add some small caps text to the menu next. To typography sticklers, this is less than ideal and results in inconsistent stroke weights, but you may find it an acceptable option for adding variety to small amounts of text.

## Changing Text Color

You change the color of text with the color property.

color

Values: colorvalue(nameornumeric)|inherit

Default:dependsonthebrowseranduser'spreferences

Applies to: allelements

Inherits: yes

Using the color property is very straightforward. The value of the colorproperty can be one of 17 predefined color names or a numeric value describing a specific RGB color. Here are a few examples, all of which make the h1 elements in a document gray:

```
h1 { color: gray; }
h1 { color: #666666; }
h1 { color: #666; }
```

## A Few More Selector Types

So far, we've been using element names as selectors. In the last chapter, you saw how selectors can be grouped together in a comma-separated list so you can apply properties to several elements at once. Here are examples of the selectors you already know.

element selector p { color: navy; }

grouped selectors p, ul, p, td, th { color: navy; }

The disadvantage of selecting elements this way, of course, is that the property (in this case, navy blue text) will apply to every paragraph and other listed elements in the document. Sometimes, you want to apply a rule to a particular paragraph or paragraphs. In this section, we'll look at three selector types that allow us to do just that: descendant selectors, ID selectors, and class selectors.

### **Descendant selectors**

A descendant selector targets elements that are contained within (therefore descendants of) another element. It is an example of a contextual selector, because it selects the element based on its context or relation to another element. The sidebar, Other Contextual Selectors, lists some more.

Descendant selectors are indicated in a list separated by a character space. This example targets emphasized text (em) elements, but only when they appear in list items (li). Emphasized text in paragraphs and other elements would be unaffected.

**li em { color: olive; }**

Here's another example that shows how contextual selectors can be grouped in a comma-separated list, just as we saw earlier. This rule targets em elements, but only when they appear in h1, h2, and h3 headings.

**h1 em, h2 em, h3 em { color: red; }**

It is also possible to nest descendant selectors several layers deep. This example targets em elements that appear in anchors (a) in ordered lists (ol).ol a em { font-variant: small-caps; }

### **ID selectors**

The id attribute can be used with any (X)HTML element, and it is commonly used to give meaning to the generic div and span elements.

ID selectors allow you to target elements by their id values. The symbol that identifies ID selectors is the octothorpe (#), also called a hash symbol. Here is an example of a list item with an id reference.

<li id="catalog1234">Happy Face T-shirt</li>

Now you can write a style rule just for that list item using an ID selector, like so (notice the # preceding the id reference):

**li#catalog1234 { color: red; }**

Because id values must be unique in the document, it is acceptable to omit the element name. This rule is equivalent to the last one:

**#catalog1234 { color: red; }**

You can also use an ID selector as part of a contextual selector. In this example, a style is applied only

to li elements that appear within any element identified as "sidebar." In this way, you can treat list items in the sidebar differently than all the other list items on the page without any additional markup.

```
#sidebar li { margin-left: 10px; }
```

You should be beginning to see the power of selectors and how they can be used strategically along with well-planned, semantic markup.

## Class selectors

the class identifier, used to classify elements into a conceptual group. Unlike the id attribute, multiple elements may share a class name. Not only that, an element may belong to more than one class. You can target elements belonging to the same class with, you guessed it, a class selector. Class names are indicated with a period (.) in the selector. For example, to select all paragraphs with class="special", use this selector (the period indicates the following word is a class selector):

```
p.special { color: orange; }
```

To apply a property to all elements of the same class, omit the element name in the selector (be sure to leave the period; it's the character that indicates a class). This would target all paragraphs and any other element that has been marked up with class="special".

```
.special { color: orange; }
```

## Specificity 101

Specificity refers to the fact

that more specific selectors have more weight when it comes to handling style rule conflicts. Now that you know a few more selectors, it is a good time to revisit this very important concept. The actual system CSS uses for calculating selector specificity is quite complicated, but this list of selector types from most to least specific should serve you well in most scenarios.

ID selectors are more specific than (and will override) Class selectors, which are more specific than (and will override) Contextual selectors, which are more specific than (and will override) Individual element selectors. So, for example, if a style sheet has two conflicting rules for the strong element, strong { color: red; }

```
h1 strong { color: blue; }
```

the contextual selector (h1 strong) is more specific and therefore has more weight than the element selector. You can use specificity strategically to keep your style sheets simple and your markup minimal. For example, it is possible to set a style for an element (p, in this example), then override when necessary by using more specific selectors.

```
p { line-height: 1.2em; }  
blockquote p { line-height: 1em; }  
p.intro { line-height: 2em; }
```

In these examples, p elements that appear within a blockquote have a smaller line height than ordinary paragraphs. However, all paragraphs with a class of "intro" will have a 2em line height, even if it appears within a blockquote, because class selectors are more specific than contextual selectors.

## Text Line Adjustments

The next batch of text properties has to do with the treatment of whole lines of text rather than the shapes of characters. They allow web authors to format web text with indents, extra leading (space between lines), and different horizontal alignments, similar to print.

## Line height

The line-height property defines the minimum distance from baseline to baseline in text. A baseline is the imaginary line upon which the bottoms of characters sit. Line height in CSS is similar to leading in traditional typesetting. Although the line height is calculated from baseline to baseline, most browsers split the extra space above and below the text, thus centering it in the overall line height.

The line-height property is said to specify a "minimum" distance because if you put a tall image on a line, the height of that line will expand to accommodate it.

line-height Values: number, length, measurement, percentage | normal | inherit

Default: normal

Applies to: all elements

Inherits: yes

These examples show three different ways of making the line height twice the height of the font size.

```
p { line-height: 2; }  
p { line-height: 2em; }  
p { line-height: 200%; }
```

When a number is specified alone, as shown in the first example, it acts as a scaling factor that is

multiplied by the current font size to calculate the line-height value. Line heights can also be specified in one of the CSS length units, but once again, the relative em unit is your best bet. Ems and percentage values are based on the current font size.

## Indents

The text-indent property indents the first line of text by a specified amount (see note).

text-indent

Values: length measurement, percentage|inherit

Default:0

Applies to:block-level elements and table cells

Inherits:yes

You can specify a length measurement or a percentage value for text-indent. Percentage values are calculated based on the width of the parent element. Here are a few examples. The results are shown in.

```
p#1 { text-indent: 2em; }  
p#2 { text-indent: 25%; }  
p#3 { text-indent: -35px; }
```

## Horizontal Alignment

You can align text for web pages just as you would in a word processing or desktop publishing program with the text-align property.

text-align

Values: left|right|center|justify|inherit

Default: left for languages that read left to right; right for languages that read right to left;

Applies to: block-level elements and table cells

Inherits: yes

This is a fairly straightforward property to use. The results of the various text-align values are shown in.

text-align: left aligns text on the left margin

text-align: right aligns text on the right margin

text-align: center centers the text in the text block

text-align: justify aligns text on both right and left margins

Font and Text Properties

## Colors and Backgrounds

### Specifying Color Values

There are two main ways to specify colors in style sheets: with a predefined color name as we have been doing so far: color: red; color: olive; color: blue; or, more commonly, with a numeric value that describes a particular RGB color (the color model on computer monitors). You've probably seen color values that look like these: color: #FF0000; color: #808000; color: #00FF00; We'll get to all the ins and outs of RGB color in a moment, but first, a short and sweet section on the standard color names.

### Color names

The most intuitive way to specify a color is to call it by name. Unfortunately, you can't make up just any color name and expect it to work. It has to be one of the color keywords predefined in the CSS Recommendation. CSS1 and CSS2 adopted the 16 standard color names originally introduced in HTML 4.01. CSS2.1 tossed in orange for a total of 17. Color names are easy to use—

just drop one into place as the value for any color-related property:

```
color: silver;  
background-color: gray;  
border-bottom-color: teal;
```

### Writing RGB values in style sheets

CSS allows RGB color values to be specified in a number of formats. Going back to that pleasant lavender, we could add it to a style sheet by listing each value on a scale from 0 to 255.

```
color: rgb(200, 178, 230);
```

You can also list them as percentage values, although that is less common. `color: rgb(78%, 70%, 90%);` Or, you can provide the web-ready version that we saw in the Color Picker. These six digits represent the same three RGB values, except they have been converted into hexadecimal (or hex, for short) values. I'll explain the hexa-decimal system in the next section. Note that hex RGB values are preceded by the # symbol and do not require the `rgb()` notation shown above.

`color: #C8B2E6;`

There is one last shorthand way to specify hex color values. If your value happens to be made up of three pairs of double-digits, such as:

`color: #FFCC00;` or `color: #993366;`

## Background Color

Before style sheets, you could apply a background color only to the entire page. Now, with the `background-color` property, you can apply background colors to any element.

`background-color`

Values: `colorvalue(nameornumeric)` | `transparent` | `inherit`

Default: `transparent`

Applies to: all elements

Inherits: no

A background color fills the canvas behind the element that includes the content area, and any padding (extra space) added around the content, extending behind the border out to its outer edge.

```
blockquote {  
  border: 4px dashed;  
  color: #508C19;  
  background-color: #B4DBE6;  
}
```

## Anchor pseudoclasses

There are four main pseudoclasses that can be used as selectors: `a:link` Applies a style to unclicked (unvisited) links `a:visited` Applies a style to links that have already been clicked `a:hover` Applies a style when the mouse pointer is over the link `a:active` Applies a style while the mouse button is pressed

Pseudoselectors are indicated by the colon (:) character.

The `:link`, `:visited`, and `:active` pseudoselectors replace the old presentational `link`, `vlink`, and `alink` attributes, respectively, that were once used to change link colors. But with CSS, you can change more than just color. Once you've selected the link state, you can apply any of the properties we've covered so far (and more).

Let's look at an example of each. In these examples, I've written some style rules for links (`a:link`) and visited links (`a:visited`). I've used the `text-decoration` property to turn off the underline under both link states. I've also changed the color of links (blue by default) to maroon, and visited links will now be gray instead of the default purple.

```
a:link {  
  color: maroon;  
  text-decoration: none;  
}  
a:visited {  
  color: gray;  
  text-decoration: none;  
}
```

The `:hover` selector is an interesting one (see note). It allows you to do cool rollover effects on links that were once possible only with JavaScript. If you add this rule to the ones above, the links will get an underline and a back-ground color when the mouse is hovered over them, giving the user feedback



that the text is a link.

```
a:hover {  
  color: maroon;  
  text-decoration: underline;  
  background-color: #C4CEF8;  
}
```

Finally, this rule using the :active selector makes links bright red (consistent with maroon, but more intense) while the link is being clicked. This style will be displayed only for an instant, but it can give a subtle indication that something has happened.

```
a:active {  
  color: red;  
  text-decoration: underline;  
  background-color: #C4CEF8;  
}
```

## **Pseudoelement Selectors**

Pseudoclasses aren't the only kind of pseudo selectors. There are also four pseudoelements that act as though they are inserting fictional elements into the document structure for styling. Pseudoelements are also indicated with a colon (:) symbol.

First letter and line

Two of the pseudoelements are based on context and are used to select the first letter or the first line of text of an element.

:first-line

This selector applies a style rule to the first line of the specified element. The only properties you can apply, however, are:

color font background  
word-spacing letter-spacing text-decoration  
vertical-align text-transform line-height

**:first-letter**

This applies a style rule to the first letter of the specified element. The properties you can apply are limited to:

color font text-decoration  
text-transform vertical-align text-transform  
background margin padding  
border float  
letter-spacing (CSS2.1) word-spacing (CSS2.1)

## **Background Images**

CSS really beats (X)HTML hands-down when it comes to background images (but then, (X)HTML really shouldn't have been dealing in background images in the first place). With CSS, you're not stuck with a repeating tile pattern, and you can position a background image wherever you like. You can also apply a background image to any element in the document, not just the whole page.

In this section, we'll look at the collection of properties used to place and push around background images, starting with the basic background-image property.

Adding a background imageThe background-image property is used to add a background image to an element. Its primary job is to provide the location of the image file.

background-image

Values: URL(locationofimage) |none |inherit

Default: none

Applies to: all elements

Inherits: no

The value of background-image is a sort of url-holder that contains the URL of the image. The URL is relative to the (X)HTML document that the image is going into, not the style sheet document (see related Tip). These examples show background images applied behind a whole page (body) and a single blockquote element with padding and a

border applied.

```
body {
```

```
  background-image: url(star.gif); }
```

```
blockquote {
```

```
  background-image: url(dot.gif);
```

```
  padding: 2em;
```

```
  border: 4px dashed;}
```

If you provide both a background-color and a background-image to an element, the image will be placed on top of the color. In fact, it is recommended that you do provide a backup color that is similar in hue, in the event the image fails to download.

## **Background Tiling**

As we saw in the last figure, images tile up and down, left and right when left to their own devices. You can change this behavior with the background-

repeat property.

background-repeat

Values: repeat |repeat-x|repeat-y|no-repeat|inherit

Default: repeat

Applies to:allelements

Inherits: no

If you want a background image to appear just once, use the no-repeat key-word value, like this.

```
body {
```

```
  background-image: url(star.gif);
```

```
  background-repeat: no-repeat;}
```

You can also restrict the image to tiling only horizontally (repeat-x) or vertically (repeat-y) as shown in these examples.

```
body {
```

```
  background-image: url(star.gif);
```

```
  background-repeat: repeat-x;
```

```
}
```

```
body {
```

```
  background-image: url(star.gif);
```

```
  background-repeat: repeat-y;
```

```
}
```

## **Background Position**

The background-position property specifies the position of the origin image in the background. You can think of the origin image as the first image that is placed in the background from which tiling images extend. Here is the property and its various values.

background-position

Values: length measurement| percentage|left|center|right|top|bottom| inherit

Default: 0% 0% (same as left top)

Applies to: all elements

Inherits: no

## Keyword positioning

The keyword values (left, center, right, top, bottom, and center) position the origin image relative to the edges of the element. For example, left positions the image all the way to the left edge of the background area. Keywords are typically used in pairs, as in these examples:

```
{ background-position: left bottom; }
```

```
{ background-position: right center; }
```

If you provide only one keyword, the missing keyword is assumed to be center. Thus, background-position: right has the same effect as the second example above.

## Length measurements

You can also specify the position by its distance from the top-left corner of the element using pixel measurements. When providing length values, the horizontal measurement always goes first.

```
{ background-position: 200px 50px; }
```

## Percentages

Percentage values are provided in horizontal/vertical pairs, with 0% 0% corresponding to the top-left corner and 100% 100% corresponding to the bottom-right corner. It is important to note that the percentage value applies to both the canvas area and the image itself. For example, the 100% value places the bottom-right corner of the image in the bottom-right corner of the canvas. As with keywords, if you only provide one percentage, the other is assumed to be 50% (centered).

```
{ background-position: 15% 100%; }
```

## Background attachment

In the previous exercise, I asked you to scroll the page and watch what happens to the background image. As expected, it scrolls along with the document and off the top of the browser window, which is its default behavior. However, you can use the background-attachment property to free the back-

ground from the content and allow it to stay fixed in one position while the rest of the content scrolls. background-attachment

Values: scroll | fixed | inherit

Default: scroll

Applies to: all elements

Inherits: no

With the background-attachment property, you have the choice of whether the background image scrolls or is fixed. When an image is fixed, it stays in the same position relative to the viewing area of the browser.

## external style sheet

There are two ways to refer to an external style sheet from within the (X)HTML document: the link element and an @import rule. Let's look at both of these attachment methods. Using the link element The best-supported method is to create a link to the .css document using the link element in the head of the document, as shown here:

```
<head>
<link rel="stylesheet" href="/path/stylesheet.css" type="text/css" />
<title>Titles are required.</title>
</head>
```

You need to include three attributes in the link element:

rel="stylesheet"

Defines the linked document's relation to the current document. The value of the rel attribute is always stylesheet when linking to a style

sheet.

href="url"

Provides the location of the .css file.type="text/css"

This identifies the data (MIME) type of the style sheet as "text/css" (currently the only option).

You can include multiple link elements to different style sheets and they'll all apply. If there are conflicts, whichever one is listed last will override previous settings due to the rule order and the cascade.

## **Importing with @import**

The other method for attaching external style sheets to a document is to import it with an @import rule in the style element, as shown in this exam-

ple:

```
<head>
<style type="text/css">
  @import url("http://path/stylesheet.css");
  p { font-face: Verdana;}
</style>
<title>Titles are required.</title>
</head>
```

In this example, an absolute URL is shown, but it could also be a relative URL (relative to the current (X)HTML document). The example above shows that an @import rule can appear in a style element with other rules, but it must come before any selectors. Again, you can import multiple style sheets and they all will apply, but style rules from the last file listed takes precedence over earlier ones. You can also use the @import function within a .css document to reference other .css documents. This lets you pull style information in from other style sheets.

## Thinking inside the box

(Padding, Borders, and Margins)

### **The Element Box**

#### **content area**

At the core of the element box is the content itself.

#### **inner edges**

The edges of the content area are referred to as the inner edges of the element box. This is the box that gets sized when you apply width and height properties.

#### **padding**

The padding is the area held between the content area and an optional border. In the diagram, the padding area is indicated by a yellow-orange color. Padding is optional.

#### **border**

The border is a line (or stylized line) that surrounds the element and its padding. Borders are also optional.

#### **margin**

The margin is an optional amount of space added on the outside of the border. In the diagram, the margin is indicated with light-blue shading, but in reality, margins are always transparent, allowing the background of the parent element to show through.

#### **outer edge**

The outside edges of the margin area make up the outer edges of the element box. This is the total area the element takes up on the page, and it includes the width of the content area plus the total amount of padding, border, and margins applied to the element. The outer edge in

the dia-gram is indicated with a dotted line, but in real web pages, the edge of the margin is invisible.

## **Specifying height**

In general practice, it is less common to specify the height of elements. It is more in keeping with the nature of the medium to allow the height to be calculated automatically, based on the size of the text and other contents. This

allows it to change based on the font size, user settings, or other factors. If you do specify a height for an element containing text, be sure to also consider what happens should the content not fit. Fortunately, CSS gives you some

options, as we'll see in the next section.

## **Handling overflow**

When an element is set to a size that is too small for its contents, it is possible to specify what to do with the content that doesn't fit, using the overflow property.overflow

Values: visible|hidden|scroll |auto| inherit

Default: visible

Applies to:Block-level elements and replaced inline elements(such as images)

Inherits: no

### **visible**

The default value is visible, which allows the content to hang out over the element box so that it all can be seen.

### **hidden**

When overflow is set to hidden, the content that does not fit gets clipped off and does not appear beyond the edges of the element's content area.

### **scroll**

When scroll is specified, scrollbars are added to the element box to let users scroll through the content. Be aware that when you set the value to scroll, the scrollbars will always be there, even if the content fits in the specified height just fine.

### **auto**

The auto value allows the browser to decide how to handle overflow. In most cases, scrollbars are added only when the content doesn't fit and they are needed.

## **Padding**

Padding is the space between the content area and the border (or the place the border would be if one isn't specified). I find it helpful to add a little padding to elements when using a background color or a border. It gives the content a little breathing room, and prevents the border or edge of the background from bumping right up against the text. You can add padding to the individual sides of any element (block-level or inline). There is also a shorthand padding property that lets you add padding on all sides at once.padding-top, padding-right, padding-bottom, padding-leftValues: length

measurement| percentage|auto|inherit

Default: auto

Applies to:all elements

Inherits: no

padding

Values: lengthmeasurement| percentage|auto|inherit

Default: auto

Applies to:all elements

Inherits: no

## **Border**

The authors of CSS didn't skip when it came to border shortcuts. They also created properties for providing style, width, and color values in one declaration. Again, you can specify the appearance of specific sides, or use the border

Property to change all four sides at once. Border-top, border-right, border-bottom, border-left Values: border-style, border-width, border-color or inherit Default: defaults for each property Applies to: all elements Inherits: no border Values: border-style border-width border-color or inherit

Default: defaults for each property Applies to: all elements Inherits: no The values for border and the side-specific border properties may include style, width, and color values in any order. You do not need to declare all three, but keep in mind that if the border style value is omitted, no border will render. The border shorthand property works a bit differently than the other short-hand properties that we covered in that it takes one set of values and always applies them to all four sides of the element. In other words, it does not use the clockwise, "TRBL" system that we've seen with other shorthand properties. Here is a smattering of valid border shortcut examples to get an idea for how

they work.

```
h1 { border-left: red .5em solid; } left border only
h2 { border-bottom: 1px solid; } bottom border only
p.example { border: 2px dotted #663; } all four sides
```

## Margins

The side-specific and shorthand margin properties work much like the padding properties we've looked at already, however, margins have some special behaviors to be aware of. margin-top, margin-right, margin-bottom, margin-left

Values: length measurement | percentage | auto | inherit

Default: auto

Applies to: all elements Inherits: no margin

Values: length measurement percentage auto inherit Default: auto Applies to: all elements Inherits: no

The margin properties are very straightforward to use. You can specify an amount of margin to appear on each side of the element, or use the margin property to specify all sides at once.

## Assigning Display Roles

As long as we're talking about boxes and the CSS layout model, this is a good time to introduce the display property. You should already be familiar with the display behavior of block and inline elements in (X)HTML. However, not all XML languages assign default display behaviors (or display roles, to use the proper term from the CSS specification) to the elements they contain. For

this reason, the display property was created to allow authors to specify how elements should behave in layouts.

display

Values: inline | block | list-item | run-in | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit

Default: disc

Applies to: ul, ol, and li (or elements whose display value is list-item)

Inherits: yes

The display property defines the type of element box an element generates in the layout. In addition to the familiar inline and block display roles, you can also make elements display as list items or the various parts of a table. Another useful value for the display property is none, which removes the content from the normal flow entirely. Unlike visibility: hidden, which just makes the element invisible, but holds the space it would have occupied blank, display: none removes the content, and the space it would have occupied is closed up. One popular use of display: none is to prevent certain content from appearing when the source document is displayed in specific media. For example, you could have a paragraph that appears when the document is printed, but is not part of the page when it is displayed



on a computer screen.

## Floating and Positioning

### Normal Flow

In the CSS layout model, text elements are laid out from top to bottom in the order in which they appear in the source, and from left to right (in left-to-right reading languages\*). Block elements stack up on top of one another and fill the available width of the browser window or other containing element. Inline elements and text characters line up next to one another to fill the block elements. When the window or containing element is resized, the block elements expand or contract to the new width, and the inline content reflows to fit.

Objects in the normal flow affect the layout of the objects around them. This is the behavior you've come to expect in web pages—elements don't overlap or bunch up, they make room for one another. We've seen all of this before, but in this chapter we'll be paying attention to whether elements are in the flow or removed from the flow. Floating and positioning changes the relationship of elements to the normal flow in different ways. Let's first look at the special behavior of floated elements (or

"floats" for short)

### Floating

Simply stated, the float property moves an element as far as possible to the left or right, allowing the following content to wrap around it. It is not a positioning scheme per se, but a unique feature built into CSS with some interesting behaviors. Floats are one of the primary tools of modern CSS-based web design, used to create multicolumn layouts, navigation toolbars from lists, and table-like alignment without tables. It's exciting stuff. Let's start with the float property itself.

float

Values: left|right|none|inherit

Default: none

Applies to: all elements

Inherits: no

The best way to explain floating is to demonstrate it. In this example, the float property is applied to an `img` element to float it to the right.

### The markup

```
<p>They went down, down,...</p>
```

### The style sheet

```
img {
  float: right;
}
p {
  padding: 15px;
  background-color: #FFF799;
  border: 2px solid #6C4788;
}
```

### The style sheet

```
span.disclaimer {
  float: right;
  margin: 10px;
  width: 200px;
  color: #FFF;
  background-color: #9D080D;
  padding: 4px;
}
p {
  padding: 15px;
```

```
background-color: #FFF799;  
border: 2px solid #6C4788;}
```

From the looks of things, it is behaving just the same as the floated image, which is what we'd expect. But there are some subtle things at work here that bear pointing out.

## **Always provide a width for floated text elements.**

First, you'll notice that the style rule that floats the span includes the width property. In fact, it is necessary to specify a width for floated text elements because without one, the content area of the box expands to its widest possible width (or, on some browsers, it may collapse to its narrowest possible width). Images have an inherent width, so we didn't need to specify a width

in the previous example (although we certainly could have).

## **Floated inline elements behave as block elements.**

Notice that the margin is held on all four sides of the floated span text, even though top and bottom margins are usually not rendered on inline elements. That is because all floated elements behave like block elements. Once you float an inline element, it follows the display rules for block-level elements, and margins are rendered on all four sides. Margins on floated elements do not collapse, however.

## **Floating block elements**

Let's look at what happens when you float a block within the normal flow. In this example, a whole paragraph element is floated to the left. The markup:

```
<p>ONCE upon a time....</p>  
<p id="float">As he had a white skin, blue eyes,...</p>  
<p>The fact was he thought them very ugly...</p>  
The style sheet:  
p#float {  
  float: left;  
  width: 200px;  
  margin-top: 0px;  
  background: #A5D3DE;  
}  
p {  
  border: 1px solid red;  
}
```

## **Clearing floated elements**

The last thing you need to know about floated elements is how to turn the text wrapping off and get back to layout as usual. This is done by clearing the element that you want to start below the float. Applying the clear property to an element prevents it from appearing next to a floated element, and forces it to start against the next available "clear" space below the float.

Clear Values: left|right|both |none |inherit

Default: none

Applies to: block-level elements only

Inherits: no

Keep in mind that you apply the clear property to the element you want to start below the floated element, not the floated element itself. The left value starts the element below any elements that have been floated to the left. Similarly, the right value makes the element clear all floats on the right

edge of the containing block. If there are multiple floated elements, and you want to be sure an element starts below all of them, use the both value to clear floats on both sides.

```
img {  
  float: left;  
  margin-right: 10px;  
}  
h2 {  
  clear: left;  
  margin-top: 2em;  
}
```

## Positioning Basics

CSS provides several methods for positioning elements on the page. They can be positioned relative to where they would normally appear in the flow, or removed from the flow altogether and placed at a particular spot on the page. You can also position an element relative to the browser window (technically known as the viewport in the CSS Recommendations) and it will stay put while the rest of the page scrolls. Unfortunately, not all positioning methods are well supported, and inconsistent and buggy browser implementation can make it challenging to achieve the results you're after. The best thing to do is get acquainted with the way positioning should work according to the specification, as we'll do in the following sections, starting with the basic position property.

### Types of positioning

position

Values: static | relative | absolute | fixed | inherit

Default: static

Applies to: all elements

Inherits: no

The position property indicates that an element is to be positioned, and specifies which positioning method should be used. I'll introduce each key-word value briefly here, then we'll take a more detailed look at each method in the remainder of this chapter.

#### static

This is the normal positioning scheme in which elements are positioned as they occur in the normal document flow.

#### relative

Relative positioning moves the box relative to its original position in the flow. The distinctive behavior of relative positioning is that the space the element would have occupied in the normal flow is preserved.

#### absolute

Absolutely positioned elements are removed from the document flow entirely and positioned relative to a containing element (we'll talk more about this later). Unlike relatively positioned elements, the space they would have occupied is closed up. In fact, they have no influence at all on the layout of surrounding elements.

#### fixed

The distinguishing characteristic of fixed positioning is that the element stays in one position in the window even when the document scrolls. Fixed elements are removed from the document flow and positioned relative to the browser window (or other viewport). Rather than another element in the document.

### Specifying position

Once you've established the positioning method, the actual position is specified with four offset properties.

top, right, bottom, left Values: length measurement | percentage | auto | inherit Default:

auto Applies to: Positioned elements (where position value is relative, absolute, or fixed) Inherits: no The values provided for each of the offset properties defines the distance the

Element should be moved away from that respective edge. For example, the value of top defines the distance the top outer edge of the positioned element should be offset from the top edge of the browser or other containing element. A positive value for top results in the element box moving down by that amount. Similarly, a positive value for left would move the positioned

Element to the right (toward the center of the containing block) by that amount. Further explanations and examples of the offset properties will be provided in the discussions of each positioning method. We'll start our exploration of positioning with the fairly straightforward relative method.

### Relative Positioning

As mentioned previously, relative positioning moves an element relative to its original spot in the flow.

The space it would have occupied is preserved and continues to influence the layout of surrounding content. This is easier to

Understand with a simple example. Here I've positioned an inline `em` element (a background color makes its boundaries apparent). First, I used the `position` property to set the method to `relative`, then I used the `top` offset property to move the element 30 pixels down from its initial position, and the `left` property to move it 60 pixels to the right. Remember, offset property values move the element away from the specified edge, so if you want something to move to the right, as I did here,

you use the `left` offset property.

```
em {  
  position: relative;  
  top: 30px;  
  left: 60px;  
  background-color: fuchsia;  
}
```

I want to point out a few things that are happening here.

**The original space in the document flow is preserved.**

You can see that there is a blank space where the emphasized text would have been if the element had not been positioned. The surrounding content is laid out as though the element were still there, therefore we say that the element still "influences" the surrounding content.

**Overlap happens.**

Because this is a positioned element, it can potentially overlap other elements.

**Absolute Positioning**

Absolute positioning works a bit differently and is actually a more flexible method for achieving page layouts than relative positioning.

```
em {  
  position: absolute;  
  top: 30px;  
  left: 60px;  
  background-color: fuchsia;  
}
```

What actually happens in absolute positioning is that the element is positioned relative to its nearest containing block. It just so happens that the nearest containing block in is the root (`html`) element (also known as the initial containing block), so the offset values position the `em` element relative to the whole browser window area. Getting a handle on the containing block concept is the first step to taking on absolute positioning

**Containing blocks**

The CSS2.1 Recommendation states, "The position and size of an element's box(es) are sometimes calculated relative to a certain rectangle, called the containing block of the element." It is critical to have an awareness of the containing block of the element you want to position.

CSS2.1 lays out a number of intricate rules for determining the containing

block of an element, but it basically boils down to this:

- If the positioned element is not contained within another positioned element, then it will be placed relative to the initial containing block (created by the `html` element).
- But if the element has an ancestor (i.e. is contained within an element) that has its position set to `relative`, `absolute`, or `fixed`, the element will be positioned relative to the edges of that element instead.

Turn the `p` element into a containing block and see what happens. All we have to do is apply the `position` property to it; we don't have to actually move it. The most common way to make an element into a containing block is to set the position to `relative`, but don't move it with offset values. (By the way, this is what I was talking about earlier when I said that relative positioning is most often used to create a context for an absolutely

Positioned element.) We'll keep the style rule for the em element the same, but we'll add a position property to the p element, thus making it the containing block for the positioned em element.

```
p {
  position: relative;
  padding: 15px;
  background-color: #DBFDBA;
  border: 2px solid #6C4788;
}
em {
  width: 200px; margin: 25px;
  position: absolute;
  top: 30px; left: 60px;
  background-color: fuchsia;
}
```

Here we can see that:

- ▲ The offset values apply to the outer edges of the element box (from margin edge to margin edge), and
- ▲ Absolutely positioned elements always behave as block-level elements. For example, the margins on all sides are maintained even though this is an inline element. It also permits a width to be set for the element.

## Pixel measurements

```
div#a {
  position: relative; /* creates the containing block */
  height: 120px;
  width: 300px;
  border: 1px solid;
  background-color: #CCC;
}
div#b {
  position: absolute;
  top: 20px;
  right: 30px;
  bottom: 40px;
  left: 50px;
  border: 1px solid;
  background-color: teal;
}
```

## Percentage values

```
img#A {
  position: absolute;
  top: 50%;
  left: 0%; /* the % symbol could be omitted for a 0 value */
}
img#B {
  position: absolute;
  bottom: 0%; /* the % symbol could be omitted for a 0 value */
  right: 0%; /* the % symbol could be omitted for a 0 value */
}
```

## Stacking order

Before we close the book on absolute positioning, there is one last related concept that I need to introduce. As we've seen, absolutely positioned elements overlap other elements, so it follows that multiple positioned elements have the potential to stack up on one another. By default, elements stack up in the order in which they appear in the document, but you can change the stacking order with the z-index property. Picture the z-axis as a line that runs perpendicular to the page, as though from the tip of your nose, through this page, and out the other side.

Values: (number)		auto		inherit
Default	:	auto		
Applies to	:	positioned	elements	
Inherits	:	no		

The value of the z-index property is a number (positive or negative). The higher the number, the higher the element will appear in the stack. Lower numbers and negative values move the element lower in the stack.

Here are three paragraph elements, each containing a letter image (A, B, and C, respectively) that have been positioned in a way that they overlap on the page. By default, paragraph "C" would appear on top because it appears last in the source. However, by assigning higher z-index values to paragraphs "A" and "B," we can force them to stack in our preferred order. Note that the values of z-index do not need to be sequential, and they do not relate to anything in particular. All that matters is that higher number values position the element higher in the stack.

The markup:

```
<p id="A"></p>
<p id="B"></p>
<p id="C"></p>
```

The style sheet:

```
#A {
  z-index: 10;
  position: absolute;
  top: 200px;
  left: 200px;
}
```

```
#B { z-index: 5;
     position: absolute;
     top: 225px; left: 175px;
}
#C { z-index: 1;
     position: absolute;
     top: 250px; left: 225px;
}
```

## Fixed Positioning

We've covered relative and absolute positioning, now it's time to take on the remaining method: fixed positioning. For the most part, fixed positioning works just like absolute positioning. The significant difference is that the offset values for fixed elements are always relative to the browser window (or other viewport), which means the positioned element stays put even when the rest of the page scrolls.

```
#award {
  position: fixed;
  top: 35px;
  left: 25px;
}
```

## Page layout with css

Now that you understand the principles of moving elements around on the page using CSS floats and positioning, we can put these tools to use in some standard page layouts. This chapter looks at the various approaches to CSS-based web design and provides some simple templates that will get you on your way to building basic two- and three-column web pages.



## Page Layout Strategies

### Liquid page design

Liquid page layouts (also called fluid layouts) follow the default behavior of the normal flow. In other words, the page area and/or columns within the page are allowed to get wider or narrower to fill the available space in the browser window. There is no attempt to control the width of the content or line breaks—the text is permitted to reflow as required.

Proponents of liquid web pages feel strongly that this is the best formatting method because it is consistent with the nature of the medium. Of course, it has both advantages and disadvantages.

#### **Advantages**

You don't have to design for a specific monitor resolution. You avoid potentially awkward empty space because the text fills the window. Liquid pages keep with the spirit and nature of the medium.

#### **Disadvantages**

On large monitors, line lengths can get very long and uncomfortable to read. See the sidebar for more information.

They are less predictable. Elements may be too spread out or too cramped at extreme browser dimensions.

#### **How to create liquid layouts**

Create a liquid layout by specifying widths in percentage values. You can also not specify a width at all, in which case the width will be set to the default auto setting, and the element will fill the available width of the window or other containing element. Here are a few examples.

```
div#main {  
  width: 70%;  
  margin-right: 5%;  
  float: left;  
  background: yellow;  
}  
div#extras {  
  width: 25%;  
  float: left;  
  background: orange; }
```

the secondary column on the left is set to a specific pixel width, and the main content area is set to auto and fills the remaining space in the window

```
div#main {  
  width: auto;  
  position: absolute;  
  top: 0;  
  left: 225px;  
  background: yellow; }  
div#extras {  
  width: 200px;  
  position: absolute;  
  top: 0;  
  left: 0;  
  background: orange;  
}
```

## Dealing with line lengths

Although long line lengths are possible in liquid layouts, it's certainly a manageable situation and not a reason to reject this layout approach. In the vast majority of cases, users have their browsers sized reasonably, that is, somewhere between 800 and 1250 pixels. If your page is two or more columns, you're in luck, because it will be difficult for the line lengths to get too out of hand at these "reasonable" browser widths. Sure, line lengths will change when the browser is resized, and they may not be in the ideal range of 55 to 72 characters per line, but text is unlikely to be unreadable. If your page consists of only one column, I suggest using left and right margins (in the 10 to 20% range, depending on preference) to reduce the resulting line length and also add valuable white space around the text. Finally, you can use the max-width property to limit the width of the content

Containers. Unfortunately, it is not supported by Internet Explorer (Windows) 6 and earlier, but those versions will eventually fall out of significant use.

### Fixed Layouts

Fixed-width layouts, as the name implies, stay put at a specified pixel width as determined by the designer. This approach is based on traditional guiding principles of graphic design, such as a constant grid, the relationship of page elements, and comfortable line lengths. When you set your page to a specific width, you need to decide a couple of things. First, you need to pick a page width, usually based on common monitor resolutions. Most fixed-width web pages as of this writing are designed to fit in an 800 x 600 pixel browser window, although more and more sites are venturing into a (roughly) 1000 pixel page width. You also need to decide where the fixed-width layout should be positioned in the browser window. By default, it stays on the left edge of the browser, with the extra space to the right of it. Some designers opt to center the page, splitting the extra space over left and right margins, which may make the page look as though it better fills the browser window.

```
#wrapper {width: 750px;
    position: absolute;
    margin-left: auto;
    margin-right: auto;
    border: 1px solid black;
    padding: 0px;}
#extras {position: absolute;
    top: 0px;
    left: 0px;
    width: 200px;
    background: orange; }
#main {margin-left: 225px;
    background-color: yellow;}
```

One of the main concerns with using fixed-width layouts is that if the user's browser window is not as wide as the page, the content on the right edge of the page will not be visible. Although it is possible to scroll horizontally, it may not always be clear that there is more content there in the first place.

### **Advantages**

The layout is predictable. It offers better control over line length. Trends come and go; however, it is worth noting that many of the most well-known web designers use fixed-width designs as of this writing.

### **Disadvantages**

Content on the right edge will be hidden if the browser window is smaller than the page. Text elements still reflow if the user resizes the font size, so it doesn't guarantee the layout will stay exactly the same. Line lengths may grow awkwardly short at very large text sizes.

### **How to create fixed-width layouts**

Fixed-width layouts are created by specifying width values in pixel units. Typically, the content of the

entire page is put into a div (often named “con-tent,” “container,” “wrapper,” or “page”) that can then be set to a specific pixel width. This div may also be centered in the browser window. Widths of column elements, and even margins and padding, are also specified in pixels.

## **Elastic Layouts**

A third layout approach is growing in popularity because it marries resizable text with predictable line lengths. Elastic (also called jello) layouts expand or contract with the size of the text. If the user makes the text larger, the box that contains it expands proportionally. Likewise, if the user likes her text size very small, the containing box shrinks to fit. The result is that line lengths (in

terms of words or characters per line) stay the same regardless of the text size. This is an advantage over liquid layouts where line lengths can get too long, and fixed layouts where very large text may result in awkwardly few characters per line.

## **Advantages**

Provide a consistent layout experience while allowing flexibility in text size. Tighter control over line lengths than liquid and fixed layouts.

## **Disadvantages**

Images don’t lend themselves to rescaling along with the text and the rest of the layout. The width of the layout might exceed the width of the browser window at largest text sizes. This can be prevented with proper planning and/or the `max-width` property (unsupported in IE6 and earlier).

## **How to create elastic layouts**

The key to elastic layouts is the `em`, a unit of measurement that is based on size of the text. For example, for an element with 12-pixel text, an `em` is 12 pixels. It is always preferable to specify font size in `ems` because it allows the text to be resized with the zoom feature in all modern browsers (remember that IE6 and earlier do not zoom text sized in pixels). In elastic layouts, the dimensions of containing elements are specified in `ems` as well. That is how the widths can respond to the text size. For example, if the text size is set to 76% (equal to about 12 pixels on most browsers), and the page is set to 40 `em`, the resulting page width would be 480 pixels (40 `em` x 12px/`em`). If the user resizes the text up to 24 pixels, the page grows to 960 pixels. Note that this is getting close to the available canvas space in browsers on 1024-pixel wide monitors. If the page were set much more than 40 `ems` wide, there would be the risk of the right edge extending beyond the browser window at extremely large text sizes.

browser window depends on a style sheet, either one you provide or the browser’s built-in default rendering. Despite all the types of information you could add to a document, there are only a dozen of these elements in (X)HTML.

# Generic Elements (div and span)

There are endless types of information in the world, but as you’ve seen, not all that many semantic elements. Fortunately, (X)HTML provides two generic elements that can be customized to describe your content perfectly. The `div` (short for “division”) element is used to indicate a generic block-level element, while the `span` element is used to indicate a generic inline element. You give a generic element a name using either an `id` or `class` attribute.

The `div` and `span` elements have no inherent presentation qualities of their own, but you can use style sheets to format the content however you like. In fact, generic elements are a primary tool in standards-based web design because they enable authors to accurately describe content and offer plenty of “hooks” for adding style rules.

We’re going to spend a little time on `div` and `span` (as well as the `id` and `class` attributes, also called element identifiers) because they will be powerful tools once we start working with Cascading Style Sheets.

## **Divide it up with a div**

The `div` element is used to identify a block-level division of text. You can use a `div` like a container around a logical grouping of elements on the page. By marking related elements as a `div` and giving it a descriptive name, you give context to the elements in the grouping. That comes in handy for making

the structure of your document clear but also for adding style properties. Let's look at a few examples of div elements. In this example, a div element is used as a container to group an image and two paragraphs into a "listing".

```
<div class="listing">
  
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  <p>A combination of type history and examples of good and bad type.
</p>
</div>
```

By putting those elements in a div, I've made it clear that they are conceptually related. It also allows me to style the p elements within listings differently than other paragraphs on the page. Here is another common use of a div used to break a page into sections for context, structure, and layout purposes. In this example, a heading and several paragraphs are enclosed in a div and identified as the "news" section.

```
<div id="news">
  <h1>New This Week</h1>
  <p>We've been working on...</p>
  <p>And last but not least,... </p>
</div>
```

Now that I have an element known as "news," I could use a style sheet to position it as a column to the right or left of the page.

## Get inline with span

A span offers all the same benefits as the div element, except it is used for inline elements that do not introduce line breaks. Because spans are inline elements, they can only contain text and other inline elements (in other words, you cannot put block-level elements in a span). Let's get right to some examples.

In this example, each telephone number is marked up as a span and classified as "phone."

```
<ul>
  <li>Joan: <span class="phone">999.8282</span></li>
  <li>Lisa: <span class="phone">888.4889</span></li>
  <li>Steve: <span class="phone">888.1628</span></li>
  <li>Morris: <span class="phone">999.3220</span></li>
</ul>
```

You can see how the labeled spans add meaning to what otherwise might be a random string of digits. It makes the information recognizable not only to humans but to (theoretical) computer programs that know what to do with "phone" information. It also enables us to apply the same style to phone numbers throughout the site.

## Element identifiers

In the previous examples, we saw the element identifiers, id and class, used to give names to the generic div and span elements. Each identifier has a specific purpose, however, and it's important to know the difference.

### The id identifier

The id identifier is used to identify a unique element in the document. In other words, the value of id must be used only once in the document. This makes it useful for assigning a name to a particular element, as though it were a piece of data. See the sidebar, id and class Values, for information on providing names for the id attribute. This example uses the book's ISBN number to uniquely identify each listing. No two book listings may share the same id.

```
<div id="ISBN0321127307">
<p><cite>The Complete Manual of Typography</cite>, James Felici</p>
<p>A combination of type history and examples of good and bad type.</p>
</div>
<div id="ISBN0881792063">
<p><cite>The Elements of Typographic Style</cite>, Robert
  Bringhurst</p>
<p>This lovely, well-written book is concerned foremost with
  creating beautiful typography.</p></div>
```

## Output



*The Complete Manual of Typography, James Felici*  
A combination of type history and examples of good and bad type.

Web authors also use `id` when identifying the various sections of a page. With this method, there may not be more than one “header,” “main,” or other named `div` in the document.

```
<div id="header">
(masthead and navigation here)
</div>
<div id="main">
(main content elements here)
</div>
<div id="links">
(list of links here)
```

```
</div>
<div id="news">
(news sidebar item here)
</div>
<div id="footer">
(copyright information here)
</div>
```

## The class identifier

The class attribute is used for grouping similar elements; therefore, unlike the `id` attribute, multiple elements may share a class name. By making elements part of the same class, you can apply styles to all of the labeled elements at once with a single style rule. Let’s start by classifying some elements in the earlier book example. In this first example, I’ve added class attributes to certain paragraphs to classify them as “descriptions.”

```
<div id="ISBN0321127307" class="listing">

<p><cite>The Complete Manual of Typography</cite>, James Felici</p>
<p class="description">A combination of type history and examples of
  good and bad type.</p>
</div>
<div id="ISBN0881792063" class="listing">

<p><cite>The Elements of Typographic Style</cite>, Robert
  Bringhurst</p>
<p class="description">This lovely, well-written book is concerned
  foremost with creating beautiful typography.</p>
</div>
```

I've also classified each div as a "listing." Notice how the same element may have both a class and an id identifier. It is also possible for elements to belong to multiple classes. In this example, I've classified each div as a "book" to set them apart from "cd" or "dvd" listings elsewhere in the document.

```
<div id="ISBN0321127307" class="listing book">

<p><cite>The Complete Manual of Typography</cite>, James Felici</p>
<p class="description">A combination of type history and examples of
good and bad type.</p>
</div>
<div id="ISBN0881792063" class="listing book">

<p><cite>The Elements of Typographic Style</cite>, Robert
Bringhurst</p>
<p class="description">This lovely, well-written book is concerned
foremost with creating beautiful typography.</p>
</div>
```

This should have given you a good introduction to how div and span are used to provide meaning and organization to documents.

## CSS 3 (Cascading Style Sheets 3)

### What is CSS 3 ?

The biggest change that is currently planned with CSS level 3 is the introduction of modules. The advantage to modules is that it (supposedly) allows the specification to be completed and approved more quickly, because segments are completed and approved in chunks. This also allows browser and user-agent manufacturers to support sections of the specification but keep their code bloat to a minimum by only supporting those modules that make sense.

For example, a text reader wouldn't need to include modules that only define how an element is going to display visually. But even if it only included the aural modules, it would still be a standards-compliant CSS 3 tool.

#### Vendor prefixes

Back when CSS3 was newly released, vendor prefixes were used all the time, as they helped browsers interpret the code. Sometimes you still need to use them today, in case the browser you are testing in doesn't read the code. So below is a short list with all the vendor prefixes for major browsers:

- moz- : Firefox
- webkit- : Webkit browsers such as Safari and Chrome
- o- : Opera
- ms- : Internet Explorer

Note that, according to an official press release from a few weeks ago, Opera will soon build their new desktop and mobile browser on webkit too, instead of their current Presto rendering engine. This means that the -o- vendor prefix will disappear at some point in time, leaving us with only three major ones.



### New properties of CSS 3



- CSS3 Rounded Corners
- CSS3 Border Images
- CSS3 Backgrounds, Colors, Gradients
- CSS3 Shadows
- CSS3 Text
- CSS3 Fonts
- CSS3 2D Transforms
- CSS3 3D Transforms
- CSS3 Transitions
- CSS3 Animations
- CSS3 Multiple Columns
- CSS3 User Interface
- CSS3 Box Sizing
- CSS3 Flexbox
- CSS3 Media Queries
- CSS3 MQ Examples

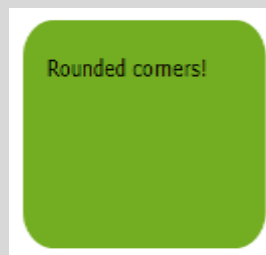
## CSS3 Rounded Corners

You can create rounded borders, add shadow to boxes, and use an image as a border –without using a design program, like Photoshop.

With CSS3, you can give any element "rounded corners", by using the **border-radius** property

### Example

```
#rcorners1 {  
  border-radius: 25px;  
  background: #73AD21;  
  padding: 20px;  
  width: 200px;  
  height: 150px;  
}
```



## CSS3 Border Images

border-image property you can use an image to create a border

The CSS3 **border-image** property allows you to specify an image to be used instead of the normal border around an element.

### Example

```
#rcorners1 {  
  border: 10px solid transparent;  
  padding: 15px;  
  border-image: url(border.png) 30% round;  
}
```

## CSS3 Backgrounds

CSS3 contains several new background properties, which allow greater control of the background element.

background-size : The background-size property specifies the size of the background image

- background-origin
- background-clip
- Multiple Background Images
- Background Gradients
- background-color

## **background-origin**

The background-origin property specifies the positioning area of the background images. The background image can be placed within the content-box, padding-box, or border-box area.

### **Example**

```
#rcorners1 {  
    background-image:url('smiley.gif');  
    background-repeat:no-repeat;  
    background-position:left;  
    background-origin:content-box;  
}
```

## **background-clip**

The background-clip property is used to determine whether the backgrounds extends into the border or not. The default is border-box, which means it DOES extend into it, but if you set it to padding-box, it doesn't. if you use content-box the background only extends to the content area.

```
#rcorners1 {  
    width:300px;  
    height:300px;  
    padding:50px;  
    background-color:yellow;  
    background-clip:content-box;  
    border:2px solid #92b901;}
```

## **Multiple Background Images**

CSS3 allows you to use several background images for an element.

### **Example**

```
#rcorners1 {  
    width: 580px; height: 200px;  
    background-image: url(img/sheep.png),  
    url(img/sheep.png),  
    url(img/betweengrassandsky.png);  
    background-repeat: no-repeat;  
    background-position: 20px 100px, 400px 50px, center bottom;  
    background-color: #EEE;  
    background-size: 70px, auto, cover;  
}
```



## Background Gradients

CSS3 gradients let you display smooth transitions between two or more specified colors. CSS3 defines two types of gradients:

- **Linear Gradients (goes down/up/left/right/diagonally)**
- **Radial Gradients (defined by their center)**

### Example

#### Linear Gradients

```
#rcorners1 {background: linear-gradient(red, blue);}
```

#### Radial Gradients

```
#rcorners1 {background: radial-gradient (red, blue);}
```

### background-COLOR

CSS supports color names, hexadecimal and RGB colors. In addition, CSS3 also introduces:

- RGBA colors [rgba(255, 0, 0, 0.2) ]
- HSL colors [hsl(120, 100%, 50%)]
- HSLA colors [hsla(120, 100%, 50%, 0.3)]
- Opacity [rgb(255,0,0);opacity:0.6 ]

## Shadows effects

There is two type of shadows effects of in CSS3

- Text Shadow
  - It is use to give shadows effect on text.

### Example Of Text Shadow

```
#shadows{  
  Color:#ff0;  
  text-shadow:5px 5px 5px #000;  
}
```



### Box Shadow

- It is use to give shadows effect on text.

### Example Of Text Box

```
#shadows{  
  Color:#000;  
  box-shadow:5px 5px 5px #000;
```

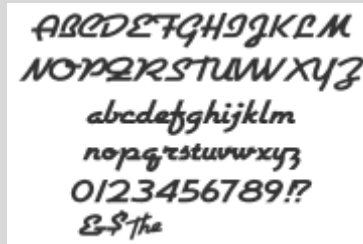
}

## Font Face

When you have found/bought the font you wish to use, just include the font file on your web server, and it will be automatically downloaded to the user when needed.

### Example Of Font Face

```
@font-face {  
  font-family: myFirstFont;  
  src: url(Name of font file.woff);  
  #shadows{  
    font-family: myFirstFont;}
```



ABCDEFGHIJKLMNOPQRSTUVWXYZ  
abcdefghijklmnopqrstuvwxyz  
0123456789!?  
£\$%&

## Transitions

**CSS transitions**, provide a way to control animation speed when changing CSS properties. Instead of having property changes take effect immediately, you can cause the changes in a property to take place over a period of time. For example, if you change the color of an element from white to black, usually the change is instantaneous. With CSS transitions enabled, changes occur at time intervals that follow an acceleration curve, all of which can be customized. The Web author can define which property has to be animated and in which way. This allows the creation of complex transitions. As it doesn't make sense to animate some properties, the list of animatable properties is limited to a finite set.

### Example

```
div {  
  width:100px;  
  height:100px;  
  background:red;  
  transition:all ease-in-out 2s;  
  -webkit-transition:width all ease-in-out 2s; /* Safari */  
}  
div:hover {  
  width:300px;  
}
```

We can defiant “S” for timing in second, Ex:- 0.3s

## Transforms

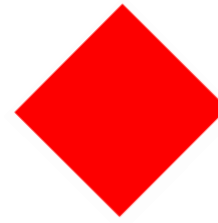
The CSS **transform** property lets you modify the coordinate space of the CSS visual formatting model. Using it, elements can be translated, rotated, scaled, and skewed. If the property has a value different than none, a stacking context will be created. In that case the object will act as a containing block for position: fixed elements that it contains.

### Example

```
div {  
  width:200px;  
  height:200px;  
  transform:rotate(180deg);  
  -webkit-transform:rotate(180deg);  
}
```

## Types of transform

- `translate(x,y,z)`  
it is defined in a pixels **Ex:-** 100deg
  - `translate`
  - `translate-X`
  - `translate-Y`
  - `translate-Z`
- `scale(x,y,z)`  
it is for scale your design in x and y axis
  - `scale-x`
  - `scale-y`
  - `scale-z`
- `rotate(x,y,z)`  
it is defined in a degree **Ex:-** 90deg
  - `rotate-x`
  - `rotate-y`
  - `rotate-z`
- `skew(x-angle,y-angle)`  
it is defined in a degree in x and y axis **Ex:-** 90deg
  - `rotate-x`
  - `rotate-y`
- `perspective(n)`



## Animation

### @keyframesRule

The @keyframesrule is where the animation is created. Specify a CSS style inside the @keyframesrule and the animation will gradually change from the current style to the new style.

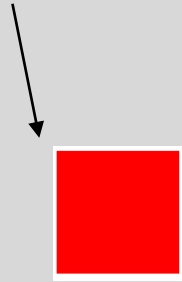
### Property

- **@keyframes:-** Specifies the animation
- **Animation-name :-** Specifies the name of the @keyframesanimation
- **Animation-duration:-** Specifies how many seconds or milliseconds an animation takes to complete one cycle. Default 0
- **Animation-timing-function:-** Describes how the animation will progress over one cycle of its duration. Default "ease"
- **Animation-delay:-** Specifies when the animation will start. Default 0
- **Animation-iteration-count:-** Specifies the number of times an animation is played. Default
- **Animation-direction:-** Specifies whether or not the animation should play in reverse on alternate cycles. Default "normal"
- **Animation-play-state:-** Specifies whether the animation is running or paused. Default "running"
- **Browser Support**

### Multiple changes

To add a transitional effect for more than one style, add more properties, separated by commas

```
Div {  
  width:100px; height:100px;  
  background:red; position:relative;  
  animation:mymove 5s infinite;  
  -webkit-animation:mymove 5s infinite;  
@keyframes mymove {  
  from {top:0px;} to {top:200px;}  
@-webkit-keyframes mymove {  
  from {top:0px;} to {top:200px;}
```



## Multiple changes

The column-count property specifies the number of columns an element should be divided

### Property

- column-count:- Specifies the number of columns an element should be divided
- column-fill:- Specifies how to fill columns
- column-gap:- Specifies the gap between the columns
- column-rule:- A shorthand property for setting all the column-rule
- column-rule-color:- Specifies the color the rule between columns
- column-rule-style:- Specifies the style of the rule between columns
- column-rule-width:- Specifies the width of the rule between columns
- column-width:- Specifies the width of the columns
- Columns:- A shorthand property for setting column-width and column-count

## Example

```
.newspaper  
{  
-moz-column-count:3; /* Firefox */  
-webkit-column-count:3; /* Safari and Chrome */  
column-count:3;  
-moz-column-gap:40px; /* Firefox */  
-webkit-column-gap:40px; /* Safari and Chrome */  
column-gap:40px;}
```

The word-wrap property was invented by Microsoft and added to CSS3. It allows long words to be able to be broken and wrap onto the next line. It takes in two values; normal or break-word. In the first paragraph below, normal is used. This is the same as if the property wasn't used, i.e. the long word breaks out of the box as there isn't enough width for it to be fully

The word-wrap property was invented by Microsoft and added to CSS3. It allows long words to be able to be broken and wrap onto the next line. It takes in two values; normal or break-word. In the first paragraph below, normal is used. This is the same as if the property wasn't used, i.e. the long word breaks out of the box as there isn't enough width for it to be fully

The word-wrap property was invented by Microsoft and added to CSS3. It allows long words to be able to be broken and wrap onto the next line. It takes in two values; normal or break-word. In the first paragraph below, normal is used. This is the same as if the property wasn't used, i.e. the long word breaks out of the box as there isn't enough width for it to be fully

## Media Queries

Media Queries is a W3C Candidate Recommendation a widely reviewed document which is ready for implementation by browser vendors.

It's an extension of media dependent stylesheets tailored for different media types (i.e. screen and print) found in CSS2.

In its essence a media query consists of a media type and an expression to check for certain conditions of a particular media feature. The most commonly used media feature is width.

By restricting CSS rules to a certain width of the device displaying a web page, one can tailor the page's representation to devices (i.e. smartphones, tablets, netbooks, and desktops) with varying screen resolution.

## What is Responsive Web Design?

The term Responsive Web Design was coined by Ethan Marcotte and is the practice of using fluid grids, flexible images, and media queries to progressively enhance a web page for different viewing contexts.

What screen resolutions do you use while taking screenshots?

- Smartphone - 320px
- Tablet - 768px
- Netbook - 1024px



## Example

```
.#d1
{ background-color: #330066; width:400px; height:400px; }
@media only screen and (min-width : 768px) {
    #d1 {
        background-color: #0f0; min-width:600px; height:400px;
    }
}
@media only screen and (min-width : 1024px) {
    #d1 {
        background-color: #0f0; min-width:800px; height:600px;
    }
}
@media only screen and (min-width : 1600px) {
    #d1 {
        background-color: #0f0; min-width:1000px; height:600px;
    }
}
```



## HTML 5

### What is HTML 5 ?

**HTML5** is a markup language used for structuring and presenting content on the World Wide Web. It was finalized, and published, on 28 October 2014 by the World Wide Web Consortium (W3C). This is the fifth revision of the HTML standard since the inception of the World Wide Web.

### Introduction

- HTML5 is the latest and most enhanced version of HTML. Technically, HTML is not a programming language, but rather a markup language.
- HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1. HTML5 is a standard for structuring and presenting content on the World Wide Web.
- HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).
- The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

### Tools

- Text Pad or Text Wrangler or Notepad
  - Dreamweaver okay but it does cost.
- Compatible browser
  - Firefox: works for all examples, including masked video and relocation
  - Chrome and Safari work for most examples
  - IE9 being tested now

### Syntax

The HTML 5 language has a "custom" HTML syntax that is compatible with HTML 4 and XHTML1 documents published on the Web, but is not compatible with the more esoteric SGML features of HTML 4

- HTML 5 does not have the same syntax rules as XHTML where we needed lower case tag names, quoting our attributes, an attribute had to have a value and to close all empty elements.
- But HTML5 is coming with lots of flexibility and would support the followings:
- Uppercase tag names.
- Quotes are optional for attributes.
- Attribute values are optional.
- Closing empty elements are optional.

## New Features

HTML5 introduces a number of new elements and attributes that helps in building a modern websites. Following are great features introduced in HTML5.

**New Semantic Elements:** These are like <header>, <footer>, and <section>.

**Forms 2.0:** Improvements to HTML web forms where new attributes have been Introduced for <input> tag.

**Persistent Local Storage:** To achieve without resorting to third-party plugins.

**WebSocket** :A a next-generation bidirectional communication technology for web applications.

**Server-Sent Events:** HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).

**Canvas:** This supports a two-dimensional drawing surface that you can program with JavaScript.

**Audio & Video:** You can embed audio or video on your web pages without resorting to third-party plugins.

**Geolocation:** Now visitors can choose to share their physical location with your web application.

**Microdata:** This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.

**Drag and drop:** Drag and drop the items from one location to another location on a the same webpage.

## New tags of HTML 5

- |               |           |             |
|---------------|-----------|-------------|
| • Article,    | • figure, | • progress, |
| • aside,      | • footer, | • rp, ruby, |
| • audio,      | • header, | • section,  |
| • canvas,     | • hgroup, | • source,   |
| • command,    | • keygen, | • summary,  |
| • datalist,   | • mark,   | • time,     |
| • details,    | • meter,  | • video     |
| • embed,      | • nav,    |             |
| • figcaption, | • output, |             |

**Section** :- The section element represents a generic section of a document applicationA section in this context is a thematic grouping content typically with a heading..

**Article** :- This article element represents a self- contained composition in a document, page, application, or site and that is intended to be independently distributable or reusable, e.g. In syndication

**Aside** :- The aside element represents a section of a page that consists of content that is tangentially related to the content around the aside element, and which could be considered separate from that content.

**Header** :- The header element represents a group of introductory or navigational aids. A header element is intended to usually contain the section's heading (an h1-h6 element or an hgroup element),but this is not required.

**Hgroup** :- The hgroup element represents the heading of a section. The element is used to group a set of h1-h6 elements when the heading has multiple levels, such as subheading, alternative titles, or taglines.

**Footer** :-The footer element represents a footer for its nearest ancestor sectioning content or sectioning root element.A footer typically contains information about its section such as s who wrote it. Links to related documents. Copyright data and the like.

**Nav** : The nav element represents a section of a page that links to other pages or to parts within the page a section with navigation links.Not all links of a document must be in a <nav> element. The <nav> element is intended only for major block of navigation links.Browsers, such as screen readers for disabled users, can use this element to determine whether to omit the initial rendering of this content.

**Figure** :- The figure element represents some flow content, optionally with a caption that is self contained and is typically referenced as a single unit from the main flow of the document.

**Video** :- Is a media element whose media data is ostensibly video data, possibly with associated audio data.

**Audio** :- The audio element is a media element whose media data is ostensibly audio data.

**Embed** :- The embed element represents an integration point for an external (typically non HTML) application or interactive content

**Canvas** :- Providing scripts with a resolution dependent bitmap canvas, which can be used for rendering graphs, game graphics, or other visual images on the fly.

**Meter** :- The meter element represents a scalar measurement within a known range or a fractional value for example disk usage the relevance of a query result, or the fraction of a voting population to have selected a particular candidate.

**Progress** :- The progress element represents the completion progress of a task. The progress is either indeterminate indicating that progress is being made but that it is not clear how much more work remains to be done before the task is complete(eg. Because the task is waiting for a remote host to respond), or the progress is a number in the range zero to a maximum giving the fraction of work that has so far been completed.

**Time** :- The time element represents either a time on a 24 hour clock or a precise date in the proleptic gregorian calendar, optionally with a time and a time zone offset. this element is intended as a way to encode modern dates and times in a machine readable way so that, for example, user agents can offer to add birthday reminders or scheduled events to the user's calendar

**Menu** :- The menu element represents a list of commands

**Command** :- The command element represents a command that the user can invoke

## HTML5 form tag

No doubt you interact with at least one form on the Web every day. Whether you're searching for content or logging in to your e-mail account or Facebook page, using online forms is one of the most common tasks performed on the Web. As designers and developers, creating forms has a certain monotony about it, particularly writing validation scripts for them. HTML5 introduces a number of new attributes, input types, and other elements for your markup

toolkit. In this article we'll be focusing on the new attributes with a future article looking at the new input types.

As we'll see, these new features will go a long way toward making your life easier while delivering a delightful user experience. The best thing about all this? You can start using them now. We'll start with a (very) brief history of HTML5 forms though.

## HTML5 Input type form tag

**Color :-** The color type is used for input fields that should contain a color

```
<input type="color">
```

**Date :-** The date type allows the user to select a date

```
<input type="datetime-local">
```

**email :-** The datetime type allows the user to select a date and time

```
<input type="email">
```

**Month :-** The month type allows the user to select a month and year

```
<input type="month">
```

**Number :-** The number type is used for input fields that should contain a numeric value. You can also set restrictions on what numbers are accepted:

```
<input type="number" name="quantity" min="1" max="5">
```

**Range :-** The range type is used for input fields that should contain a value from a range of numbers. You can also set restrictions on what numbers are accepted

```
<input type="range" name="points" min="1" max="10">
```

**Search :-** The search type is used for search fields

```
<input type="search" name="s">
```

**Time :-** The time type allows the user to select a time

```
<input type="time">
```

**Url :-** The url type is used for input fields that should contain a URL address. The value of the url field is automatically validated when the form is submitted

```
<input type="url">
```

**week :-** The week type allows the user to select a week and year

```
<input type="week">
```

**Autocomplete :-** Specifies whether an <input> element should have autocomplete enabled

```
<form action="demo_form.asp" autocomplete="on">
```

```
  First name:<input type="text" name="fname"><br>
```

```
  Last name: <input type="text" name="lname"><br>
```

```
  E-mail: <input type="email" name="email" autocomplete="off"><br>
```

```
  <input type="submit">
```

```
</form>
```

**Accept :-** Specifies the types of files that the server accepts (only for type="file"). We can attach all types of doc, image and zip file

```
<input type="file" accept="image/*">
```

**Autofocus :-** Specifies that an <input> element should automatically get focus when the page loads

```
<input type="text" autofocus="autofocus"/>
```

**List :-** Refers to a <datalist> element that contains pre-defined options for an <input> element

```
<form action="demo_form.asp" method="get">
```

```
  <input list="browsers" name="browser">
```

```
  <datalist id="browsers">
```

```
    <option value="Internet Explorer"><option value="Firefox">
```

```
    <option value="Chrome"><option value="Opera">
```

```
    <option value="Safari">
```

```
  </datalist>
```

```
<input type="submit">
</form>
```

**Pattern** :- Specifies a regular expression that an <input> element's value is checked against

```
<input type="text" name="country_code" pattern="[A-Za-z]{3}" title="Three letter country code">
```

```
<input type="submit">
```

**Placeholder** :- Specifies a short hint that describes the expected value of an <input> element

```
<input type="text" name="fname" placeholder="First name">
```

**Keygen** :- The <keygen> tag specifies a key-pair generator field used for forms.

When the form is submitted, the private key is stored locally, and the public key is sent to the server.

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">
  0 <input type="range" id="a" value="50">100
  + <input type="number" id="b" value="50">=
  <output name="x" for="a b">
</output>
</form>
```

## Javascript

### What is Javascript?

Javascript is a scripting language (scripting language is light weight programming language). It was design to add interactivity to HTML Pages. Javascript is an interpreted language means that scripts execute without preliminary compilation. It's code directly embedded to HTML Page. It works for all major browsers.

Javascript is a client side, interpreted, Object Oriented, high level scripting language. Can not runs stand alone. Created By Netscape in 1995, evolved from Netscape's Live Script.

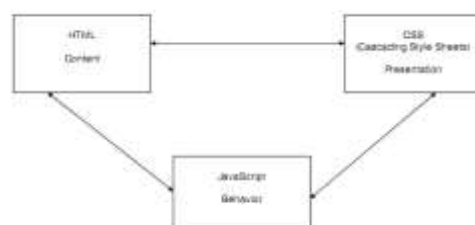
### What Can Javascript Do?

Javascript use for following :

- JavaScript gives HTML designers a programming tool
- JavaScript can put dynamic text into an HTML page
- JavaScript can react to events
- JavaScript can read and write HTML elements
- JavaScript can be used to validate input data
- JavaScript can be used to detect the visitor's browser
- JavaScript Improve Appearance
  - Especially graphics
  - Visual Feed back
- JavaScript Perform Calculation

## How it Works?

Javascript work with HTML and css (Cascading Style Sheets).



Javascript directly embedded to html and it work with html element, also work with css.

## Add Javascript to HTML Page

With the help of **<script>** tag (also use the **type** attribute to define script language), we can create javascript code in html page. There is tow part where we can create JavaScript code in html page.

1. **head part**
2. **body part**

### 1. head part

In head part we can add javascript as following:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Test HTML</title>
<script type="text/javascript">
    <!--write Javascript code here-->
</script>
</head>

<body>
    <!--write html tag here for design-->
</body>

</html>
```

### 2. body part

In body part we can add javascript as following:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Test HTML</title>
</head>
<body>
<script type="text/javascript">
    <!--write Javascript code here-->
</script>
    <!--write html tag here for design-->
</body>
</html>
```

## Referencing external Javascript File

We can also create a seperate file for javascript code and then access that file using **src** attribute in **script** tag. Using this we can add javascript that create locally in html/css template folder and also add that is remotely from any web site.

For local Javascript.js file ,

```
<script type="text/javascript" src="js/JavaScript.js"></script>
```

For Remotely access Javascript.js file,



```
<script type="text/javascript" src="http://somesite/JavaScript.js"></script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Untitled Document</title>

<script type="text/javascript">
    document.write("this is my first javascript example");
</script></head>
<body></body>
</html>
```

## Example 1

there is no any text written in body tag but when this html page is open in browser than we can show "this is my first javascript example" text because this code of javascript is runs as page is loaded automatically

## JavaScript Variable

To hold some values or object we have to create variable in javascript. It done by **var** statement.

```
var strname="John"
```

you can also create any variable without using **var** statement.

```
strname="john"
```

## JavaScript Function

Javascript function is use when we work on html events like onload, onunload of page and click event of submit button .

```
function function_name(parameter list){
    <!-- Code -->
}
```

using **function** statement we can create function in javascript.

## JavaScript Popup Box

There is three type of popup box in javascript.

1Alert box

2Confirm Box

3Prompt Box

### 1) Alert Box

An alert box is often used if you want to make sure information comes through to the user. When an alert box pops up, the user will have to click "OK" to proceed.

## Example 2

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Test Document</title>

<script type="text/javascript">
    alert("Hello! I am an alert box! ");
</script>
</head>
<body>
</body>
</html>
```

Output:-



## Output

This code show alert box like figure, in browser when page open. We can also run javascript function on Events like Button Onclick event.

### Example 3

#### 2) Confirm Box

A confirm box is often used if you want the user to verify or accept something.

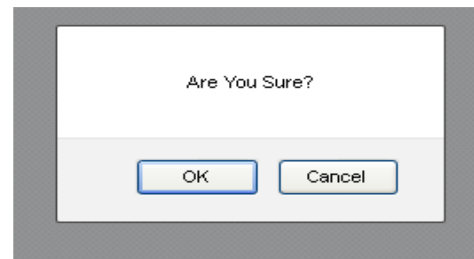
When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

#### Example 4:

```
<html >
<head>
<title>Test Document</title>
<script type="text/javascript">
    function confirmit(){
        if(confirm("Are You Sure?")){
            alert("You Click On Ok!");
        }else {
            alert("You Click On Cancel! ");
        }
    }
</script></head><body>
<input type="button" value="Click Me!"
onclick="confirmit()"/>
</body></html>
```

Output:-



#### 3) Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

```
<html><head>
<script type="text/javascript">
function PromtpBox(){
    var name=prompt("Enter Your Name","here");
    alert("Your Name is:\t"+name);
}
</script></head>
<body>
<input type="button" value="Click Me!"
onclick="PromtpBox()"/>
</body></html>
```

Output:-



After type you name when you click on Ok then name is store in variable name and show in alert box, but if you click on cancel than it show null value in alert box.

## Get Html Element in Javascript

- We can get any html element using id in Javascript variable and then use that element value in javascript . To get any element by it's id we have to use **getElementById()** function.

On Button click you can show following output.

```
<html><head>
<script type="text/javascript">
    function GetTextBox(){
        var text=document.getElementById("txt");
        alert("Value in TextBox is:"+text.value);
    }
</script></head>
<body><input type="text" id="txt" />
<input type="button" value="Get TextBox Value!" onclick="GetTextBox()"/>
</body></html>
```

Output:-



## Javascript with CSS Style

- We can also use javascript with css styles like text color, background color and much more.
- For this you have to use getElementById() to get that element which stylesheet we have to change.
- This is done by changing styles on ID elements directly and showing how to change the class of an element .

### Example7

```
<html>
<head>
<script type="text/javascript">
    function ChangeColor1(){
        document.getElementById("id1").style.backgroundColor='#000';
        document.getElementById("id1").style.color='#FFF';}    function ChangeColor2(){

        document.getElementById("id1").style.backgroundColor='#FFF';
        document.getElementById("id1").style.color='#000';
    }
</script></head>
<body ><div id="id1" onmouseover="ChangeColor1()"
onmouseout="ChangeColor2()">
<center>this is my text of this div.</center></div>
</body></html>
```

### Output

When you over mouse on div than ChangeColor1 called and using document.getElementById("id1").**style.backgroundColor** we can change it's background color and using document.getElementById("id1").**style.color** we can change it's text color property.

### Javascript with CSS Class

- We can also directly working with the class property of any element in body part using javascript.
- For this first we have to class define in style sheet.
- In this we can change element class using javascript function look in following example.

## Javascript Validation

- Validation is the process of checking that a form has been filled in correctly before it is processed.
- For example, User Name filled in text box, email id format is correct, mobile number filled with digits only etc.
- To validate this on client side javascript is used.
- In Javascript we can check all possibility of wrong input at client side.
- For Example,

## Example9

### Val.js

```
<script type="text/javascript">
function ValidateForm(){
    var fnm=document.getElementById("txtfname");
    var lnm=document.getElementById("txtlname");
    var address=document.getElementById("address");
    var con=document.getElementById("country");
    var mob=document.getElementById("txtmob");
    var eid=document.getElementById("txteid");
    var p1=document.getElementById("txtpass1");
    var p2=document.getElementById("txtpass2");
    var errors="";
    if(fnm.value.length==0)
    {
        errors=errors+"\n Enter Your First Name";
    }
    if(lnm.value.length==0)
    {
        errors=errors+"\n Enter Your Last Name";
    }
    if(address.value.length==0)
    {
        errors=errors+"\n Enter Your Addres";
    }
    if(mob.value.length==0)
    {
        errors=errors+"\n Enter Your Mobile Number";
    }
    if(eid.value.length==0)
    {
        errors=errors+"\n Enter Your Email Id";
    }
    if(p1.value.length==0)
    {
        errors=errors+"\n Enter Your Enter Password";
    }
    if(errors!="")
    {
        alert(errors);
        return false;
    }
    else
    {
        var letters = /^[A-Za-z]+$/.test("");
        var numbers = /^[0-9]+$/.test("");
        var mailformat = /^[w+([.-]?\w+)*@\w+([.-]?\w+)*(\.w{2,3})+$/;
    }
}
```

## Index.html

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><script src="val.js"></script>
</head>
<body >
<form>
<table>
<caption>
    Registration Form
</caption>
<tr>
<td>First Name</td>
<td><input id="txtfname" type="text" /></td>
</tr>
<tr>
<td>Last Name</td>
<td><input id="txtlname" type="text" /></td>
</tr>
<tr>
<td>Address</td>
<td><textarea rows="5" class="40" id="address"></textarea></td>
</tr>
<tr>
<td>Mobile Number</td>
<td><input id="txtmob" type="text" /></td>
</tr>
<tr>
<td>Email Id</td>
<td><input id="txteid" type="text" /></td>
</tr>
<tr>
<td>Password</td>
<td><input id="txtpass1" type="password" /></td>
</tr>
<tr>
<td>Re-Enter Password</td>
<td><input id="txtpass2" type="password" /></td>
</tr>
<tr>
<td>
<input type="button" id="Save" value="Save Data" onclick="return ValidateForm()" />
</td>
<td>
<input type="reset" id="reset" value="Reset" />
</td>
</tr></table>
</form></body></html>
```

## Output:

The screenshot shows a 'Registration Form' with the following fields and values: First Name (123), Last Name (123), Address (aaaa), Mobile Number (abc), Email Id (abc), Password (masked with dots), and Re-Enter Password (masked with dots). There are 'Save Data' and 'Reset' buttons at the bottom left. A white dialog box on the right lists validation errors: 'First Name Only a-z or A-Z allow', 'Last Name Only a-z or A-Z allow', 'Mobile number only 0-9 allow', 'Mobile number only 10 digit allow', 'Wrong Email Id', and 'Password Not Match'. An 'OK' button is at the bottom right of the dialog box.

## Example : 10

### Handling the KeyPress Even...

```
<html>
<head>
    <script type="text/javascript">
        function checkonkey()
        {
            if(event.keyCode>=48 && event.keyCode<=57)
            {
                return true;
            }
            else{
                return false;
            }
        }
    </script>
</head><body>
<input type="text" id="txtPincode" name="txtPincode" onkeypress="return checkonkey();" />
</body></html>
```

**Output:** It will allow only digit. You cannot fill text in this textbox.



## Jquery

### What is jQuery?

Is a new kind of JavaScript Library?

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript.

jQuery is a multi-browser JavaScript library designed to simplify the client-side scripting of HTML. It was released in January 2006 at BarCamp NYC by John Resig. It is currently developed by a team of developers led by Dave Methvin. Used by over 55% of the 10,000 most visited websites, jQuery is the most popular JavaScript library in use today.

jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and web applications.

**jQuery includes the following features:**

- Events
- Effects and animations
- Ajax
- Extensibility through plug-ins
- Utilities - such as user agent information, feature detection
- Multi-browser (not to be confused with cross-browser) support.

## About Browser Compatibility

jQuery actively supports these browsers:

Firefox Current - 1 version  
Internet Explorer 6+  
Safari Current - 1 version  
Opera Current - 1 version  
Chrome Current - 1 version

Any problem with them should be considered and reported as a bug in jQuery.

There are known problems with:

Mozilla Firefox 1.0.x  
Internet Explorer 1.0-5.x  
Safari 1.0-2.0.1  
Opera 1.0-9.x  
Konqueror

jQuery generally works with Konqueror and Firefox 1.0.x, but there may be some unexpected bugs since we do not test them as regularly.

## Including the library

The jQuery library is a single JavaScript file, containing all of its common DOM, event,

# TOPS Technologies

effects, and Ajax functions. It can be included within a web page by linking to a local copy, or to one of the many copies available from public servers.

You can find this library online and include it using its link or download it.

Show in following, **Code:**

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>jQuery Document</title>
<script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
<input type="button" value="Click Me!" id="btn1"/>
<script type="text/javascript">
    $("#btn1").click(
        function(){alert("I am Clicked!");}
    );
</script>
</body>
</html>
```

Output:

When Button is clicked then alert box show in window.

1 Here we are not going to add javascript function in button onclick event but using jquery code we can add that on button.

2 \$ sign is use to declare code of jquery.

3 In Head part, JQuery latest library is included online we can also download and include that locally from folder of template. We can also use following code to add any function on button event in head part.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>jQuery Document</title>
<script src="http://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
    $(document).ready (
        function()
        {
            $("#btn1").click(
                function()
                {
                    alert("I am Clicked!");
                }
            );
        }
    );
</script></head><body>
<input type="button" value="Click Me!" id="btn1"/>
</body></html>
```

This code also work as above code. Show alert box when button is clicked.

## Jquery Selector

Jquery selector is use for select element, class and etc for performing some process on it. As show in previous example, #btn1 is id of button that is selected for add alert box on it's click event.

The basic selector are:

## All Selector ("\*")

^ jQuery('\*')

**Description:** Selects all elements.

Caution: The all, or universal, selector is extremely slow, except when used by itself.

```
<html>
<head>
<title>jquery Test</title>
<style type="text/css">
h3 { margin: 0;}
div, span, p
{
width: 80px; height: 40px; float:left; padding: 10px; margin: 10px;
background-color: #EEEEEE;
}
</style>
<script src="http://code.jquery.com/jquery-latest.js"></script>
</head><body>
<div>This is DIV</div>
<span>This is SPAN</span>
<p><button>Go!</button></p>
<script type="text/javascript">
    var elementCount = $("*").css("border","2px outset black").length;
    $("body").prepend
    (
        "<h3>" + elementCount + " elements found</h3>"
    );
</script></body></html>
```

Output:-



## animated Selector

^jQuery(':animated')

**Description:** Select all elements that are in the progress of an animation at the time the selector is run.

```
<html>
<head>
<title>jquery Test</title>
<style type="text/css">
div {
    background-color:#00F;
    border:2px outset #00F;
    width:40px;
    height:40px;
    margin:0 5px;
    float:left;
}
div.colored {
    background-color:#F00;
}
</style>
<script src="http://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
    $(document).ready(
        function()
        {
            $("#run").click(function(){
                $("div:animated").toggleClass("colored");
            });
            function animateIt() {
                $("#mover").slideToggle("slow", animateIt);
            }
            animateIt();
        });
</script>
</head>
<body>
<button id="run">Change Color</button>
<div></div>
<div id="mover"></div>
<div></div>
</body>
</html>
```

## Class Selector (".class")

^jQuery('.class')

1class

A class to search for. An element can have multiple classes; only one of them must match.

**Description:** Selects all elements with the given class.

For class selectors, jQuery uses JavaScript's native `getElementsByClassName()` function if the browser supports it.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>jQuery Document</title>
<script src="http://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
    $(document).ready(function(){
        $(".class1").css("background-color", "#00F");
    });
</script>
</head>
<body class="class1">
    <h1>this is body part!</h1>
</body>
</html>
```

using this code we can add css for particular class in HTML pages.

## Element Selector (“element”)

^ `jQuery('element')`

1 element

An element to search for. Refers to the `tagName` of DOM nodes.

**Description:** Selects all elements with the given tag name.

JavaScript's `getElementsByTagName()` function is called to return the appropriate elements when this expression is used.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>jQuery Document</title>
<script src="http://code.jquery.com/jquery-latest.js"></script>

<script type="text/javascript">
    $(document).ready(function(){
        $(".class1").css("background-color", "#00F");
        $("div").css("background-color", "#FAF");
    });
</script>

</head>
```

```
<body class="class1">
<h1>this is body part!</h1>
<div><h3>This is div 1</h3></div>
<div><h3>This is div 2</h3></div>
</body>
</html>
```

all div background color is changed to #FAF in html page using this element selector. It select html Element directly using tag name.

## Empty Selector

jQuery(':empty')

**Description:** Select all elements that have no children (including text nodes).

This is the inverse of :parent. One important thing to note with :empty (and :parent) is that child elements include text nodes.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><script src="http://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
    $(document).ready(function(){
        $("div:empty").text("Div is empty!").css('background','rgb(255,220,200)');
    });
</script></head>
<body>
<h1>this is body part!</h1>
<div></div>
<div><h3>This is div 1</h3></div>
<div><h3>This is div 2</h3></div>
<div></div>
</body>
</html>
```

### Attributes

.addClass()

## addClass( className )

**Description:** Adds the specified class(es) to each of the set of matched elements.

**className** One or more class names to be added to the class attribute of each matched element.

It's important to note that this method does not replace a class. It simply adds the class, appending it to any which may already be assigned to the elements.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style type="text/css">
div {
    background-color:#00F;
    border:2px outset #00F;
    width:100px;
```



```
        height:40px;
        margin:0 5px;
        float:left;
    }
    .class1{
        width:200px;
        height:200px;
        background-color:#999;
        padding:1px;
        border-style:2px outset #03C;
    }
</style>
<script src="http://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
    $(document).ready(
        function(){
            $("#div1").click(
                function(){
                    $("#div1").addClass("class1");
                });
        });
</script>
</head>
<body>
<div id="div1">This is test Class</div>
</body>
</html>
```

## removeClass()

.removeClass( [className] )

**Description:** Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

**className** One or more space-separated classes to be removed from the class attribute of each matched element.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style type="text/css">
div {
    background-color:#00F;
    border:2px outset #00F;
    width:100px; height:40px;
    margin:0 5px; float:left;
}
.class1
{
```

```
width:200px; height:200px;
background-color:#999;
padding:1px; border-style:2px outset #03C;
}
</style>
<script src="http://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
    $(document).ready(
        function(){
            $("#div1").click(
                function(){
                    $("#div1").removeClass("class1");

                }
            );
        }
    );
</script></head>
<body>
<div id="div1" class="class1">This is test Class</div></body></html>
```

## toggleClass()

.toggleClass( className )

**Description:** Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

**ClassName** One or more class names (separated by spaces) to be toggled for each element in the matched set.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style type="text/css">
div {
    background-color:#00F;
    border:2px outset #00F;
    width:100px; height:40px;
    margin:0 5px; float:left;}
.class1{
    width:200px; height:200px;
    background-color:#999;
    padding:1px; border-style:2px outset #03C;}
</style><script src="http://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
    $(document).ready(
        function(){$("#div1").click(function(){
            $("#div1").toggleClass("class1");

        }
    );
    });
</script></head>
<body><div id="div1">This is test Class</div></body></html>
```

In this Example when you click on div then it's class is set to class1 and when you click again is reset to previous one. Here the class1 is toggle.

## Effects

The jQuery library provides several techniques for adding animation to a web page. These include simple, standard animations that are frequently used, and the ability to craft sophisticated custom effects. In this chapter, we'll closely examine each of the effect methods, revealing all of the mechanisms jQuery has for providing visual feedback to the user.

Basics Effects

▲ **.hide**

▲ **.show**

▲ **.toggle**

### .hide( )

**Description:** Hide the matched elements.

**.hide()**

simple hide function.

**.hide( duration [, callback] )**

**duration**A string or number determining how long the animation will run.

**callback**A function to call once the animation is complete.

**.hide( [duration] [, easing] [, callback] )**

**duration**A string or number determining how long the animation will run.

**easing**A string indicating which easing function to use for the transition.

**callback**A function to call once the animation is complete.

```
<html>
<head>
<script src="http://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
$(document).ready(
function(){
$("#p").hide();
});
</script>
</head>
<body>
<p>Hello</p>
<p>Here is another paragraph</p>
</body>
</html>
```

**p tag is not show any text in browser because of hide() api in jquery.**

```
<html>
<head>
<style>
p { background:#dad; font-weight:bold; }
</style>
<script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
```

```
<button>Hide all</button>
```

```
<p>Hiya</p>
```

```
<p>Such interesting text, eh?</p>
```

```
<script>
```

```
    $("button").click(function () {  
        $("p").hide("slow");  
    });
```

```
</script>
```

```
</body>
```

```
</html>
```

on button click all p tag hide but slowly because this hide() api use with duration.

show()

show( )

**Description:** Display the matched elements.

.show()

.show( duration [, callback] )

**duration**A string or number determining how long the animation will run.

**callback**A function to call once the animation is complete.

**.show( [duration] [, easing] [, callback] )**

**duration**A string or number determining how long the animation will run.

**easing**A string indicating which easing function to use for the transition.

**callback**A function to call once the animation is complete.

```
<html><head><style>    p { background:yellow; }
```

```
</style>
```

```
<script src="http://code.jquery.com/jquery-latest.js"></script>
```

```
</head>
```

```
<body>
```

```
<button>Show it</button>
```

```
<p style="display: none">Hello 2</p>
```

```
<script>
```

```
    $("button").click(function () {  
        $("p").show("slow");  
    });
```

```
</script>
```

```
</body>
```

```
</html>
```

toggle()

.toggle( [duration] [, callback] ) Returns: [jQuery](#)

**Description:** Display or hide the matched elements.

## ^ .toggle( [duration] [, callback] )

**duration**A string or number determining how long the animation will run.

**callback**A function to call once the animation is complete.

## ^ .toggle( [duration] [, easing] [, callback] )

**duration**A string or number determining how long the animation will run.

**easing**A string indicating which easing function to use for the transition.

**callback**A function to call once the animation is complete.

## ^ .toggle( showOrHide )

**showOrHide**A Boolean indicating whether to show or hide the elements.

```
<html>
<head>
<script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
<button>Toggle</button>
<p>Hello</p>
<p style="display: none">Good Bye</p>
<script>
$("button").click(function () {
  $("#p").toggle();
});
</script>
</body>
</html>
```

## Custom Effects

.animate( properties [, duration] [, easing] [, complete] )

**Description:** Perform a custom animation of a set of CSS properties.

.animate( properties [, duration] [, easing] [, complete] )

**properties**A map of CSS properties that the animation will move toward.

**duration**A string or number determining how long the animation will run.

**easing**A string indicating which easing function to use for the transition.

**complete**A function to call once the animation is complete.

.animate( properties, options )

**properties**A map of CSS properties that the animation will move toward.

**options**A map of additional options to pass to the method. Supported keys:

- duration: A string or number determining how long the animation will run.
- easing: A string indicating which easing function to use for the transition.
- complete: A function to call once the animation is complete.

## Example

```
<html>
<head>
  <style>
    div {
      background-color:#bca;
      width:100px;
      border:1px solid green;
    }
  </style>
  <scriptsrc="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
  <buttonid="go">&raquo; Run</button>

  <divid="block">Hello!</div>
  <script>
    $("#go").click(function(){
      $("#block").animate({ width:"70%", opacity:0.4, marginLeft:"0.6in",
        fontSize:"3em", borderWidth:"10px"
      },1500);
    });
  </script>

</body>
</html>
```

delay()

delay( duration [, queueName] )

**Description:** Set a timer to delay execution of subsequent items in the queue.

▲ **.delay( duration [, queueName] )**

**duration**An integer indicating the number of milliseconds to delay execution of the next item in the queue.

**queueName**A string containing the name of the queue. Defaults to fx, the standard effects queue.



Added to jQuery in version 1.4, the `.delay()` method allows us to delay the execution of functions that follow it in the queue. It can be used with the standard effects queue or with a custom queue. Only subsequent events in a queue are delayed; for example this will *not* delay the no-arguments forms of `.show()` or `.hide()` which do not use the effects queue.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings 'fast' and 'slow' can be supplied to indicate durations of 200 and 600 milliseconds, respectively.

```
<html>
<head>
<style>
div { position: absolute; width: 60px; height: 60px; float: left; }
.first { background-color: #3f3; left: 0;}
.second { background-color: #33f; left: 80px;}
</style>
<script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
<p><button>Run</button></p>
<div class="first"></div>
<div class="second"></div>

<script>
  $("button").click(function() {
    $("div.first").slideUp(300).delay(800).fadeIn(400);
    $("div.second").slideUp(300).fadeIn(400);
  });
</script>

</body>
</html>
```

## Fading

`.fadeIn()`

`.fadeIn( [duration] [, callback] )`

**Description:** Display the matched elements by fading them to opaque.

`.fadeIn( [duration] [, callback] )`

**Duration**A string or number determining how long the animation will run.

**Callback**A function to call once the animation is complete.

`.fadeIn( [duration] [, easing] [, callback] )`

**Duration**A string or number determining how long the animation will run.

**Easing**A string indicating which easing function to use for the transition.

**Callback**A function to call once the animation is complete.

The `.fadeIn()` method animates the opacity of the matched elements. Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings 'fast' and 'slow' can be supplied to indicate durations of 200 and 600 milliseconds, respectively. If any other string is supplied, or if the duration parameter is omitted, the default duration of 400 milliseconds is used.

## Example

```
<html>
<head>
  <style>
    span { color:red; cursor:pointer; }
    div { margin:3px; width:80px; display:none;
      height:80px; float:left; }
    div#one { background:#f00; }
    div#two { background:#0f0; }
    div#three { background:#00f; }
  </style>
  <script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
  <span>Click here...</span>

  <div id="one"></div>
  <div id="two"></div>
  <div id="three"></div>
<script>
  $(document.body).click(function () {
    $("div:hidden:first").fadeIn("slow");
  }); </script></body>
</html>
```

**.fadeOut( [duration] [, callback] )**

**Description:** Hide the matched elements by fading them to transparent.

**.fadeOut( [duration] [, callback] )**

**duration**A string or number determining how long the animation will run.

**callback**A function to call once the animation is complete.

- **.fadeOut( [duration] [, easing] [, callback] )**

**duration**A string or number determining how long the animation will run.

**easing**A string indicating which easing function to use for the transition.

**callback**A function to call once the animation is complete.

The **.fadeOut()** method animates the opacity of the matched elements. Once the opacity reaches 0, the **display** style property is set to **none**, so the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings **'fast'** and **'slow'** can be supplied to indicate durations of 200 and 600 milliseconds, respectively. If any other string is supplied, or if the duration parameter is omitted, the default duration of 400 milliseconds is used.

**.fadeTo()**

**.fadeTo( duration, opacity [, callback] )**

# TOPS Technologies

---

**Description:** Adjust the opacity of the matched elements.

- **.fadeTo( duration, opacity [, callback] )**

**duration**A string or number determining how long the animation will run.

**opacity**A number between 0 and 1 denoting the target opacity.

**callback**A function to call once the animation is complete.

- **.fadeTo( duration, opacity [, easing] [, callback] )**

**duration**A string or number determining how long the animation will run.

**opacity**A number between 0 and 1 denoting the target opacity.

**easing**A string indicating which easing function to use for the transition.

**callback**A function to call once the animation is complete.

The .fadeTo() method animates the opacity of the matched elements.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings 'fast' and 'slow' can be supplied to indicate durations of 200 and 600 milliseconds, respectively. If any other string is supplied, the default duration of 400 milliseconds is used. Unlike the other effect methods, .fadeTo() requires that duration be explicitly specified.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but this is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

## Example

```
<html>
<head>
  <scriptsrc="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
  <p>Click this paragraph to see it fade.</p>
  <p>Compare to this one that won't fade.</p>
  <script>
    $("p:first").click(function(){
      $(this).fadeTo("slow",0.33);
    });
  </script>

</body>
</html>
```

**.fadeToggle()**

**.fadeToggle( [duration] [, easing] [, callback] )**

**Description:** Display or hide the matched elements by animating their opacity.

**.fadeToggle( [duration] [, easing] [, callback] )**

**duration**A string or number determining how long the animation will run.

**easing**A string indicating which easing function to use for the transition.

**callback**A function to call once the animation is complete.

**Example**

```
<html>
<head>
  <scriptsrc="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>

  <button>fadeToggle p1</button>
  <button>fadeToggle p2</button>
  <p>This paragraph has a slow, linear fade.</p>

  <p>This paragraph has a fast animation.</p>
  <divid="log"></div>

  <script>
    $("button:first").click(function(){
      $("p:first").fadeToggle("slow","linear");
    });
    $("button:last").click(function(){
      $("p:last").fadeToggle("fast",function(){
        $("#log").append("<div>finished</div>");
      });
    });
  </script>

</body>
</html>
```

The `.fadeToggle()` method animates the opacity of the matched elements. When called on a visible element, the element's `display` style property is set to `none` once the opacity reaches 0, so the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of 200 and 600 milliseconds, respectively.

## keyPress()

The keypress event is sent to an element when the browser registers keyboard input. This is similar to the keydown event, except in the case of key repeats. If the user presses and holds a key, a keydown event is triggered once, but separate keypress events are triggered for each inserted character. In addition, modifier keys (such as Shift) trigger keydown events but not keypress events.

## Syntax

**.keypress( handler(eventObject) )**

## Example

```
<html>
<head>
<script src="http://code.jquery.com/jquery-latest.js"></script>
<script>
    $(document).ready(function () {
        $("#fname").keypress(function (e) {
            if (e.which >= 65 && e.which <= 90) {
                return true;
            }
            else {
                return false;
            }
        });
    });
</script>
</head>
<body>
<input type="text" id="fname" />
</body>
</html>
```

# Bootstrap

## What is Twitter Bootstrap

Bootstrap is a powerful front-end framework for faster and easier web development. It includes HTML and CSS based design templates for common user interface components like Typography, Forms, Buttons, Tables, Navigations, Dropdowns, Alerts, Modals, Tabs, Accordion, Carousel and many other as well as optional JavaScript extensions.

There are two versions available for download, **compiled Bootstrap** and **Bootstrap source** files. We can download it from [www.getbootstrap.com](http://www.getbootstrap.com)

Compiled download contain complied and minified version of CSS and JavaScript files as well as Sprite Images for faster and easy web development, while the source download contain original source files for all CSS and JavaScript, along with a local copy of the docs.

For the purpose of better understanding we'll focus on the compiled Bootstrap files. It saves your time because you don't have to bother every time including separate files for individual functionality.

## Precompiled Bootstrap

Once downloaded, unzip the compressed folder to see the structure of (the compiled) Bootstrap. You'll see something like this:

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   ├── bootstrap-theme.min.css
│   └── bootstrap-theme.min.css.map
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphs-halflings-regular.eot
    ├── glyphs-halflings-regular.svg
    ├── glyphs-halflings-regular.ttf
    ├── glyphs-halflings-regular.woff
    └── glyphs-halflings-regular.woff2
```

This is the most basic form of Bootstrap: precompiled files for quick drop-in usage in nearly any web project. We provide compiled CSS and JS (bootstrap.\*), as well as compiled and minified CSS and JS (bootstrap.min.\*). CSS source maps (bootstrap.\*.map) are available for use with certain browsers' developer tools. Fonts from Glyphicons are included, as is the optional Bootstrap theme.

## Basic template

Start with this basic HTML template, or modify these examples. We hope you'll customize our templates and examples, adapting them to suit your needs.

```
<html lang="en">
<head>
<title>Bootstrap 101 Template</title>
<link href="css/bootstrap.min.css" rel="stylesheet">
</head><body>
<h1>Hello, world!</h1>
<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<!-- Include all compiled plugins (below), or include individual files as needed -->
<script src="js/bootstrap.min.js"></script>
</body>
</html>
```



## Grid System

### Old concept of grid

Bootstrap's grid system allows up to 12 columns across the page.

If you do not want to use all 12 column individually, you can group the columns together to create wider columns:

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4				span 4				span 4			
span 4				span 8							
span 6						span 6					
span 12											

Bootstrap' grid system is responsive, and the columns will re-arrange depending on the screen size: On a big screen it might look better with the content organized in three columns, but on a small screen it would be better if the content items where stacked on top of each other.

### New concept of grid

Using a single set of `.col-md-*` grid classes, you can create a basic grid system that starts out stacked on mobile devices and tablet devices (the extra small to small range) before becoming horizontal on desktop (medium) devices. Place grid columns in any `.row`.

#### Example: Stacked-to-horizontal

.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1
.col-md-8									.col-md-4			
.col-md-4				.col-md-4				.col-md-4				
.col-md-6						.col-md-6						

#### Example: Mobile and desktop

.col-xs-12 .col-md-8						.col-xs-6 .col-md-4					
.col-xs-6 .col-md-4				.col-xs-6 .col-md-4				.col-xs-6 .col-md-4			
.col-xs-6						.col-xs-6					

### Grid System Rules

Some Bootstrap grid system rules:

- Rows must be placed within a `.container` (fixed-width) or `.container-fluid` (full-width) for proper alignment and padding
- Use rows to create horizontal groups of columns
- Content should be placed within columns, and only columns may be immediate children of rows

- Predefined classes like `.row` and `.col-sm-4` are available for quickly making grid layouts
- Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on `.rows`
- Grid columns are created by specifying the number of 12 available columns you wish to span. For example, three equal columns would use three `.col-sm-4`

The classes above can be combined to create more dynamic and flexible layouts.

**Tip:** Each class scales up, so if you wish to set the same widths for `xs` and `sm`, you only need to specify `xs`.

## Containers

- Bootstrap requires a containing element to wrap site contents and house our grid system. You may choose one of two containers to use in your projects. Note that, due to padding and more, neither container is nestable.
- Use `.container` for a responsive fixed width container.
  - `<div class="container"> ...</div>`
- Use `.container-fluid` for a full width container, spanning the entire width of your viewport.
  - `<div class="container-fluid">...</div>`

## How To Format DIV?

```
<div class="row">
  <div class="col-sm-9">
    Level 1: .col-sm-9
    <div class="row">
      <div class="col-xs-8 col-sm-6">
        Level 2: .col-xs-8 .col-sm-6
      </div>
      <div class="col-xs-4 col-sm-6">
        Level 2: .col-xs-4 .col-sm-6
      </div>
    </div>
  </div>
</div>
```

## Grid Options

The following table summarizes how the Bootstrap grid system works across multiple devices:

	Extra small devices Phones (<768px)	Small devices Tablets (>=768px)	Medium devices Desktops (>=992px)	Large devices Desktops (>=1200px)
<b>Grid behaviour</b>	Horizontal at all times	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints
<b>Container width</b>	None (auto)	750px	970px	1170px
<b>Class prefix</b>	.col-xs-	.col-sm-	.col-md-	.col-lg-
<b>Number of columns</b>	12	12	12	12
<b>Column width</b>	Auto	~62px	~81px	~97px
<b>Gutter width</b>	30px (15px on each side of a column)	30px (15px on each side of a column)	30px (15px on each side of a column)	30px (15px on each side of a column)
<b>Nestable</b>	Yes	Yes	Yes	Yes
<b>Offsets</b>	Yes	Yes	Yes	Yes
<b>Column ordering</b>	Yes	Yes	Yes	Yes

Bootstrap includes a responsive, mobile first fluid grid system that appropriately scales up to 12 columns as the device or viewport size increases. It includes predefined classes for easy layout options, as well as powerful mixins for generating more semantic layouts.

## Introduction

Grid systems are used for creating page layouts through a series of rows and columns that house your content. Here's how the Bootstrap grid system works:

- Rows must be placed within a .container (fixed-width) or .container-fluid (full-width) for proper alignment and padding.
- Use rows to create horizontal groups of columns.
- Content should be placed within columns, and only columns may be immediate children of rows.
- Predefined grid classes like .row and .col-xs-4 are available for quickly making grid layouts. Less mixins can also be used for more semantic layouts.
- Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on .rows.
- The negative margin is why the examples below are outdented. It's so that content within grid columns is lined up with non-grid content.

# TOPS Technologies

- Grid columns are created by specifying the number of twelve available columns you wish to span. For example, three equal columns would use three `.col-xs-4`.
- If more than 12 columns are placed within a single row, each group of extra columns will, as one unit, wrap onto a new line.
- Grid classes apply to devices with screen widths greater than or equal to the breakpoint sizes, and override grid classes targeted at smaller devices. Therefore, e.g. applying any `.col-md-*` class to an element will not only affect its styling on medium devices but also on large devices if a `.col-lg-*` class is not present.

Look to the examples for applying these principles to your code.

## Responsive

- **Enabling responsive features**
- Turn on responsive CSS in your project by including the proper meta tag and additional stylesheet within the `<head>` of your document. If you've compiled Bootstrap from the Customize page, you need only include the meta tag.
- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`
- `<link href="assets/css/bootstrap-responsive.css" rel="stylesheet">`
- Media queries allow for custom CSS based on a number of conditions—ratios, widths, display type, etc—but usually focuses around min-width and max-width.
- Modify the width of column in our grid
- Stack elements instead of float wherever necessary
- Resize headings and text to be more appropriate for devices

Label	Layout width	Column width	Gutter width
Large display	1200px and up	70px	30px
Default	980px and up	60px	20px
Portrait tablets	768px and above	42px	20px
Phones to tablets	767px and below	Fluid columns, no fixed widths	
Phones	480px and below	Fluid columns, no fixed widths	

## Responsive utility classes

For faster mobile-friendly development, use these utility classes for showing and hiding content by device. Below is a table of the available classes and their effect on a given media query layout (labeled by device). They can be found in `responsive.less`.

# TOPS Technologies

Class	Phones 767px and below	Tablets 979px to 768px	Desktops Default
<b>.visible-phone</b>	Visible	Hidden	Hidden
<b>.visible-tablet</b>	Hidden	Visible	Hidden
<b>.visible-desktop</b>	Hidden	Hidden	Visible
<b>.hidden-phone</b>	Hidden	Visible	Visible
<b>.hidden-tablet</b>	Visible	Hidden	Visible
<b>.hidden-desktop</b>	Visible	Visible	Hidden

## Typography

- The classes for text colors are: .text-muted, .text-primary, .text-success, .text-info, .text-warning, and .text-danger
- The classes for background colors are: .bg-primary, .bg-success, .bg-info, .bg-warning, and .bg-danger

Class	Description
<b>.lead</b>	Makes a paragraph stand out
<b>.small</b>	Indicates smaller text (set to 85% of the size of the parent)
<b>.text-left</b>	Indicates left-aligned text
<b>.text-center</b>	Indicates center-aligned text
<b>.text-right</b>	Indicates right-aligned text
<b>.text-justify</b>	Indicates justified text
<b>.text-nowrap</b>	Indicates no wrap text
<b>.text-lowercase</b>	Indicates lowercased text
<b>.text-uppercase</b>	Indicates uppercased text
<b>.text-capitalize</b>	Indicates capitalized text
<b>.initialism</b>	Displays the text inside an <abbr> element in a slightly smaller font size
<b>.list-unstyled</b>	Removes the default list-style and left margin on list items (works on both <ul> and <ol>). This class only applies to immediate children list items (to remove the default list-style from any nested lists, apply this class to any nested lists as well)
<b>.list-inline</b>	Places all list items on a single line
<b>.dl-horizontal</b>	Lines up the terms (<dt>) and descriptions (<dd>) in <dl> elements side-by-side. Starts off like default <dl>s, but when the browser window expands, it will line up side-by-side
<b>.pre-scrollable</b>	Makes a <pre> element scrollable

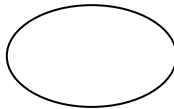
## Table

- A basic Bootstrap table has a light padding and only horizontal dividers. The `.table` class adds basic styling to a table
- The `.table-striped` class adds zebra-stripes to a table
- The `.table-bordered` class adds borders on all sides of the table and cells
- The `.table-hover` class enables a hover state on table rows
- The `.table-condensed` class makes a table more compact by cutting cell padding in half
- Contextual classes can be used to color table rows (`<tr>`) or table cells (`<td>`)
  - `.active`
  - `.success`
  - `.info`
  - `.warning`
  - `.danger`

## Image Shapes



- The `.img-rounded` class adds rounded corners to an image (IE8 do not support rounded corners):
  - ``



- The `.img-rounded` class adds rounded corners to an image (IE8 do not support rounded corners):
  - ``



- The `.img-rounded` class adds rounded corners to an image (IE8 do not support rounded corners):
  - ``

## Button

### Button Styles

To achieve the button styles above, Bootstrap has the following classes

- `.btn-default`
- `.btn-primary`
- `.btn-success`
- `.btn-info`
- `.btn-warning`
- `.btn-danger`
- `.btn-link`

Ex :- `<a href="#" class="btn btn-default">Click Me</a>`

### Button Sizes

- To achieve the button styles above, Bootstrap has the following classes:
- To achieve the button styles above, Bootstrap has the following classes:
  - `.btn-lg`
  - `.btn-md`
  - `.btn-sm`
  - `.btn-xs`



- To achieve the button styles above, Bootstrap has the following classes:

## Block Level Buttons

- Add class **.btn-block** to create a block level button:

## Active/Disabled Buttons

- The class **.active** makes a button appear pressed, and the class **.disabled** makes a button unclickable

## Button Groups

- Bootstrap allows you to group a series of buttons together (on a single line) in a button group:
  - Use a `<div>` element with class `.btn-group` to create a button group:

```
<div class="btn-group">
  <button type="button" class="btn btn-primary">Apple</button>
  <button type="button" class="btn btn-primary">Samsung</button>
  <button type="button" class="btn btn-primary">Sony</button>
</div>
```

## Vertical Button Groups

- Bootstrap also supports vertical button groups:
  - Use the class `.btn-group-vertical` to create a vertical button group:

```
<div class="btn-group-vertical">
  <button type="button" class="btn btn-primary">Apple</button>
  <button type="button" class="btn btn-primary">Samsung</button>
  <button type="button" class="btn btn-primary">Sony</button>
</div>
```

## Nesting Button Groups & Drop Down Menus

- Nest button groups to create drop down menus:

```
<div class="btn-group">
  <button type="button" class="btn btn-primary">Apple</button>
  <button type="button" class="btn btn-primary">Samsung</button>
  <div class="btn-group">
    <button type="button" class="btn btn-primary dropdown-
toggle" data-toggle="dropdown">
      Sony
      <span class="caret"></span>
    </button>
    <ul class="dropdown-menu" role="menu">
      <li><a href="#">Tablet</a></li>
      <li><a href="#">Smartphone</a></li>
    </ul>
  </div>
</div></div>
```

## Glyphicons

Includes over 250 glyphs in font format from the Glyph icon Halflings set. Glyph icons Halflings are normally not available for free, but their creator has made them available for Bootstrap free of cost. As a thank you, we only ask that you include a

link back to Glyph icons whenever possible.



## How to use

For performance reasons, all icons require a base class and individual icon class. To use, place the following code just about anywhere. Be sure to leave a space between the icon and text for proper padding.

Ex :-

```
<span class="glyphicon glyphicon-search" aria-hidden="true"></span>
```

## Progress Bar With Label

- Default progress bar.
- Remove the `<span>` with `.sr-only` class from within the progress bar to show a visible percentage.
- To ensure that the label text remains legible even for low percentages, consider adding a `min-width` to the progress bar.
- Progress bars use some of the same button and alert classes for consistent styles.
  - `.progress-bar`
  - `.progress-bar-success`
  - `.progress-bar-info`
  - `.progress-bar-warning`
  - `.progress-bar-danger`
- Uses a gradient to create a striped effect. Not available in IE9 and below.
  - `progress-bar-striped`
- Add `.active` to `.progress-bar-striped` to animate the stripes right to left. Not available in IE9 and below.

```
<div class="progress">  
  <div class="progress-bar progress-bar-striped active" style="width: 45%">  
    <span class="sr-only">45% Complete</span>  
  </div>
```

## Basic Pagination

If you have a web site with lots of pages, you may wish to add some sort of pagination to each page. A basic pagination in Bootstrap looks like this:

- **Basic Pagination & Pagination Sizing**

- .pagination
  - Ex:-  
`<ul class="pagination"><li><a href="#">1</a></li></ul>`
- .pagination-lg
  - Ex:-  
`<ul class="pagination-lg"><li><a href="#">1</a></li></ul>`
- .pagination-sm
  - Ex:-  
`<ul class="pagination-sm"><li><a href="#">1</a></li></ul>`

- **Active State & Disabled State**

- .active
  - Ex:-  
`<ul class="pagination-sm">  
 <li class="active"><a href="#">1</a></li>  
</ul>`
- .disabled
  - Ex:-  
`<ul class="pagination-sm">  
 <li class="disabled"><a href="#">1</a></li>  
</ul>`

## List Groups

- **Basic Pagination & Pagination Sizing**

- To create a basic list group, use an `<ul>` element with class `.list-group`, and `<li>` elements with class `.list-group-item`
  - Ex:-  
`<ul class="list-group">  
 <li class="list-group-item">First item</li>  
</ul>`

- **List Group With Badges**

- .To create a badge, create a `<span>` element with class `.badge` inside the list item
  - Ex:-  
`<ul class="list-group">  
 <li class="list-group-item"><span class="badge">12</span>New</li></ul>`

- **Active State & Disabled State**

- .active
  - Ex:-  
`<li class="active"><a href="#">1</a></li>`  
or  
`<li class="disabled"><a href="#">1</a></li>`

## Form Inputs

Form controls automatically receive some global styling with Bootstrap:

All textual `<input>`, `<textarea>`, and `<select>` elements with class `.form-control` have a width of 100%.

**Bootstrap provides three types of form layouts:**

- Vertical form (this is default)
- Horizontal form
- Inline form

**Standard rules for all three form layouts:**

- Always use `<form role="form">` (helps improve accessibility for people using screen readers)
- Wrap labels and form controls in `<div class="form-group">` (needed for optimum spacing)
- Add class `.form-control` to all textual `<input>`, `<textarea>`, and `<select>` elements

## Supported Form Controls

Bootstrap supports the following form controls:

- Input - `.form-control`
- Textarea - `.form-control`
- Checkbox - `<label class="checkbox-inline"><input type="checkbox">Option 1</label>`
- Radio - `<label class="radio-inline"><input type="radio">Option 1</label>`
- Select - `.form-control`
- Height Sizing - `.input-lg` or `.input-sm`

## Bootstrap Input

Bootstrap supports all the HTML5 input types: text, password, datetime, datetime-local, date, month, time, week, number, email, url, search, tel, and color.

**Note:** Inputs will NOT be fully styled if their type is not properly declared!

## Horizontal Form

A horizontal form stands apart from the other forms both in the amount of markup, and in the presentation of the form.

Additional rules for a horizontal form:

- Add class `.form-horizontal` to the `<form>` element
- Add class `.control-label` to all `<label>` elements

**Tip:** Use Bootstrap's predefined grid classes to align labels and groups of form controls in a horizontal layout.

The following example creates a horizontal form with two input fields, one checkbox, and one submit button

## Bootstrap Form Control States

- **INPUT FOCUS** - The outline of the input is removed and a box-shadow is applied on focus
- **DISABLED INPUTS** - Add a disabled attribute to disable an input field
- **DISABLED FIELDSETS** - Add a disabled attribute to a fieldset to disable all controls within
- **READONLY INPUTS** - Add a readonly attribute to an input to prevent user input
- **VALIDATION STATES** - Bootstrap includes validation styles for error, warning, and success messages. To use, add .has-warning, .has-error, or .has-success to the parent element
- **ICONS** - You can add feedback icons with the .has-feedback class and an icon
- **HIDDEN LABELS** - Add a .sr-only class on non-visible labels

The following example demonstrates some of the form control states above in a Horizontal form:

```
<form class="form-horizontal" role="form"><div class="form-group">
<label class="col-sm-2 control-label">Focused</label>
<div class="col-sm-10">
<input class="form-control" id="focusedInput" type="text" value="Click to focus">
</div>
</div>
<div class="form-group">
<label for="inputPassword" class="col-sm-2 control-label">Disabled</label>
<div class="col-sm-10">
<input class="form-control" id="disabledInput" type="text" disabled>
</div>
</div>
<fieldset disabled>
<div class="form-group">
<label for="disabledTextInput" class="col-sm-2 control-label">Fieldset
disabled</label>
<div class="col-sm-10">
<input type="text" id="disabledTextInput" class="form-control">
</div></div><div class="form-group">
<label for="disabledSelect" class="col-sm-2 control-label"></label>
<div class="col-sm-10">
<select id="disabledSelect" class="form-control">
<option>Disabled select</option>
</select>
</div>
</div>
</fieldset>
```

```
<div class="form-group has-success has-feedback">
<label class="col-sm-2 control-label" for="inputSuccess">
  Input with success and icon</label>
<div class="col-sm-10">
<input type="text" class="form-control" id="inputSuccess">
<span class="glyphicon glyphicon-ok form-control-feedback"></span>
</div>
</div>
<div class="form-group has-warning has-feedback">
<label class="col-sm-2 control-label" for="inputWarning">
  Input with warning and icon</label>
<div class="col-sm-10">
<input type="text" class="form-control" id="inputWarning">
<span class="glyphicon glyphicon-warning-sign form-control-feedback"></span>
</div>
</div>
<div class="form-group has-error has-feedback">
<label class="col-sm-2 control-label" for="inputError">
  Input with error and icon</label>
<div class="col-sm-10">
<input type="text" class="form-control" id="inputError">
<span class="glyphicon glyphicon-remove form-control-feedback"></span>
</div></div></form>
```

And here is an example of some of the form control states in an Inline form:

```
<form class="form-inline" role="form">
<div class="form-group">
<label for="focusedInput">Focused</label>
<input class="form-control" id="focusedInput" type="text">
</div>
<div class="form-group">
<label for="inputPassword">Disabled</label>
<input class="form-control" id="disabledInput" type="text" disabled>
</div>
<div class="form-group has-success has-feedback">
<label for="inputSuccess2">Input with success</label>
<input type="text" class="form-control" id="inputSuccess2">
<span class="glyphicon glyphicon-ok form-control-feedback"></span>
</div>
<div class="form-group has-warning has-feedback">
<label for="inputWarning2">Input with warning</label>
<input type="text" class="form-control" id="inputWarning2">
<span class="glyphicon glyphicon-warning-sign form-control-feedback"></span>
</div>
<div class="form-group has-error has-feedback">
<label for="inputError2">Input with error</label>
<input type="text" class="form-control" id="inputError2">
<span class="glyphicon glyphicon-remove form-control-feedback"></span>
</div>
</form>
```

## Navbar

### Default navbar

Navbars are responsive meta components that serve as navigation headers for your application or site. They begin collapsed (and are toggle able) in mobile views and become horizontal as the available viewport width increases.

Ex:-

```
<nav class="navbar navbar-default">
<div class="container-fluid">
<!-- Brand and toggle get grouped for better mobile display -->
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
target="#bs-example-navbar-collapse-1" aria-expanded="false">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="#">Brand</a>
</div>

<!-- Collect the nav links, forms, and other content for toggling -->
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
<ul class="nav navbar-nav">
<li class="active"><a href="#">Link <span class="sr-only">(current)</span></a></li>
<li><a href="#">Link</a></li>
<li class="dropdown">
<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-
haspopup="true" aria-expanded="false">Dropdown <span
class="caret"></span></a>
<ul class="dropdown-menu">
<li><a href="#">Action</a></li>
<li><a href="#">Another action</a></li>
<li><a href="#">Something else here</a></li>
<li role="separator" class="divider"></li>
<li><a href="#">Separated link</a></li>
<li role="separator" class="divider"></li>
<li><a href="#">One more separated link</a></li>
</ul>
</li>
</ul>
<form class="navbar-form navbar-left" role="search">
<div class="form-group">
<input type="text" class="form-control" placeholder="Search">
</div>
<button type="submit" class="btn btn-default">Submit</button>
```



```
</form>
<ul class="nav navbar-nav navbar-right">
<li><a href="#">Link</a></li>
<li class="dropdown">
<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-
haspopup="true" aria-expanded="false">Dropdown <span
class="caret"></span></a>
<ul class="dropdown-menu">
<li><a href="#">Action</a></li>
<li><a href="#">Another action</a></li>
<li><a href="#">Something else here</a></li>
<li role="separator" class="divider"></li>
<li><a href="#">Separated link</a></li>
</ul>
</li>
</ul>
</div><!-- /.navbar-collapse -->
</div><!-- /.container-fluid -->
</nav>
```

## Inverted navbar

Modify the look of the navbar by adding `.navbar-inverse`.

```
<nav class="navbar navbar-inverse">
...
</nav>
```

## Breadcrumbs

Indicate the current page's location within a navigational hierarchy. Separators are automatically added in CSS through `:before` and `content`.

```
<ol class="breadcrumb">
<li><a href="#">Home</a></li>
<li><a href="#">Library</a></li>
<li class="active">Data</li>
</ol>
```

## The Modal Plugin

### How To Create a Modal

The following example shows how to create a basic modal:

```
<!-- Trigger the modal with a button -->
<button type="button" class="btn btn-info btn-lg" data-toggle="modal" data-
target="#myModal">Open Modal</button>
```

```
<!-- Modal -->
<div class="modal fade" id="myModal" role="dialog" aria-
labelledby="myModalLabel" aria-hidden="true">
  <div class="modal-dialog">
    <!-- Modal content-->
    <div class="modal-content" style="height:200px;">
    </div>
  </div>
</div>
```

## The "Modal" part:

- The parent <div> of the modal must have an ID that is the same as the value of the data-target attribute used to trigger the modal ("myModal").
- The .modal class identifies the content of <div> as a modal and brings focus to it.
- The .fade class adds a transition effect which fades the modal in and out. Remove this class if you do not want this effect.
- The attribute role="dialog" improves accessibility for people using screen readers.
- The .modal-dialog class sets the proper width and margin of the modal.

## The "Modal content" part:

The <div> with class="modal-content" styles the modal (border, background-color, etc.). Inside this <div>, add the modal's header, body, and footer.

The .modal-header class is used to define the style for the header of the modal.

The <button> inside the header has a data-dismiss="modal" attribute which closes the modal if you click on it. The .close class styles the close button, and the .modal-title class styles the header with a proper line-height.

The .modal-body class is used to define the style for the body of the modal. Add any HTML markup here; paragraphs, images, videos, etc.

The .modal-footer class is used to define the style for the footer of the modal. Note that this area is right aligned by default.

## Tooltip

Inspired by the excellent jQuery.tipsy plugin written by Jason Frame; Tooltips are an updated version, which don't rely on images, use CSS3 for animations, and data-attributes for local title storage.

### Ex:-

- <a href="#" data-toggle="tooltip" data-placement="top" title="Hooray!">Hover</a>
- <a href="#" data-toggle="tooltip" data-placement="bottom" title="Hooray!">Hover</a>
- <a href="#" data-toggle="tooltip" data-placement="left" title="Hooray!">Hover</a>
- <a href="#" data-toggle="tooltip" data-placement="right" title="Hooray!">Hover</a>

**Note:** Tooltips must be initialized with jQuery: select the specified element and call the tooltip() Method. The following code will enable all tooltips in the document:

## JS Cord

```
<script>
  $(document).ready(function(){
    $('[data-toggle="tooltip"]').tooltip();
  });
</script>
```

## Popover

To create a popover, add the data-toggle="popover" attribute to an element.

Use the title attribute to specify the header text of the popover, and use the data-content attribute to specify the text that should be displayed inside the popover's body:

### Ex:-

- `<a href="#" title="Header" data-toggle="popover" data-placement="top" data-content="Content">Click</a>`
- `<a href="#" title="Header" data-toggle="popover" data-placement="bottom" data-content="Content">Click</a>`
- `<a href="#" title="Header" data-toggle="popover" data-placement="left" data-content="Content">Click</a>`
- `<a href="#" title="Header" data-toggle="popover" data-placement="right" data-content="Content">Click</a>`

**Note:** Popovers must be initialized with jQuery: select the specified element and call the popover() method. The following code will enable all popovers in the document:

## JS Cord

```
<script>
  $(document).ready(function(){
    $('[data-toggle="popover"]').popover();
  });
</script>
```

## Alert messages alert.js

Add dismiss functionality to all alert messages with this plugin.

When using a .close button, it must be the first child of the .alert-dismissible and no text content may come before it in the markup.

```
<button type="button" class="close" data-dismiss="alert" aria-label="Close">
<span aria-hidden="true">&times;</span>
</button>
```

## JS Cord

```
$('#myAlert').on('closed.bs.alert', function () {
  // do something...
})
```

## Alert messages alert.js

Add data-toggle="button" to activate toggling on a single button.

Pre-toggled buttons need .active and aria-pressed="true"

For pre-toggled buttons, you must add the .active class and the aria-pressed="true" attribute to the button yourself.

```
<button type="button" class="btn btn-primary" data-toggle="button" aria-pressed="false" autocomplete="off">  
Single toggle  
</button>
```

## Checkbox / Radio

Add data-toggle="buttons" to a .btn-group containing checkbox or radio inputs to enable toggling in their respective styles.

Preselected options need .active

For preselected options, you must add the .active class to the input's label yourself.

### Visual checked state only updated on click

If the checked state of a checkbox button is updated without firing a click event on the button (e.g. via <input type="checkbox" checked=""> or via setting the checked property of the input), you will need to toggle the .active class on the input's label yourself.

```
<div class="btn-group" data-toggle="buttons">  
  <label class="btn btn-primary active">  
    <input type="checkbox" autocomplete="off" checked=""> Checkbox 1  
  (pre-checked)  
  </label>  
  <label class="btn btn-primary">  
    <input type="checkbox" autocomplete="off"> Checkbox 2  
  </label>  
  <label class="btn btn-primary">  
    <input type="checkbox" autocomplete="off"> Checkbox 3  
  </label>  
</div>
```

## Collapse collapse.js

Flexible plugin that utilizes a handful of classes for easy toggle behavior.

Plugin dependency

Collapse requires the transitions plugin to be included in your version of Bootstrap.

## Example

Click the buttons below to show and hide another element via class changes:

- .collapse hides content
- .collapsing is applied during transitions
- .collapse.in shows content

You can use a link with the href attribute, or a button with the data-target attribute. In both cases, the data-toggle="collapse" is required.

```
<a class="btn btn-primary" role="button" data-toggle="collapse"
href="#collapseExample" aria-expanded="false" aria-controls="collapseExample">
  Link with href
</a>

<button class="btn btn-primary" type="button" data-toggle="collapse" data-
target="#collapseExample" aria-expanded="false" aria-controls="collapseExample">
  Button with data-target
</button>

<div class="collapse" id="collapseExample">
  <div class="well">
    ...
  </div>
</div>
```

## Customize CSS in BootStrap

And so finally we get to the last tutorial in the series, which will show you how to create custom color sets.

If you recall back at the beginning, when we were going through the major changes, I mentioned that the BS authors had provided an additional file to make BS3 look just like BS2, rather than using its flat look.

Throughout this series, we've used the default BS3 flat look, but if you download the prebuilt JavaScript version of BS3 from the Bootstrap website, you'll find a new file inside the archive you download called bootstrap-theme.css.

If you link this file into your project immediately after your inclusion of the core bootstrap.css file, you'll find that even though you're now using the new BS3, the look and feel of your application still resembles BS2.

The authors realized that one of the main barriers to the adoption of BS3 in new applications was the inability to style it in a custom manner, but with ease.

If you were a user of LESS, or even Sass (as BS3 now has Sass bindings), then this wasn't an issue. You simply opened up the LESS sources, tweaked the variables and

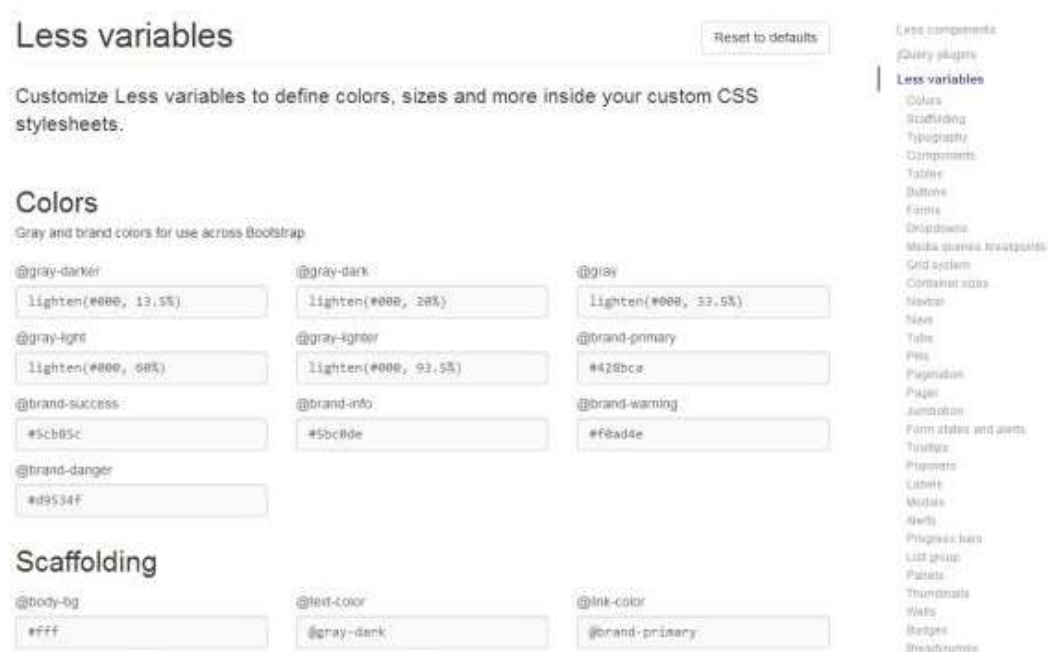
# TOPS Technologies

Mixins you needed, and ran things through the LESS compiler to get your new CSS script.

Unfortunately, not everyone used LESS; in fact, many developers and designers only had the ability to download and include the plain old pre-compiled CSS and JavaScript files, so a better way had to be found.

The first change was opening up the entire Less sources to an in-page customization tool directly on the Bootstrap main site, but this isn't a new thing; you were able to use this page before in a limited way. With BS3, however, the Less customization tool has had a complete overhaul, and you can now redefine everything that BS3 uses, from font sizes and typefaces, right through to grid sizes, trigger points, and basic contextual color sets.

In fact, there is now nothing that cannot be changed before you decide to download your new customized CSS, as the following image shows:



Because of the sheer size of the tool, it's impossible to show the entire thing in this tutorial, but it's easy enough to access. Simply go to [the Bootstrap website](#) and click **Customize** in the top menu bar. You'll also see that you have many other options, such as which JavaScript plugins and toolkits to include, which components to include, and base style that you may not want.

For example, if all you want to use is the grid system, and nothing else, then you can simply select only the grid system, and unselect all the other components.

The BS3 site will then generate just the required code to include, and no more. This is a boon for those people who complain that all Bootstrap sites look the same, because it means that your site absolutely does not have to look the same as the rest—you can just use the bits you need, and use your own stuff for everything else.

There are two other ways you can customize your build. The first is to take the additional 'bootstrap-theme' CSS style sheet, make a copy, and then change the

styles as you see fit. This is not as easy as using the customization tools, but it's also not as difficult as the alternative.

Most of the class names and settings that you'll want to change to stamp your personal mark in BS3 are already separated out in the BS2 theme, so the quickest way to experiment is simply to put together a prototyping page with the main controls and elements you want to change, and then link in your copy of the BS2 theme.

If you're using Node and something like Bower, it gets even easier, because you can use live reload, then just watch your sample page change in near real time as you tweak your custom version of the theme sheet.

The second way is slightly more involved, and as described in the BS3 docs, comes in two flavors: light customizations and heavy customizations.

## Flash

What is Flash?

Flash is a multimedia graphics program especially for use on the Web.

Flash enables you to create interactive "movies" on the Web. Flash uses vector graphics, which means that the graphics can be scaled to any size without losing clarity/quality. Flash does not require programming skills and is easy to learn.

### Flash vs. Animated Images and Java Applets

Animated images and Java applets are often used to create dynamic effects on Web pages.

The advantages of Flash are:

Flash loads much faster than animated images

Flash allows interactivity, animated images do not

Flash does not require programming skills, Java applets do

## Flash Embedded in HTML

After creating a Flash movie you choose File > Save As from the top menu to save your movie. Save the file as "Somefilename fla".

To embed the Flash movie you just made into an HTML page, you should go back to your Flash program and do the following steps:

**Step 1** Choose File > Open. Open a Flash movie you have created.

**Step 2** Choose File > Export Movie.

**Step 3** Name the file "somefilename.swf". Choose the location where the file is to be stored (in your Web folder). Click OK.

**Step 4** Open the HTML page where you want to insert your Flash movie. Insert this code:

```
<object width="550" height="400">
```

```
<param name="movie" value="somefilename.swf"><embed src="somefilename.swf"
```



`width="550" height="400"></embed></object>`

**Note:** This is the minimum code you need to embed a Flash movie in a browser. A broken icon will appear on the Web page if the user does not have the Flash plug-in installed.

**Note:** In the code above there is both an `<embed>` tag and an `<object>` tag. This is because the `<object>` tag is recognized by Internet Explorer, and Netscape recognizes the `<embed>` tag and ignores the `<object>` tag.

**Step 5**Type in the address of the HTML file in your browser and look at your first Flash movie.

The code above is the absolute minimum code to embed Flash movies in HTML pages. It is not recommended to use the minimum code. There should be a few more attributes added:

classid is an attribute to the `<object>` tag. It tells Internet Explorer to load the ActiveX plug-in if it is not installed plugins page is an attribute to the `<embed>` tag. It displays a link to the Shockwave download page if Netscape does not have it

**The Flash program can add these attributes for you:**

**Step 1**Choose File > Publish. Flash will now create the `<object>`, `<param>`, and `<embed>` tags for you. It will also create the classid and pluginspage attributes.

**Step 2**Open the HTML document that Flash created, view the HTML source and copy the code into your HTML page where you want your Flash movie.

**Step 3**Be sure that you have the "somefilename.swf" in your Web folder.

**Step 4**Type in the address of the HTML file in your browser and look at your first Flash movie.

## Tweening

Tweening comes from the words "in between".

With Tweening you can go from one keyframe to another and specify changes in the animation and let the Flash program create the frames in between.

How To Built Tweening :

**Step 1**Create a small circle to the left in the Stage area. Do this by selecting the circle tool from the left toolbar. Draw the circle in the Stage area.

**Step 2**Select the Arrow tool from the left toolbar. Double-click on the circle to select it.

**Step 3**Now we have to convert the circle to a symbol. When the circle is converted to a symbol we can create instances of the circle. From the top menu choose Modify > Convert to Symbol. Name the symbol "Ball", and select OK.

**Step 4**Go to Frame 10 in the Timeline. Do this by clicking the gray field below 10. Then right click in this field. Choose Insert Keyframe. Keyframes appear as circles in a frame. This operation duplicates the image.

**Note:** A keyframe specifies changes in an animation. You create keyframes at important points in the Timeline and let Flash create the frames in between.

**Step 5**Select the circle and move it to the right a couple of inches.

**Step 6**Click on the Timeline any place between Frame 1 and Frame 10. Then right click and choose Create Motion Tween.

**Step 7**Choose Control > Test Movie from the top menu to test your Flash movie.

## Flash Guide Tweening

With Motion Guide Tweening you can move an object from one location to another along a specified path.

### Steps for built Flash Guide Tweening:

**Step 1** Choose Window > Common Libraries > Graphics. Select the image you want to use. In this example we have used a blue mouse.

**Step 2** Click on the image and drag it outside the left edge of the Stage.

**Step 3** Go to Frame 40 in the Timeline. Do this by clicking the gray field below 40. Then right click in this field. Choose Insert Keyframe. Keyframes appear as circles in a frame. This operation duplicates the image.

**Step 4** Click on the Timeline any place between Frame 1 and Frame 40. Then right click and choose Create Motion Tween.

**Step 5** Right click on Layer 1 (Click on the layer name, where it says "Layer 1"). Choose Add Motion Guide in the pop-up menu. The Flash program will now insert a motion guide layer on top of layer 1. Motion guide layers are used to draw lines an animated symbol should follow.

**Step 6** Click on the Motion Guide Layer to make sure it is the active layer (Click on the layer name, where it says "Guide: Layer 1").

**Step 7** Click on the Pencil tool in the left toolbox. Set the Pencil Mode to Smooth (in the Options section of the left toolbox).

**Step 8** Draw a line. Begin on the image and draw a line to the other side of the Stage.

**Step 9** Go back to Frame 1 in the Timeline. Click on the Arrow tool in the left toolbox. Select the "Snap to Objects" button in the Options section of the left toolbox.

**Step 10** Place the image with its center on the beginning of the motion guide (the black line you have drawn with the Pencil). The center of the image shows as a +. A black circle appears when the image is snapped to the motion guide. Release the mouse button when the image is snapped to the guide.

**Step 11** Go to Frame 40. Place the image with its center on the end of the motion guide.

**Step 12** Choose Control > Test Movie from the top menu to test your Flash movie.

## Flash Tint Tweening

**With Tint Tweening you can change the color of an object.**

### Steps for built Flash Tint Tweening:

**Step 1** Choose Insert > New Symbol.

**Note:** To add Tint effects the object must be a symbol.

**Step 2** Name the symbol "changecolor" and select the Graphic option in Behavior. Click OK.

**Note:** You will now be taken to the symbol generator in the Flash program. Here you create symbols. Symbols can be dragged to the stage of your movie after you have created them.

**Step 3** Choose the Text tool in the left toolbox. Choose Text > Size > 36 from the top

menu to make the text big. Choose Text > Style > Bold to make the text thick.

**Step 4** Click in the work area and write "Color Changing Text".

**Step 5** Jump back to the movie. Do this by choosing Edit > Edit Movie.

**Step 6** Insert the symbol you just created into the movie. Choose Window > Library. Select the "changecolor" symbol and drag it into the middle of the Stage.

**Step 7** Insert a keyframe in Frame 15 and in Frame 30.

**Step 8** Go to Frame 15. Right click on the text in the Stage. In the pop-up menu, choose Panels > Effect.

**Step 9** Choose Tint from the drop down menu. A color map will show. Set the colors to: R=0, G=255, B=0.

**Step 10** Click on the Timeline any place between Frame 1 and Frame 15. Then right click and choose Create Motion Tween.

**Step 11** Click on the Timeline any place between Frame 15 and Frame 30. Then right click and choose Create Motion Tween.

**Step 12** Choose Control > Test Movie from the top menu to test your Flash movie.

## Flash Shape Tweening

With Shape Tweening you can change one object into another.

### Steps for built Flash Shape Tweening

In this example you will learn how to change one object into another.

**Step 1** Choose the Text tool in the left toolbox. Choose Text > Size > 48 from the top menu to make the text big. Choose Text > Style > Bold to make the text thick.

**Step 2** Click in the work area and write "Hello".

**Step 3** Right click on the text you just wrote and choose Panels > Align from the pop-up menu.

**Step 4** In the Align box select the "To Stage" button first. Then click on the "Align Horizontal Center" button and the "Align Vertical Center" button. Close the Align box.

**Step 5** Select the Arrow Tool and click on the text. Choose Modify > Break Apart from the top menu.

**Step 6** Insert keyframes at Frame 24, 50 and 51.

**Step 7** Delete the text "Hello" in Frame 24. Select it and press the Delete button on your keyboard.

**Step 8** Write a new text on the Stage. Write "World!" (Font size: 48, style: bold).

**Step 9** Right click on the text you just wrote and choose Panels > Align from the pop-up menu. In the Align box select the "To Stage" button first. Then click on the "Align Horizontal Center" button and the "Align Vertical Center" button. Close the Align box.

**Step 10** Select the Arrow Tool and click on the text. Choose Modify > Break Apart from the top menu.

**Step 11** Insert a keyframe in Frame 26.

**Step 12** Double click the keyframe in Frame 1. In the small pop-up box click on the Frame tab. Set Tweening to Shape. Close the pop-up box.

**Step 13** Double click the keyframe in Frame 26. In the small pop-up box click on the Frame tab. Set Tweening to Shape. Close the pop-up box.

**Step 14** Double click the keyframe in Frame 51. In the large pop-up box click on the Frame Actions tab. Click on the + sign. Choose Basic Actions > Go To. Close the pop-up boxes.

**Step 15** Choose Control > Test Movie from the top menu to test your Flash movie.

## Steps for built Flash Button1

In this example you will learn how to insert an image, convert it to a button, and add a URL to it so it becomes a link.

**Step 1** Choose File > Import to import an image that will become a button. Locate the image and click Open. The image will be saved in the Library.

**Step 2** Select the image with the Arrow tool.

**Step 3** Convert the image to a symbol. Choose Insert > Convert to Symbol from the top menu. Name the symbol "button", choose Button from the Behavior list and click OK.

**Step 4** Right click on the image. Choose Actions from the pop-up menu.

**Step 5** In the Object Actions box click on the + sign. Choose Basic Actions > Get URL.

**Step 6** Choose target in the Window field. Close the Object Actions box.

**Step 7** Choose Control > Test Movie from the top menu to test your Flash movie.

## Steps for built Flash Button2

**Step 1** Choose Insert > New Symbol from the top menu.

**Step 2** Name the symbol "button", choose Button from the Behavior list and click OK. In the Timeline area, you will now see the four states of a button: up, over, down, hit.

**Step 3** Select the Rectangle tool, pick a light red Fill Color and draw a rectangle in the work area.

**Step 4** Select the Text tool, pick a dark Fill Color and write "Click Me" over the rectangle.

**Step 5** Select the Arrow tool and place the text in the middle of the rectangle.

**Step 6** Add a key frame to the Over State in the Timeline. The Over State indicates what should happen when you mouse over the button.

**Step 7** Select the Rectangle, change the Fill color to a light green.

**Step 8** Choose Edit > Edit Movie to go back to the movie.

**Step 9** Choose Window > Library to locate the button. Drag the button into the work area.

**Step 10** Right click on the image. Choose Actions from the pop-up menu.

**Step 11** In the Object Actions box click on the + sign. Choose Basic Actions > Get URL.

**Step 12** Enter a full URL in the URL field (like <http://www.TOPS-int.com>).

**Step 13** Choose target in the Window field. Close the Object Actions box.

**Step 14** Choose Control > Test Movie from the top menu to test your Flash movie.

## Flash Animation

### Steps for built Flash Animation

**Step 1** Insert a text in the upper left corner of the Stage area. Do this by selecting the text tool from the left toolbar. Write some text in the "text area".

**Step 2** Select the arrow tool from the left toolbar. Click on the text once to select it.

**Step 3** Convert the text to a symbol. From the top menu choose Insert > Convert to Symbol. Name the symbol "text", choose graphic from the Behavior list and select OK.

**Step 4** Go to Frame 30 in the Timeline. Do this by clicking the gray field below 30. Then right click in this field. Choose Insert Keyframe. Keyframes appear as circles in a frame.

**Step 5** Click on the Timeline any place between Frame 1 and Frame 30. Then right click and choose Create Motion Tween.

**Step 6** Go back to Frame 30 in the Timeline. Move the text to the lower right corner.

### What is Flash ?

- Adobe's Flash format is among the most widely-used for delivering dynamic, rich-media content via the web.
- Flash is commonly used for delivering interactive, multimedia-based materials through a web browser.
- Each Flash document consists of 3 basic items:
- Timeline – a record of every frame, layer, and scene in your movie.
- Edit bar – displays text and menus for choosing symbols and scenes to work on.
- Stage – the actual area in which your movie displays.

### Why learn Flash & programming?(as an Interaction Designer)

Flash is the leading tool/technique for creating multimedia, applications, presentations, games on Internet – a powerful platform for developing systems, prototypes, desktop applications and mobile applications

Examples of general usage for a designer:

- Presentations / online portfolio
- Video applications for design-projects
- Concept design (with basic interactivity/navigation)
- Interaction interfaces
- Prototype development (usability tests)
- Working in project-teams (communicate with/understand programmers)
- Physical computing: Sensors (Phidgets), Webcam, Wii-remote, Multi-touch etc.
- and more....

### Program versions

- **Program versions**
  - **Flash CS3 (March 27, 2007)**
    - Brand New Interface, New and Improved Flash Video Importer
    - Improved Skinning of Components, Exporting Motions and Animations, Import Photoshop Files

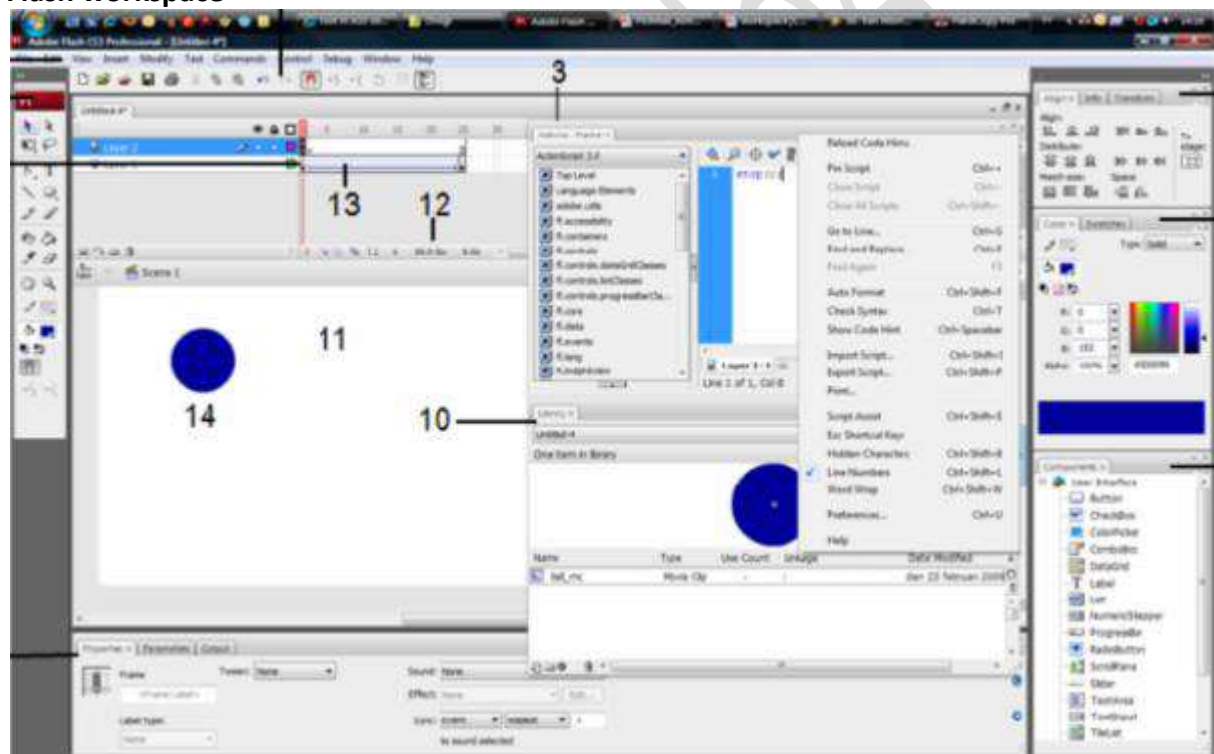
# TOPS Technologies

- **Flash CS4 (October 15, 2008)**
  - nObject-based animation, Bones tool (Inverse kinematics), 3D transformation, Motion editor, Motion presets, Metadata (XMP) support, Authoring for Adobe AIR, XFL support, Adobe Kulerpanel, H.264 support
- **Flash CS5 (April, 2010)**
  - iPhone support, Code Snippets, Improved Text-framework & Motion Editor, Developed integration: InDesign and PSP

## ActionScript versions

- ActionScript 2.0 (Flash 7 MX 2004)
  - Objectoriented, classes, objects, inheritance, components etc
- ActionScript 3.0 (2006)
  - Faster, cleaner, more powerful, easier to debug, more feature rich, strict and secure
  - Object-oriented structure: Built by classes, libraries, objects, functions, and properties
  - Expands the Flash-based technology

## Flash workspace

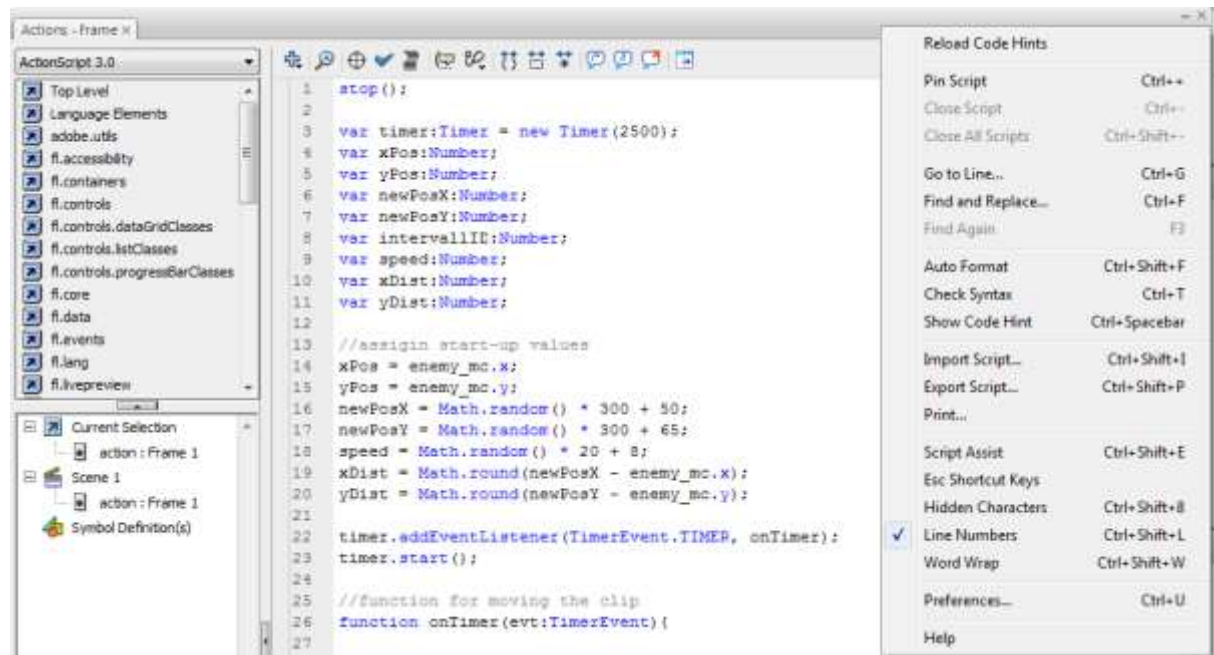


- |                                 |                         |                         |
|---------------------------------|-------------------------|-------------------------|
| 1. Menu                         | 6. Main toolbar         | 11. Library             |
| 2. Tool panel                   | 7. Align/Info/Transform | 12. Main stage          |
| 3. ActionScript panel           | 8. Color/swatches       | 13. Document Properties |
| 4. Timeline (main)              | 9. Components           | 14. Frames/tween        |
| 5. Properties/parameters/output | 10. Library             | 15. Object on stage     |

## ActionScript Panel

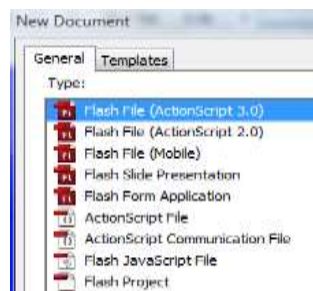






# TOPS Technologies



1. Packages/Script Library
2. Script/Symbol(s)/quick navigation
3. Topmenu
4. Code/Composing section
5. Script Assist
6. Right menu
7. Find and Replace
8. Help menu1

## File-types & formats



	FLA: The Project/work File (export will generate the .swf file)
	SWF: The exported program file (embedded in HTML)
	AS: ActionScript Class File (Built-in or user defined class files)
	FLV: Flash Video File

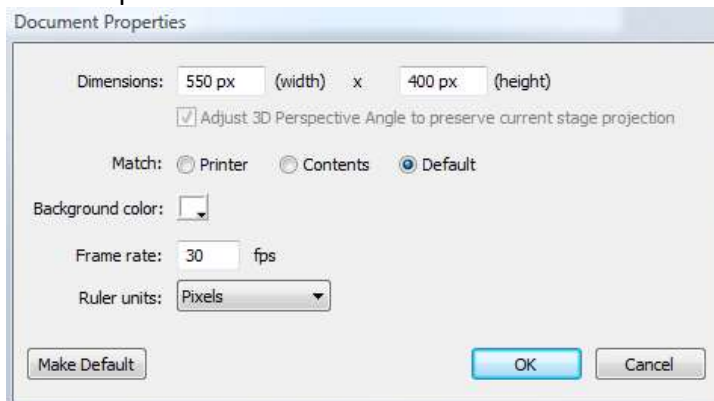
## Document Properties & Frame-rate



# TOPS Technologies

- Dimensions and frame-rate of the movie can be set in the Document Properties panel
  - Framerate (fps) Frames per second (30 fps movie: 30 times a second)
  - Recommendation 25-31 fps (12 fps is often too slow for tween animations, resulting in "jerky" motion)
  - SWFs published to the same fps, will in general run slower on the Mac Flash Player

Most of today's computer processors cannot keep up with a frame-rate higher than 31 fps



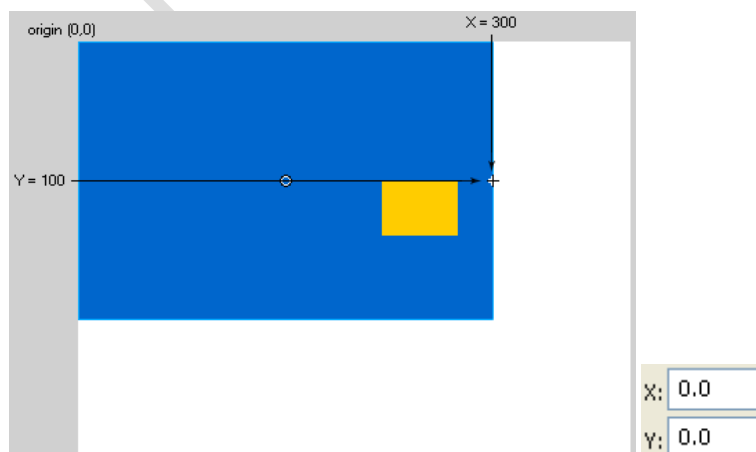
## Naming elements

- It's recommended to have an intuitive *naming structure for your elements (movie-clips, buttons, frames, layers, components etc)*
  - Use unique names
  - Keep names as short as possible while retaining clarity
  - Start with a lowercase letter
  - Use mixed case for concatenated words
  - Don't use the same element name with different cases
    - *Tip! Practical to organize every project the same way*

## MovieClips (1/2)

- MovieClips are the key-element for Flash-based animations
  - Every movie-clip has its own timeline
  - Movie-clips can be *nested* (movie-clip inside another mc)
  - A movieClip can be used as a button-object
  - Each MovieClip has a coordinate system in which the origin (0, 0) is located in the registration point. For the main timeline this is the top left corner

✓ *Tip! Use the suffix `_mc` for movieClips, ex. ball\_mc*



## MovieClips (2/2)

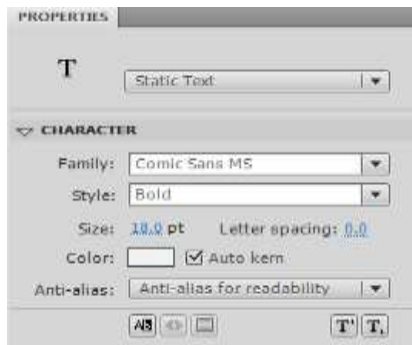
# TOPS Technologies

- Every created *MovieClip-instance* has the properties and methods of the *MovieClip* class - these properties and methods can be accessed and manipulated by code
- A Movie-clip has also *Display Properties* that we can access and manipulate (manually or by code)

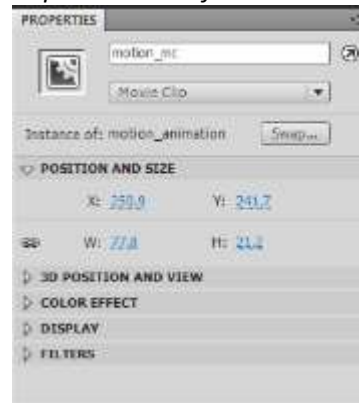
## Properties Panel

- The *Properties Panel* allows us to adjust most objects in Flash
- It is context sensitive, displaying options for whatever object is currently selected

Properties Panel for a selected text

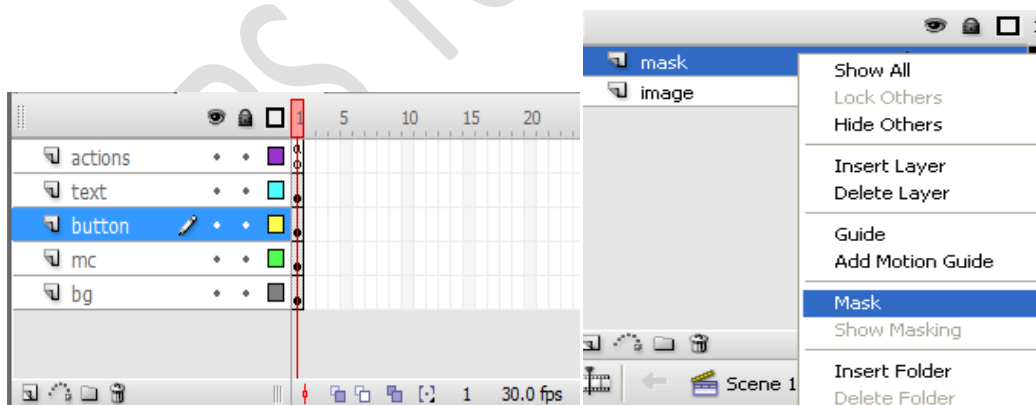


Properties Panel for a selected movieClip



## Layers & Layer Masks

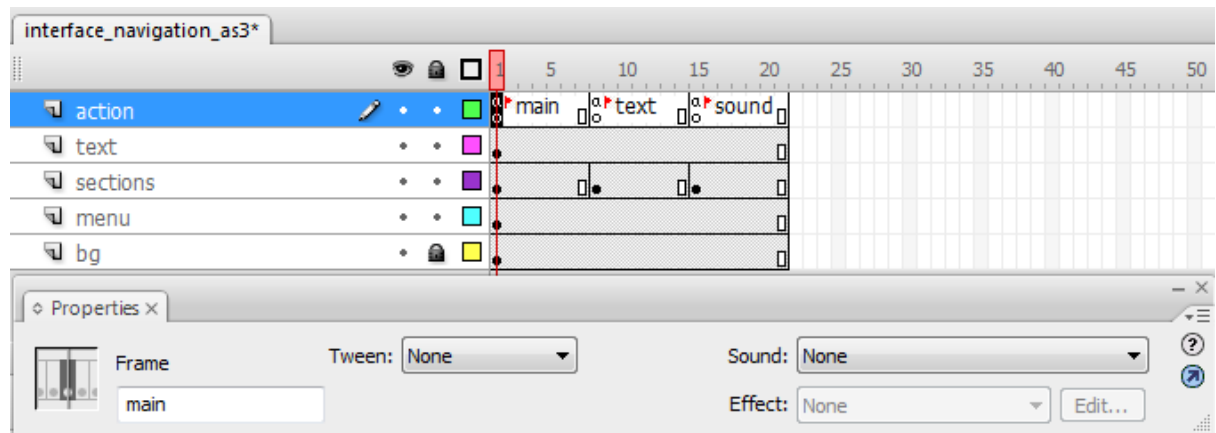
- Separate ActionScript layer and separate layers for each element group (intuitive naming)
- Layers can be locked
- Folders can be used for arranging the layers
- Layers can be used for Layer Masks (for masking of an area)



## Frames & navigation

- Frame-names (intuitive naming)
- Frames can be used for navigation/systems
- AS methods for jumping between Frames ex. on a Button-press: `gotoAndStop("main")`

# TOPS Technologies



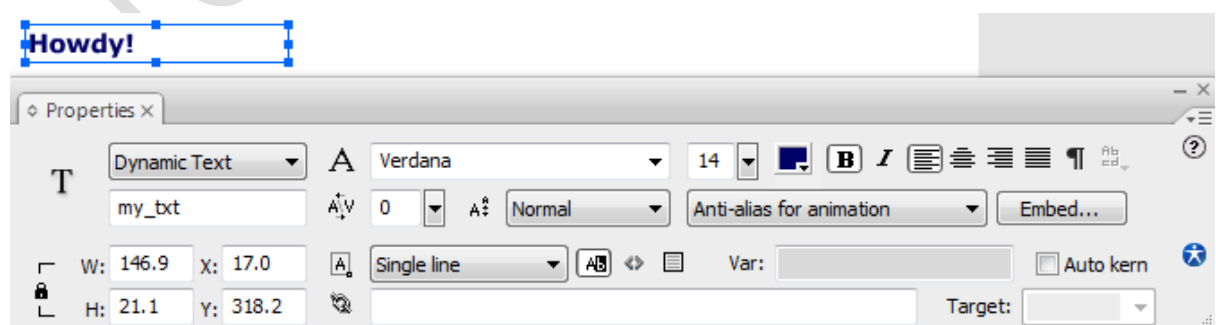
## Library & stage-elements

- Folders with *intuitive naming in library* (large projects)
- Same *naming structure for objects on stage* can be used



## Text-fields

- There are basically three different text-field types in Flash
  - Static Text: Animations, common use, standard fonts
  - Input Text: Input-text, forms, passwords, variables
  - Dynamic Text: Dynamic text, HTML-text, selectable/copy text, outputs, non-standard fonts (embed fonts), scrolling text, loading data by using :
    - Textfile
    - XML
    - Database (PHP, ASP)

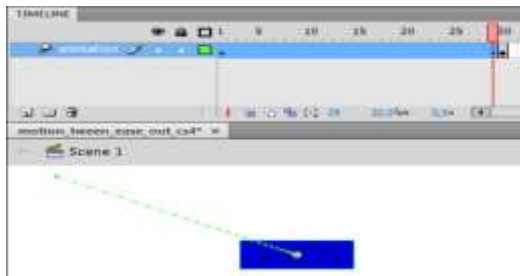


## Motion tween (CS4/CS5)(Timeline-animations)

- The new Flash CS4/CS5 *Motion Tween* is object-based and can be controlled by the *Motion Editor*

# TOPS Technologies

- These tweens can also be saved to the *Motion Presets-panel*



- ✓ Tip! See also the upcoming slides about the Motion Editor and Motion Presets

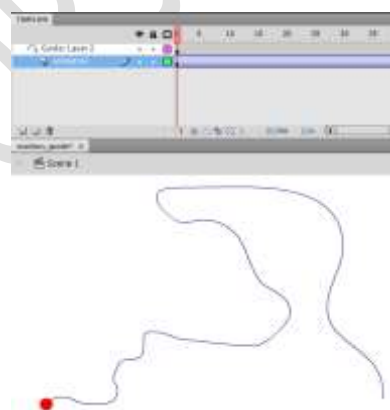
## Shape tween(Timeline-animations)

- A Shape Tween can be used for morphing a shape into another within a time-frame
- Flash cannot shape tweengroups, symbols, text blocks, or bitmap images. If you want to apply Shape Tweening to any of these objects, you must first break them down by clicking: MODIFY BREAK APART



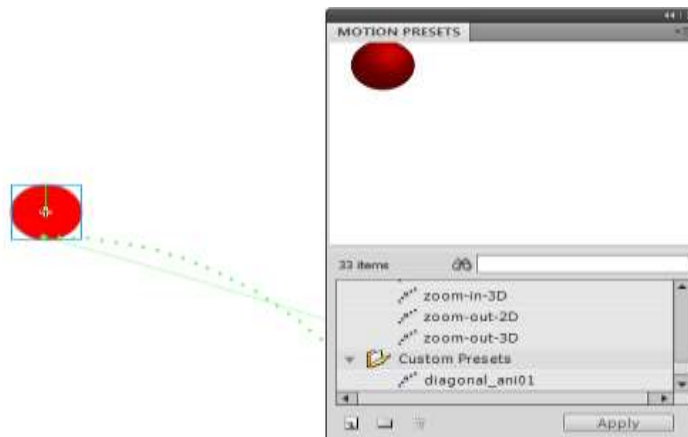
## Motion Guide(Timeline-animations)

- A Motion Guide can be used for animating a movieClip along a predefined path
- First, create a Guide Layer, draw your path –then make a Motion tween and snap your clip to the guide



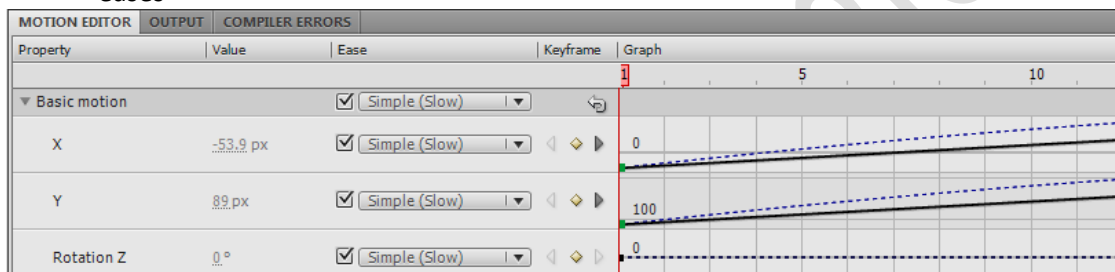
## Motion Presets

- The Flash CS4/CS5, Motion Presets is a panel with pre-defined animations
- These animation/tweens can be applied to your movieClip
- It's also possible to add your own animations to the Motion Presets list (Custom Presets)



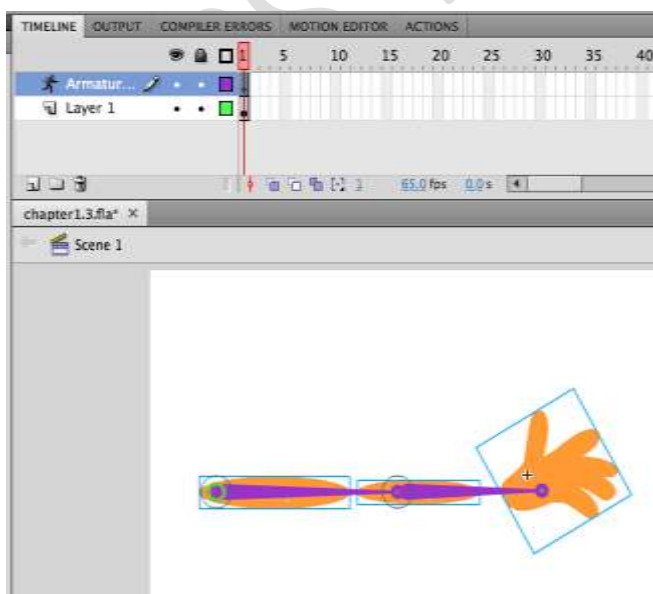
## Motion Editor

- With the Flash CS4/CS5 Motion Editor, you can manipulate your Motion tweens by editing *Motion curves*, and/or *adjusting specific parameters (without touching your timeline-tween)*
- For example parameters like: Basic motion, transformation, color effects, filters, eases



## Bone Tool(Inverse Kinematics)

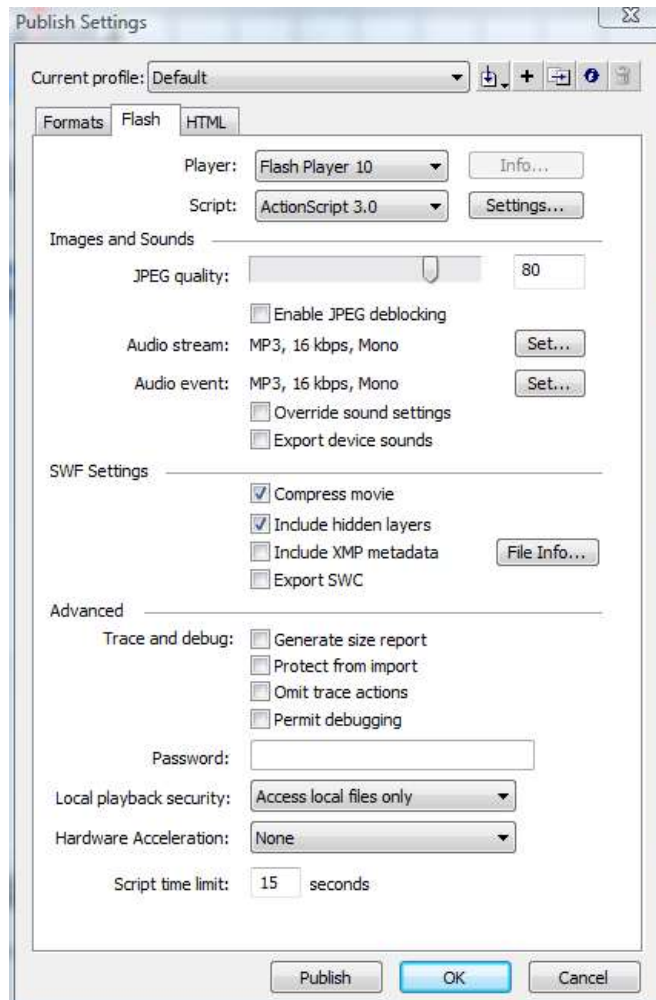
- The Flash CS4/CS5, Inverse kinematics (IK) is a way to animate parts of a whole object
- This can be done by using the Bone Tool from the Tools-panel



## Publishing(Formats, Flash & HTML)

# TOPS Technologies

- In the Publish Settings-dialogue, it's possible to select Flash Player version for the export or other publishing formats (ex. AIR, iPhone, Flash Lite), Action Script version etc.
- We can also select HTML-parameters for the embedded Flash-movie (swf-file)



## Publishing Flash files

JavaScript file for browser and Flash detection (.js)	→	AC_RunActiveContent.js	9 KB	JScript Script File
	→	Cash_reg-public_d-296.wav	66 KB	Wave Sound
Windows Executable (.exe)	→	CeoAssistant.exe	2,420 KB	Application
Flash (.fla)	→	CeoAssistant.fla	128 KB	Flash Document
HTML document to view movie in browser	→	CeoAssistant.html	3 KB	HTML Document
Flash Player Movie (.swf)	→	CeoAssistant.swf	30 KB	Flash Movie

## Interview Question

1. What is a Web Server?
2. What is the difference between a Web Browser and a Web Server?
3. What is a domain?
4. What is FTP? How is it used?
5. What is an IP?
6. What is TCP/IP?
7. What is a router
8. What is an ISP?
9. What is an SMTP server?
10. What is a web design and basic requirement?
11. Explain problem specification with browser comparability.
12. Which resolution is better for the perfect design?
13. What is w3c?
14. Explain what is the use of editor and their types?
15. Explain graphics base editor?
16. Is HTML case sensitive?
17. What is the difference between HTML and XHTML?
18. How many HTML versions are available in market?
19. What is DOCTYPE?
20. Explain HTML structure?
21. What is a caption?
22. What is HTML? How HTML tags are formed? Explain HTML comments.
23. What is the use of Anchor tag? Explain two ways to specify URL in href attribute?
24. Why images are used on web pages?
25. Which html tags are used to create table structure? Explain in brief at least five.
26. What is the difference between attributes cellpadding and cellspacing in the table tag?
27. Explain colspan and rowspan attributes with example.
28. What is use of from tag? Explain action and method attribute.
29. What is the difference between HTML and HTML5?
30. Explain Video tag in HTML5?
31. What is a tag? How can we use MARQUEE in HTML?
32. How do we create a link? What are the three types of form tags in HTML?
33. How do we specify page breaks in HTML?
34. How do we use an image instead of the standard submit button?
35. What is Semantic HTML? What are the reasons for validating a HTML?
36. What is BODY in HTML document?
37. What is SPAN in HTML?
38. What are differences between DIV and SPAN? What are the differences between cell spacing and cell padding?
39. How do we make a picture as a background on my web pages?
40. What is a CSS?
41. Explain types of CSS with example.
42. Difference between internal and external CSS?
43. Is CSS case sensitive or not?
44. What are style sheet properties?
45. Inline, embedded and external style sheet.
46. Explain Class?
47. What is grouping?
48. Explain ID and Class?
49. What is the use of selectors?
50. Explain HTML and CSS Selector?



51. Difference between DIV and SPAN selector?
52. What is inheritance in CSS?
53. Difference between inline and block?
54. What are the various text formatting tag in HTML?
55. List of various font attributes used in style sheet.
56. What is generic element used in CSS?
57. What is the difference between margin and padding?
58. How you define at-rule in CSS?
59. How you define first line pseudo element in CSS?
60. What is pseudo class?
61. How you define ruleset in CSS?
62. How you define floating element in CSS?
63. How to create CSS border?
64. Describe CSS1?
65. Compare CSS2 and CSS3?
66. What is the use of floating attributes?
67. What is the difference between Text-decoration and Text-indent?
68. Explain Alpha and Opacity attributes.
69. What is the use of content attribute?
70. What is alternate style?How to link?
71. Which browser support CSS?
72. How to customize your textbox?
73. How do I combine multiple sheets into one?
74. What is a Web Server?
75. What is the difference between a Web Browser and a Web Server?
76. What is a domain?
77. What is the difference between Graphic designing and Web Designing?
78. What web browser do you use?
79. What web standards and guidelines do you follow for designing web sites?
80. How do you make your website consistent among all the browsers?
81. What is the difference between DIV and SPAN tags?
82. How do you handle transparency in web-page elements
83. What are RGB/CMYK/HSI color models?
84. What is SEO?
85. Can you explain the difference between server-side scripting and client-side scripting.
86. In Dreamweaver what is used to apply same layout to my pages.
87. Which HTML tag is used to define an internal style sheet?
88. Whats is the difference between cell spacing and cell padding?
89. If a page has to be loaded over all frames in window, what should be the value of TARGET attributes?
90. How you define at-rule in CSS?
91. Which browser support CSS? How to customize your textbox?
92. How to unlock background?
93. If you are given with an image which is blurred, which tools you would use in order to sharpen it up?
94. How can you reduce noise of an image?
95. What do you understand by red-eye removal?
96. What is PSB & how it is different from PSD?
97. What is a Photoshop plug-in? Name few of them.
98. What are four main components of Photoshop workspace?
99. How many menus will you find on the menu bar? List all of them.
100. What is the shortcut to zoom back to 100%?
101. What is default color for foreground & background in Photoshop?

102. How can you make an image transparent?
103. Which tool you would prefer to use when you want to crop a human portrait from an image?
104. How one can resize an image in Photoshop?
105. How will you pick color from an image?
106. What is a smart object?
107. How can you create artistic borders for an image?
108. What is the quickest way to achieve artistic borders?
109. Explain about stress analysis in Adobe Photoshop?
110. How can we align layers of an image relative to each other?
111. What all operations you can perform over a layer?
112. What is a lasso tool? For what purposes it can be used?
113. Name few cropping techniques that could be performed with Photoshop.
114. How can you adjust the zoom level of the canvas?
115. How can one achieve dodging, burning & smudging effects in Photoshop?
116. What is a clone tools?
117. How can you draw fuzzy lines in Photoshop? Which tool will you prefer to use?