WEB SITE DESIGN - HTML (2022)



LibreTexts™

Web Site Design - HTML

This text is disseminated via the Open Education Resource (OER) LibreTexts Project (https://LibreTexts.org) and like the thousands of other texts available within this powerful platform, it is freely available for reading, printing, and "consuming."

The LibreTexts mission is to bring together students, faculty, and scholars in a collaborative effort to provide an accessible, and comprehensive platform that empowers our community to develop, curate, adapt, and adopt openly licensed resources and technologies; through these efforts we can reduce the financial burden born from traditional educational resource costs, ensuring education is more accessible for students and communities worldwide.

Most, but not all, pages in the library have licenses that may allow individuals to make changes, save, and print this book. Carefully consult the applicable license(s) before pursuing such effects. Instructors can adopt existing LibreTexts texts or Remix them to quickly build course-specific resources to meet the needs of their students. Unlike traditional textbooks, LibreTexts' web based origins allow powerful integration of advanced features and new technologies to support learning.



LibreTexts is the adaptable, user-friendly non-profit open education resource platform that educators trust for creating, customizing, and sharing accessible, interactive textbooks, adaptive homework, and ancillary materials. We collaborate with individuals and organizations to champion open education initiatives, support institutional publishing programs, drive curriculum development projects, and more.

The LibreTexts libraries are Powered by NICE CXone Expert and was supported by the Department of Education Open Textbook Pilot Project, the California Education Learning Lab, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation nor the US Department of Education.

Have questions or comments? For information about adoptions or adaptions contact info@LibreTexts.org or visit our main website at https://LibreTexts.org.

This text was compiled on 01/15/2026

TABLE OF CONTENTS

Licensing

1: Creating Web Pages

- 1.1: Internet History and Usage
- 1.2: Serving Web Pages
- 1.3: HTML Tags and CSS Styles
- 1.4: HTML Tags and CSS Styles
- 1.5: Creating Your First Web Page

2: Web Site Design

- 2.1: Goal identification
- 2.2: Design Priniciples
- 2.3: Choosing Colors
- 2.4: Typography
- 2.5: Accessibility
- 2.6: Web Site Development
- 2.7: Mobile Web Development

3: Basic Document Layout

- 3.1: Structuring Page Content
- 3.2: Document Structure Tags
- 3.3: Paragraphs
- 3.4: Line Breaks
- 3.5: Headings
- 3.6: Horizontal Rules
- 3.7: List Structures
- 3.8: Linking Pages
- 3.9: Displaying Pictures

4: Basic Document Styling

- 4.1: Applying Styles
- 4.2: Margin Styles
- 4.3: Alignment Styles
- 4.4: Width and Height Styles
- 4.5: Basic Color Styles
- 4.6: Basic Color Styles
- 4.7: ID and Class Selectors

5: Text Formatting

- 5.1: Font Settings
- 5.2: Text Properties
- 5.3: Font Style Tags
- 5.4: Text and Background Colors
- 5.5: Special Characters

6: Graphic Images

- 6.1: Graphic Formats
- 6.2: Displaying Images
- 6.3: Background Images
- 6.4: Image Transparency

7: Applying Special Styles

- 7.1: Styling Lists
- 7.2: Borders and Padding
- 7.3: Sizing Elements
- 7.4: Positioning Elements

8: Linking Pages

- 8.1: Text Links
- 8.2: Graphic Links
- 8.3: Image Maps
- 8.4: Redirecting Pages
- 8.5: Traversing Folders

9: Using Tables

- 9.1: Creating Tables
- 9.2: Table Borders
- 9.3: Table Captioning
- 9.4: Table Captioning
- 9.5: Table Size and Alignment
- 9.6: Aligning Text and Cells
- 9.7: Column and Row Spanning
- 9.8: Table Groupings

10: Playing Multimedia

- 10.1: Audio and Video Formats
- 10.2: Playing Audio
- 10.3: Playing Video
- 10.4: Third Party Content

11: Creating Forms

- 11.1: Form Controls
- 11.2: Creating Forms
- 11.3: Textbox Controls
- 11.4: Textarea Control
- 11.5: Radio Buttons
- 11.6: Checkboxes
- 11.7: Drop-Down Lists
- 11.8: Submit and Reset Buttons
- 11.9: Other Buttons
- 11.10: Grouping and Labels
- 11.11: Submitting Forms

- 11.12: Forms and CSS

12: HTML/CSS Appendix

- 12.1: New HTML5 Tags
- 12.2: HTML Reference
- 12.3: CSS Reference
- 12.4: HTML Special Characters
- 12.5: HTML Colors

Index

Glossary

Detailed Licensing

Licensing

*A detailed breakdown of this resource's licensing can be found in **Back Matter/Detailed Licensing**.*

CHAPTER OVERVIEW

1: Creating Web Pages

1.1: Internet History and Usage
1.2: Serving Web Pages
1.3: HTML Tags and CSS Styles
1.4: HTML Tags and CSS Styles
1.5: Creating Your First Web Page

1: Creating Web Pages is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

1.1: Internet History and Usage

1.1: Internet History and Usage is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

1.2: Serving Web Pages

1.2: Serving Web Pages is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

1.3: HTML Tags and CSS Styles

1.3: HTML Tags and CSS Styles is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

1.4: HTML Tags and CSS Styles

1.4: HTML Tags and CSS Styles is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

1.5: Creating Your First Web Page

1.5: Creating Your First Web Page is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

2: Web Site Design

2.1: Goal identification
2.2: Design Priniciples
2.3: Choosing Colors
2.4: Typography
2.5: Accessibility
2.6: Web Site Development
2.7: Mobile Web Development

2: Web Site Design is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

2.1: Goal identification

Site Structure

Whether you are developing Web pages for personal use or you become involved in commercial Web ventures the point arrives where you need to become conscious of Web site design. You will need to bring organization and logic to the proliferation of pages. Site design is not an exacting science; still, there are organizing principles and design strategies that can help you produce a more manageable site that assists your visitors in navigating its pages to find the information they desire or that you intend.

Web Site Goals

It should go without saying, but a Web site should have a purpose. There are few Web activities as frustrating as navigating a site that is a hodge-podge a pages with no apparent focus. You probably have visited sites that contain one very long Web page intermixed with content and links that seemingly have little relationship to one another; or you have encountered sites at the other extreme, with tiny content pages within a labyrinth of links that leave you isolated at its farthest reaches, staring at irrelevant content, and with no apparent way to get back to where you came from.

Both of these extremes show a lack of foresight and planning. The site designer probably had no notion of its intent other than to create a Web presence. Therefore, the site evolved in helter-skelter fashion, creating an unpleasant navigational journal for visitors.

A key factor in site design, then, is to have a clear idea of its purpose. That purpose can be modest or elaborate. In either case, you should be able to frame its purpose verbally or in writing. Even for a personal Web site it is helpful to produce a statement of intent. Consider, for instance, the following statements of purpose:

- To present a personal resume of education, experience, and work portfolio for reference by prospective employers.
- To create a family Web site to summarize current activities of family members.
- To create a Web site to document vacation travels.
- To maintain a geneology of the family.
- To manage classwork related to my Web development.

Each of these statements brings focus to the site while permitting it to grow within reasonable bounds. You avoid the risk of creating a single site that tries to incorporate all aspects of your personal life in a conglomeration of unrelated pages and links.

If you are, or become, a professional Web developer, then the statement of purpose is usually easier to define. It often derives from an organization's vision statement or mission statement, the guiding purpose for the organization. In a commercial enterprise, for example, common Web site purposes include

- To market and sell the primary product line of the company.
- To provide technical support services to customers.
- To increase public awareness of the company and its products.
- To lower business costs through electronic transactions with suppliers and distributors.
- To enhance in-house communications of company directions and activities.

Like the personal goals described above, these objectives set the boundaries of the site. It becomes clearer whether or not particular content or links contribute to the purpose.

Web Site Content

Being able to define the purpose of a Web site also helps to determine the type of content to be presented and its organization. The process of structuring the content of a Web site is not unlike the process of designing a database. You can, in fact, think of a Web site as a database of information composed of pages of content arranged by their linked relationships. In database parlance, your task is to create "entity-relationship" models of the information to be presented; in Web parlance the task is to create "content-link" models of the information.

A good statement of purpose for your Web site will help you determine items of content appropriate to the site. In the same way that database designers identify the major entities -- the objects of informational interest -- to be included in a database, you need to

identify the major objects of informational interest in your Web site. Taking the example above of a personal site to manage classwork for a course you are taking, the major items of interest might be

- textbook reading summaries
- lecture notes
- assignment solutions
- projects
- grades
- calendar

Although you may think of other informational entities related to a class, this list presents a good start in breaking down the purpose of the site into those content areas related to its purpose -- and *only* those areas related to its purpose. At this point it is not necessary to know all the informational details about your site. It is important only to identify the major topics around which the site will be developed. Analogous to writing a report, you need to outline your topic to determine the main headings that will be expanded later into paragraphs, sentences, and words.

Web Site Structure

The topics, or informational entities, that comprise a Web site are logically related in some fashion. At least they should be related since they are derived from a common purpose. Further, these topics eventually need to be physically packaged as Web pages with links between them; plus, the physical arrangement of pages and links needs to parallel their logical relationships.

There are three primary ways in which to organize topics and their container Web pages -- **hierarchical**, **linear**, and **mixed**. Any of these structures, or their combinations, are devised for your content to become Web accessible.

Hierarchical Structure

A hierarchical, or **tree**, structure of topics is the most common arrangement of Web pages. A tree structure arranges topics from general to specific -- or from summary to detail -- from the top to the bottom of the hierarchy.

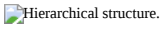


Figure **2.1.1**: Hierarchical arrangement of Web pages.

The top page in the hierarch is usually the "home" page providing an overview of the site, with links to the main topical subdivisions. Each of these subdivisions, in turn, link to pages containing more detail about the topic. Navigation, then, tends to be down the hierarchy with returns links up the hierarchy that retrace the same downward path. There can be occasional crosslinks between topics at the same level of the tree, but not usually. This structure brings logical organization to a site and permits visitors to find information in a rational top-down fashion.

Linear Structure

A linear, or sequential, structure of Web pages is most appropriate where the topical information needs to be accessed in a particular order.

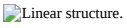


Figure **2.1.2**: Linear arrangement of Web pages.

In presenting a story, a time-line of events, or the steps in a process, a linear sequence of Web pages probably works best. The links are from one page to the next with return links retracing the sequence. If necessary, additional links can be made between the opening page and each of the detail pages for jumping into and out of the sequence. Still, the basic structure is linear.

Mixed Structure

Most likely, a Web site is not exclusively hierarchical or linear, although one of the structures dominates. It is common to find an overall hierarchical structure with linear sequences of pages at the detail level.

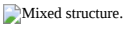


Figure **2.1.3**: Mixed arrangement of Web pages.

Random Structure

A Web site that offers no clear path through the site is said to have a random structure. A random structure is not as common as the hierarchical, linear, or mixed structures. This type of site structure is not recommended for commerical Web sites.

The above organizational structures are not meant to imply that Web content must be shoe-horned into these arrangements. They are presented as conceptual schemes only; you will need to determine the best arrangement of pages for presenting your content in an organized fashion. By keeping them in mind, however, you are forced to think about how the information itself is structured and to devise compatible linkage structures for offering it in the clearest and most logically consistent manner.

Web Site Navigation

The following are considered best practices when creating Web site navigation:

- Easy to follow navigation menus
- Short Pages
- Table of Contents
- Site Maps

2.1: Goal identification is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

2.2: Design Priniciples

As you develop the structural layout of your Web site you also begin packaging its content as individual Web pages with specified links between them. A common way of modeling the content and links between Web pages is called **storyboarding**. This technique visually maps Web content to Web pages and describes the links that implement the logical relationships between topics. Storyboarding forces you to flesh out your topics and refine their relationships.

Content Packaging

Consider the following illustration of a portion of the structure of this HTML tutorial. This diagram represents the layout of the first section of tutorials with topics identified along with menu links to the associated pages.



Figure 2.2.1: Structure of portion of HTML tutorial.

Since the pages appear inside frames there is no need for return links. The menu is always accessible for direct linkage to any page. Although additional linear links between pages would be an option, these extra links would be redundant.

Design Drawings

You shouldn't pay a lot of attention to the format and symbols used in the high-level storyboard in Figure 12-4. There are no formal graphical standards for storyboarding. You might, in fact, draw your layout in crayon on the back of a grocery bag and accomplish your purpose just as well. Sometimes professional developers become overly committed to using formal graphical tools to make design drawings and lose sight of their purpose, which is to construct mental models, or abstractions, of designs in order to manipulate them intellectually. The purpose is not to produce pretty drawings. Pick the symbols and techniques that work best for you. Of course, if you work professionally as a site designer, there may be in-house architectural standards for design drawings. However, you can always reproduce your conceptual drawings as design documentations when that time comes.

With this structural layout in hand the next step is to elaborate the content appearing on each page. One way to do this is with a text outline of topics, subtopics, and "talking points," much like you would do for a written report. For instance, the first page of this tutorial can be outlined as follows.

Internet History and Usage

1. Introduction
2. ARPANET
    1. Origins of ARPA
    2. Purposes of ARPANET
    3. Distributed packet switching (Leonard Kleinrock)
    4. ARPA network growth
    5. TCP/IP
3. NSFNET
    1. Origins and purpose
    2. Defunding implications
    3. Internet definition
4. WWW
    1. Hypertext concept (Ted Nelson)
    2. World Wide Web beginnings (Tim Berners-Lee)
    3. Browser development
5. Technical Convergence
    1. Protocols
    2. Formatting language
6. Internet Usage Statistics
    1. Country ranking by usage
    2. Regional ranking
7. Internet Technologies
    1. Connection speeds
    2. Browser version
    3. Screen resolution
    4. Color depth

Figure **2.2.2**: Outline of first page of HTML tutorial.

Since a Web page can be as much a visual experience as a textual one, some Web page authors utilize graphical storyboarding in addition to text outlines to indicate page content. This technique roughs out the main formatting characteristics of the page as well as organizes the content. The following illustration is one way to graphically characterize the first portion of the opening page of this tutorial.

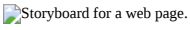


Figure **2.2.3**: Storyboard for a Web page.

Although it is shown here as a graphic image, the storyboard for a page can be a hand drawing or an HTML layout of the page. You might wish to use WYSIWYG editors such as FrontPage or Dreamweaver to produce "quick and dirty" layouts. The purpose is not to generate specific content but to indicate some beginning page design thoughts. You might wish to note the main headings that appear on the page, any graphics or tables that need to be included, a rough estimate of the amount of space to be devoted to different topics, and any links to internal pages or external sites.

You should, though, at this point begin to firm up the basic, common design that will be used across all pages. In order to visually and operationally tie your pages together with a common "look and feel" it is appropriate to settle on design considerations such as page margins and paragraph styles, color schemes, heading sizes and colors, text fonts and sizes, link formats, and other general design characteristics that will be common to your pages.

One of the advantages of using HTML editors at this point is that you can experiment with different designs. As you settle these overall design issues you also can begin constructing the Cascading Style Sheets that will be applied to your pages. Use a linked style sheet for overall design and formatting with embedded or in-line style sheets to alter or enhance styles for particular pages.

Design Issues

The following are other design issues that you should keep in mind when storyboarding and developing a Web site:

- Repetition
- Contrast
- Proximity
- Alignment
- Load Time
- Horizontal Scrolling
- White Space
- Screen Resolution

Page Sizes

One major design issue relates to the appropriate size for a Web page. The width of the page needs to take into account the width of the browser window which, in turn, is determined by the user's screen resolution. Standard horizontal and vertical screen resolutions are 800x600, 1024x768, and 1280x1024 pixels.

Currently, 1366x768 is the most common screen resolution. Unless you have reason to do otherwise -- knowing, for example, that your population of visitors uses other resolutions -- then it makes sense to target your pages to 1366x768 screens. In short it is good practice to optimize Web pages for 1366x768, but use a liquid layout that stretches well for any resolution, from 800x600 to 1280x1024.

The browser window, though, contains borders, menu bars, status bars, scroll bars, and other trappings that reduce the amount of space within which a Web page is displayed. The following table gives the dimensions of the effective sizes of browser display windows under different screen resolutions.

Screen Resolution	Browser Window Size
800x600	778x430
1024x768	1004x598
1200x1024	1259x553 or more

Figure **2.2.4**: Monitor screen resolutions and browser window sizes.

Horizontal Sizing

The browser word-wraps free-flowing text so that it always stays within page margins. Other page elements, however -- such as graphics or text containers that are of fixed sizes -- may not fit within these boundaries. The problem that is caused is a *horizontal scroll bar* on the browser window, something that should be avoided on all Web pages. There are few things more irritating, or unproductive for that matter, than having to scroll back and forth to view Web page content.

The following browser window illustrates the problem and the solution. The first shaded division has a width setting of 760 pixels, proper for a screen resolution of 800 x 600. However, when displayed in a browser on a 640 x 480 screen, the division extends beyond the page margin and introduces a horizontal scroll bar to view its entire width.

Page Widths

Here is a text paragraph. By default the browser word wraps text to keep it within the boundaries of the browser window. Other page elements, however, can extend beyond the margins of the page.

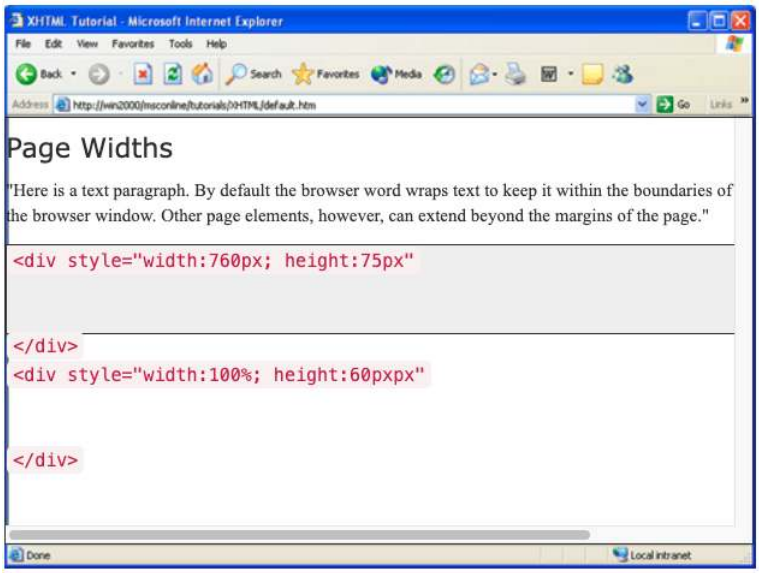


Figure **2.2.5**: Managing Page Widths

The best solution for sizing text blocks is to use a percentage measure as is done in the second division. The width shown here is 100%. Therefore, the division will extend only to the width of the page margins irrespective of screen resolution.

When displaying graphics, horizontal sizing is a little more problematic since adjusting widths may distort the picture or cause pixelization when forced to widths larger than its original size. It is always best to size the picture itself rather than adjusting it with style settings. Pick a size that fits within a 600 x 400 window to ensure that it fits properly within any other browser window.

Vertical Sizing

Obviously, there's not a lot that can be done about the vertical sizes of Web pages. They expand to encompass the amount of content. Of course, you can always subdivide your content into separate, shorter pages with links between them. The trade-off becomes one of vertical scrolling versus link clicking, neither of which has particular priority for user friendliness.

Design Mythologies

You may encounter page design guidelines that recommend nonscrolling pages in all cases -- that all pages should fit within the browser window without even vertical scrolling. Not only does this guideline make it difficult for the designer to find physical page breaks where none my logically exist, it introduces excessive clicking and linking that can disrupt the reader's flow of thought. It is best never to let physical containers dictate the flow of their content. Use reasoned judgement to determine how best to package your information.

Page Organization

Recognize that the reader's first impression is produced by what is seen at the top of the page when it first loads. In newspaper parlance this is the page area "above the fold" which provides the first clues about page content. This is the most important real estate on the page when it comes to serving the browsing needs of visitors.

Visitors arrive at a Web page looking for information. The initial page load should either provide that information or quickly direct visitors to it. A typical design for the top of each page includes,

- A common identity -- a banner, logo, and/or headings -- to tie the page visually to other pages of the site.
- The most important information on the page or a summary, possibly with links down-page to sections of detail information.
- A common menu of links to other major sections of the site.
- A menu of links pertinent to the content of the page.

Readers should know at first glance whether this page contains the information they are looking for. If so, they can scroll down to find it; if not, they can immediately navigate to other pages.

Page Layout Techniques

Fixed or Absolute Design

A fixed design utilizes CSS to configure a fixed-width block level element or page wrapper to format the page. A fixed design page is aligned with the left margin and the width of the page is set using a fixed unit such as pixels. The right hand side of the browser window will often contain empty or white space. When using a fixed page layout, it is possible to configure a page to look best at a certain screen resolution such as 1024 by 768. A disadvantage of the fixed design is that a page is designed to be optimized for a specific screen resolution. Viewing the page at other resolutions will include large amounts of white space on the right hand side of the browser window and horizontal scrolling will occur at lower screen resolutions.

A fixed layout is often preferred if you want tighter control over the appearance of your pages. Pages that contain lots of graphic images or other elements that are absolutely positioned, tend to benefit from a fixed page layout. If a fixed layout is used, the fixed dimensions should be set at reasonable levels to accommodate the most common screen resolution settings used by page visitors. This will help to prevent horizontal scrolling and a layout bordered by a sea of white space.

Fixed Width Page optimized for 800 X 600 screen resolution. Height and Width of the page are fixed and will not change to accomodate other screen resolutions. When viewed at higher screen resolutions, large amounts of white space appear on the right side of the browser window.

Width: 778 px

Height: 430 px

A wrapper division is often used to set the page dimensions

```
#wrapper {width:778px;height:430px}

<div id="wrapper">

Page content goes here

</div>
```

Figure 2.2.6: Fixed or Absolute Design

Liquid or Relative Design

A liquid design results in a web page layout that takes up 100% (or other percentage values) of the browser window regardless of the user's screen resolution. Unlike the fixed design, a liquid design uses percentages or a relative unit of measure to set the page width. With the liquid design there is no blank margin on the left or right side. A site that uses a liquid layout will expand and contract to fit the size of the browser window. A Web page with a width of 80% will always span 80% of the browser window regardless of the screen resolution.

Liquid Width Page. The Width of the page is set to 100% and will expand and contract to accomodate all screen resolutions. The Height of the container will expand to accomodate its contents. Otherwise, an absolute Height value can be set.

Width: 100%

A wrapper division is often used to set the page dimensions

```
#wrapper {width:100%;height:430px}

<div id="wrapper"

Page content goes here

</div>
```

Figure **2.2.7**: Liquid or Relative Design

Jello Design

A jello design involves centering all content on the page. This design can be configured using a fixed width or liquid width. Regardless of the screen resolution, the content is always centered on the page. This is accomplished by using the "margin-left" and "margin-right" CSS properties.

Jello Design. The Width of the page is set to 75%. The container or wrapper is centered using the CSS margin-left and margin-right properties. Setting these properties to "auto" will center the wrapper in the browser window. The Height of the container will expand to accomodate its contents. Otherwise, an absolute Height value can be set.

Width: 75%

A wrapper division is often used to set the page dimensions

```
#wrapper {width:75%;height:430px;margin-left:auto;margin-right:auto}

<div id="wrapper">

Page content goes here

</div>
```

Figure **2.2.8**: Jello Design

When designing a page, there is no simple answer to which layout technique will work best. Designing for different resolutions can sometimes seem like you are standing between a rock and a hard place. The use of one method over the other will not solve all your problems. Regardless of which technique is used, it is important to always test your pages under a reasonably high resolution and a

low resolution to see how it appears. In most cases, you will have to live with some imperfections when your site is viewed under extreme settings.

2.2: Design Priniciples is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

2.3: Choosing Colors

Page Colors

Colors draw attention to a Web page. They focus interest on page elements and help to highlight important information. Colors can also be distractive. Improperly used they can draw attention to themselves and away from central page content. One of the most important choices you make as a page designer, then, is a color scheme to complement or enhance information appearing on a Web page.

Color Wheel

A **color wheel** is a chart that shows how colors are related to make it easier to choose harmonious color combinations that attract attention to page content. The color wheel is divided into three categories of colors.



Figure **2.3.1**: Color wheel.

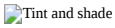


Figure **2.3.2**: Tints and shades.

The **primary colors** are **red**, **yellow**, and **blue**. These are the foundation colors from which all other colors derive. They are called primary colors because no other colors can be mixed to create these colors. They are evenly spaced around the color wheel.

Secondary colors are produced by combining any two of the primary colors. The three secondary colors are **orange** (red + yellow), **green** (yellow + blue), and **violet** (red + blue).

Tertiary colors are produced by mixing a primary color and an adjacent secondary color. The six tertiary colors are **red-orange**, **red-violet**, **yellow-green**, **yellow-orange**, **blue-green**, and **blue-violet**.

The purest value of a color is its **hue**. A color's **tint** is a lighter value of the hue made by adding white; a color's **shade** is a darker value of the hue made by adding black.

Picking Colors

When choosing colors for a Web site it is best to select only a few colors. You do not want to overpower and detract from the information contained on the page; you want to complement or emphasize it. Generally, you will pick a dominant color along with other colors that are analogous to it or that contrast with it. There are standard color-matching schemes that can be followed in picking these colors.

Color Wheel Pro - a unique software program that allows you to see color theory in action. With Color Wheel Pro, you can create harmonious color schemes and preview them on real-world examples.

Other useful color scheme Web sites include

- Colors on the Web - Color Wizard
- Color Scheme Designer
- Color Jack - Color Formulas

Monochromatic

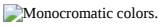


Figure **2.3.3**: Monocromatic colors.

A monochromatic color scheme uses a single hue with various tints and shades for contrast. Normally you will pick darker colors for text presentation and lighter colors for backgrounds. The various tints and shadings range from bold to subtle, and you need to choose the combination that works well for the message you are trying to get across and the mood you are trying to set.

Monochromatic Table Colors

background-color:#FFFFE0;

Section	Hue	Hex Value	RGB Value
Header	Red (dark)	#CC6666	204, 102, 102
Column	Red (light)	#FAC8C8	250, 200, 200
Background	Red (light)	#F6E1E1	246, 225, 225

Figure **2.3.4**: Examples of monocromatic colors.

Analogous

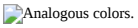


Figure **2.3.5**: Analogous colors.

An analogous color scheme uses adjacent hues and their tints and shadings. All of the colors share a common hue, for example, **red**-violet, **red**, and **red**-orange. The presentation and feeling is similar to a monochromatic scheme with a larger assortment of hues from which to choose.

Analogous Table Colors

Section	Hue	Hex Value	RGB Value
Header	Red-Violet (dark)	#C4028F	196, 2, 143
Column	Red-Orange (light)	#FFAD5B	255, 173, 91
Background	Red (light)	#FFCCCC	255, 204, 204

Figure **2.3.6**: Examples of analogous colors.

Complementary



Figure **2.3.7**: Complementary colors.

A complementary color scheme uses a hue on the opposite side of the color wheel from the dominant color. This color combination provides the greatest contrast and makes both colors more intense and brighter than when they are used alone.

Complementary Table Colors

Section	Hue	Hex Value	RGB Value
Header	Red (dark)	#C4028F	196, 2, 143
Column	Green (dark)	#02D0BF	2, 208, 191
Background	Green (light)	#D0F2E0	208, 242, 224

Figure **2.3.8**: Examples of complementary colors.

Split-Complementary

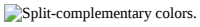


Figure **2.3.9**: Split-complementary colors.

A split-complementary color scheme uses a hue along with two colors that are adjacent to its complement. For example, if the dominant color is red, then split-complementary hues are yellow-green and blue-green.

Split-Complementary Table Colors

Section	Hue	Hex Value	RGB Value
Header	Red (light)	#FF6666	255, 102, 102
Column	Blue-Green (dark)	#25EADA	37, 234, 218
Background	Yellow-Green (light)	#E6FBC6	230, 251, 198

Figure **2.3.10**: Examples of split-complementary colors.

Triadic



Figure **2.3.11**: Triadic colors.

A triadic color scheme uses three hues that are evenly spaced from each other on the color wheel. For example, if the dominant color is red, then blue and yellow hues are the triadic complements. Like the complementary color scheme, this one also emphasize the colors through their contrasts.

Triadic Table Colors

Section	Hue	Hex Value	RGB Value
Header	Red (light)	#FF7575	255, 117, 117
Column	Blue (light)	#7676FB	118, 118, 153
Background	Yellow	#FFFF99	255, 255, 153

Figure **2.3.12**: Examples of triadic colors.

Achromatic

Although they do not appear in the color wheel, don't forget about black, white, and the range of gray tints that can be applied to highlight or emphasize page content. These achromatic tones can offer high contrast or subtle shading.

Achromatic Table Colors

Section	Hue	Hex Value	RGB Value
Header	50% Gray	#818181	255, 117, 117
Column	25% Gray	#C1C1C1	193, 193, 193
Background	5% Gray	#F6F6F6	246, 256, 256

Figure **2.3.13**: Examples of achromatic colors.

Selecting a Color Palette

A half-dozen or so colors is normally a good starting point for building a pallet of analogous or complementary colors for a page color scheme. In some cases you pick the colors based on preferences or existing standards.



Figure **2.3.14**: Selecting a triadic color scheme.

When working with photographs, a good starting point is to select colors based on those *appearing in the picture*. In the accompanying illustration, a palette of five colors, plus black and white, have been chosen from the picture. Various muted shades of red and blue -- part of the triadic color scheme -- produce the subtle contrast needed between text and background. The intent is to *recede* the text in order to bring the picture to attention. These colors were extracted from the photograph using the *Photoshop* eyedropper tool.

As a general rule, it is probably best to stay away from pure hues as is used in the second illustration. Pure colors tend to be garish, with high contrast that can confuse the eye as to the point of focus. In this case, the boldness of the text tends to draw attention away from the picture and towards the upper and lower portions of the illustration. If that is the look you are going for, then fine, although there are better ways to emphasize text without high contrast colors.



Figure **2.3.15**: Selecting pure hues.

With 16 million or so colors to choose from, it can be difficult picking the half-dozen you need for your pages. An effective way of choosing colors is with various types of color-wheel software that is available. Color Wheel Pro - a unique software program that allows you to see color theory in action. With Color Wheel Pro, you can create harmonious color schemes and preview them on real-world examples.

2.3: Choosing Colors is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

2.4: Typography

Usually, Web pages are meant to be read rather than viewed. This means that you need to devote thought to the text styling that can make your pages most legible and readable while fitting in with other design considerations. **Typography** is this relationship between letterforms on a page, playing a dual role in the visual design of the page and in enhancing its readability.

The term **font** has become the catch-all term to denote text characteristics employed on a page. More accurate, the term **typeface** refers to a consistent visual design for the symbols of the alphabet; a **font family** is a related set of typefaces, including their rendering in bold, italic, and other typographical off-shoots. The following illustration shows some common terminology used in describing text styling.

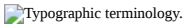


Figure 2.4.1: Typographic terminology.

Outline Fonts

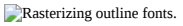


Figure 2.4.2: Rasterizing outline fonts.

Most digital fonts -- such as Microsoft *TrueType* and Adobe *PostScript* -- are stored as character outlines using mathematical descriptions of the characters. Use of these outline fonts means that only one outline per character is needed to produce all the sizes of that character; it can be scaled to different sizes and even skewed and rotated without losing its characteristic shape.

In order to be displayed on a computer screen or on printed paper, the outlines need to be "filled in," or **rasterized** as pixel points. Rasterizing software performs this function as shown in the accompanying illustration.

Font Families

It is important to remember that a Web page can only display those fonts that are installed on the user's computer Therefore, nonstandard fonts should be avoided. The table below contains common PC and MAC fonts.

Windows Fonts	MAC Fonts
Arial	Helvetica
Arial Black	**Arial Black**
Courier New	Courier New
Georgia	Georgia
Impact	**Impact**
Times New Roman	Times
Trebuchet MS	Helvetica
Tahoma	Geneva
Verdana	Verdana

Figure 2.4.3: Typical core fonts installed on desktop computers.

Of course, most users have other software installed on their systems which come with additional fonts to support those applications. Those fonts may or may not be accessible for browser display.

Fonts can be classified as **serif** and **sans-serif** (without-serif) styles. Serifs are the small "tick marks" at the end of character strokes. You can see this difference in the Times New Roman (serif) and Arial (sans-serif) fonts shown below:

Serif

Sans-Serif

Figure 2.4.4: Serif and non(sans)-serif type faces.

Generally one serif font for text and one sans serif font for headings (or vice-versa) are a good combination. A page should contain no more than two different typefaces or four different type variations such as type size and bold or italic style.

Font Size

Readability is affected by font sizes. Print type is traditionally measured in **points**, which is 1/72 of an inch (on paper). However, computer screens are based on pixels, rather than points, so rendering software must translate point sizes to pixels units taking into account screen resolution. You can see from the two lines below how 12-point type size equates to approximately 16-pixel type size on the screen.

12pt Times New Roman (print)
16px Times New Roman (screen)

Figure **2.4.5**: Point and pixel type sizes.

While print type size is based on absolute units of measure such as points and pixels,relative units such as **percentages** and **em (preferred)**, should be used in web documents since absolute font sizes render inconsistently across platforms and can't be resized by the browser.

1 em unit is equivalent to **12 points**. A type size of 1 em makes for comfortable reading by most people. If smaller sizes are necessary, then choose fonts with wider spacing and larger x-heights. The x-height is the height of lower-case letters measured from the baseline. The following two lines are displayed at .67em size in Arial and Verdana type faces. The latter line is not as crowded at that size and is easier to read.

A quick brown fox jumps over the lazy dog.

A quick brown fox jumps over the lazy dog.

Figure **2.4.6**: Comparison of type faces and type size.

Incidentally, Verdana is becoming a popular Web font because it is designed to map accurately to pixel positions on a screen. Also, a non-serif font such as Arial or Verdana is easier to read on screen, while a serif font such as Times New Roman is easier to read on paper.

While print type size is based on absolute units of measure such as points and pixels,relative units such as **percentages** and **em (preferred)**, should be used in web documents since absolute font sizes render inconsistently across platforms and can't be resized by the browser.

1 em unit is equivalent to **12 points**.

Line Size

Readability also depends on line height and line length. In general, default spacing in the browser provides decent spacing between lines. A line length of approximately 60 to 80 characters permits good tracking of the eyes from the end of one line to the beginning of the next. The following lines contain approximately 75 Verdana characters at 9-pt size.

The standard **Lorem Ipsum** passage, used since the 1500s: "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged.

Figure \(\PageIndex{7\): Typical font size, line length, and line spacing.

White Space

A prevelant mistake of page authors is to crowd too much information between tiny margins. It can be monotonous and tiring reading line after line of text from border to border without breaks. Make sure you leave *plenty* of white space in the margins and

that sufficient numbers of headings and graphics break up the monotony of the page.

Text Color

Color can be used for text variety. As in the case with graphic images, choose a small number of coordinated colors and be consistent in their use. Readers tend to see colors, as they do text sizes, as visual clues with logical meaning. For instance, you will normally use different font sizes for different levels of headings. The size of the font comes to mean a certain type of topical change in content. By the same token, colors take on expected meanings. Be consistent in the color cues you give to readers.

ecorative Fonts

One way to introduce nonstandard fonts on a Web page is to use graphic images of letterforms like the character "D" in the previous heading. This technique ensures that the font will be displayed irrespective of the visitor's installed fonts, and it introduces variety to the page. Use graphic letters selectively, though, since they increase page download times and they increase the risk of cluttering the page with hard-to-read characters.

Dynamic Style Settings

The current page is rendered in 10pt Arial font. You can see how changing fonts and sizes changes the appearance and readability of this page by clicking the following buttons:

◉ Arial	○ .67em (8pt)
○ Verdana	○ .75em (9pt)
○ Georgia	◉ .83em (10pt)
○ Times New Roman	○ 1 em (12pt)
○ Courier New	○ 1.2 em (14pt)

Figure **2.4.8**:

Enhancing the Usability of A Page

The following guidelines should be used to help increase the readability of your Web pages:

- Use common fonts such as Arial (Helvetica), Verdana, or Times New Roman (Times). San serif fonts are easier to read on a computer screen, whereas serif fonts are easier to read in print. Use the CSS media types - @print and @screen to set appropriate fonts for each media type.
- Avoid long blocks of text and long paragraphs. Be concise
- When possible use relative units such as the em when setting font size - 1 em = 12 pt.
- Hyperlink keywords or phrases, not entire sentences
- Avoid the use of the words "click here"
- Always check grammar and spelling

2.4: Typography is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

2.5: Accessibility

Web technologies are forever forging ahead. Each day brings new gadgetry and wizardry to the browsing experience, opening up reams of electronic information and entertainment at an accelerating pace for those with the technical where-with-all to access it. Yet, many Web citizens are not in a postion to reap the advantages because of both technical deficiencies and physical disabilities for experiencing the content as originally conceived and presented.

Web **accessibility** issues pertain to page design for users who

- may not be able to see, hear, move, or may not be able to process some types of information easily or at all.
- may have difficulty reading or comprehending text.
- may not have or be able to use a keyboard or mouse.
- may have a text-only screen, a small screen, or a slow Internet connection.
- may not speak or understand fluently the language in which the document is written.
- may be in a situation where their eyes, ears, or hands are busy or interfered with (e.g., driving to work, working in a loud environment, etc.).
- may have an early version of a browser, a different browser entirely, a voice browser, or a different operating system.

Web developers should consider these situations during page design. Granted, there may be situations where it is impractical or unwise to compromise a design for accessibility purposes. A Web site that features inaccessible technologies need not apologize for lack of presentation alternatives if none are truly available to duplicate the experience in other ways. There are also cost and marketing considerations. It may be resource-prohibitive to create a parallel site for a small population of users and a competitive disadvantage to delay available until it is completed.

Still, the search for presentation alternatives should continue. In some cases there are technical solutions to accessibility problems. In other cases there are page coding remedies.

Assistive Technologies

Assistive technologies provide mechanical and software means to overcome physical or cognitive disabilities. Examples include

- **Eye movement mouse.** A camera mounted on the computer monitor focuses on the user's eye. The mouse cursor is positioned where the user is looking and mouse clicks are performed with an eye blink.
- **Foot control mouse.** A foot control mouse uses two pedals, one for cursor movement and the other for mouse clicks.
- **Head tracking mouse.** A signal from atop the computer monitor tracks a reflector placed on the user's head or eyeglasses. Movement of the user's head controls the direction and distance of cursor movement.
- **Touch sensitive keyboard/mouse.** The keyboard works with the slightest touch of a hand-held wand or mouth stick requiring no force of pressure. The mouse function works like a standard mouse but requires no hand dexterity.
- **Mouth controlled joy stick.** A pointer controls cursor movement on the screen; "sipping" or "puffing" on the mouth stick simulates mouse clicking.
- **Touch screen.** Screen overlays are sensitive to finger and wand touch to track cursor movement and activate mouse clicks.
- **Screen and text readers.** Software reads aloud or converts to brail information appearing on the monitor screen or speaks text entered through the keyboard or appearing in documents such as text files, email messages, or word processing documents.
- **Screen magnification.** Software enlarges screen display to as high as 16x magnification.
- **Sign language communicator.** Converts text or speech to video sign language or computer-generated voice.
- **Speech recognition.** Devices and software to control computers through spoken words and phrases; assistive writing by coverting speech into text without use of pencil and paper.
- **Cognitive planners/reminders.** Assist users in accessing and remembering information; provides on screen cueing and planning assistance for people with memory and attention disorders, keeping them on schedule with graphic displays, selection menus, and personalized verbal and audio signals.

Most of these solutions take place on the end-user side. They must be purchased and installed on individual computers. The Web developer has little or no impact on whether Web content may be accessible by these methods. The developer can, however, assist many of these assistive technologies by following compatible site design and coding practices when creating Web pages.

W3C Accessibility Guidelines

The World Wide Web Consortium (W3C) produces a set of *Web Content Accessibility Guidelines* (http://www.w3.org/TR/WCAG10/) on how to make Web content accessible to people with disabilities. By following these guidelines content developers can create pages that remain accessible despite the constraints of physical, sensory, and cognitive disabilities. Some general keys to designing such pages include:

- Separate structure from presentation. Recognize that the information content of a Web page is different from, and usually separate from, its visual appearance.
- Provide text alternatives to audio and video content. Text can be rendered in ways that are available to almost all browsing devices and accessible to almost all users.
- Create documents that do not rely on one type of hardware. Pages should be usable by people without mice, with small screens, low resolution screens, black and white screens, no screens, and with only voice or text output.

The W3C provides fourteen guidelines under three priority levels commensurate with their impact on accessibility. Some of these guidelines refer to technologies or techniques that are not covered in this tutorial.

Priority 1 guidelines **must** be satisfied; otherwise, one or more groups will find it impossible to access information in the document.

Priority 2 guidelines **should** be satisfied; otherwise, one or more groups will find it difficult to access information in the document.

Priority 3 guidelines **may** be satisfied; otherwise, one or more groups will find it somewhat difficult to access information in the document.

The following discussion summarizes these guidelines. You should visit the W3C accessibility site for complete details.

Guideline 1. Provide equivalent alternatives to auditory and visual content.

This guideline emphasizes the importance of providing text equivalents of non-text content (images, pre-recorded audio, video). The power of text equivalents lies in their capacity to be rendered in ways that are accessible to people from various disability groups using a variety of technologies. Text can be readily output to speech synthesizers and braille displays, and can be presented visually in a variety of sizes on computer displays and paper.

Coding Practices:

- Provide a text equivalent for every non-text element. For example, use alt text for the `<img>` tag. For complex content where alt text does not provide a complete text equivalent, provide a link to a text description. For image maps use the alt attribute with `<area>` tags. [Priority 1]
- Until user agents can automatically read aloud the text equivalent of a visual track, provide an auditory description of the important information of the visual track of a multimedia presentation. [Priority 1]
- For any time-based multimedia presentation (e.g., a movie or animation), synchronize equivalent alternatives (e.g., captions or auditory descriptions of the visual track) with the presentation. [Priority 1]
- Until user agents render text equivalents for image map links, provide redundant text links for each active region of a client-side image map. [Priority 3]

Guideline 2. Don't rely on color alone.

Ensure that text and graphics are understandable when viewed without color. If color alone is used to convey information, people who cannot differentiate between certain colors and users with devices that have non-color or non-visual displays will not receive the information. When foreground and background colors are too close to the same hue, they may not provide sufficient contrast when viewed using monochrome displays or by people with different types of color deficits.

Coding Practices:

- Ensure that all information conveyed with color is also available without color, for example from text context or markup. [Priority 1]

- Ensure that foreground and background color combinations provide sufficient contrast when viewed by someone having color deficits or when viewed on a black and white screen. [Priority 2].

Guideline 3. Use markup and style sheets properly.

Mark up documents with the proper and intended use of XHTML elements, that is, as information structuring devices rather than styling techniques. Control presentation with style sheets rather than with attributes. Using markup improperly hinders accessibility.

Coding Practices:

- When an appropriate markup language exists, use markup rather than images to convey information. Avoid using images to represent text -- use text and style sheets instead. [Priority 2]
- Create documents that validate to published XHTML standards. [Priority 2]
- Use style sheets rather than deprecated attributes to control layout and presentation. [Priority 2]
- Use relative measurement units (percentages) rather than absolute units (pixels) in markup attribute values and style sheet property values. [Priority 2]
- Use header elements to convey document structure, not for style effects. For example, use H2 to indicate a subsection of H1, not because of size differences in the fonts. [Priority 2]
- Nest OL, UL, and DL list items properly. [Priority 2]

Guideline 4. Clarify natural language usage

Use markup that facilitates pronunciation or interpretation of abbreviated or foreign text. When content developers code natural language in a document, speech synthesizers and braille devices can automatically switch to the new language, making the document more accessible to multilingual users. Also, natural language markup allows search engines to find key words and identify documents in a desired language.

Coding Practices:

- Clearly identify changes in the natural language of a document's text and captions. For example, use "xml:lang" in page prologs. [Priority 1]
- Specify the expansion of each abbreviation or acronym in a document where it first occurs. [Priority 3]

Guideline 5. Create tables that transform gracefully.

Ensure that tables have necessary markup to be transformed by accessible browsers and other user agents. Tables should be used to mark up truly tabular information ("data tables") and should be avoided to lay out pages ("layout tables"). Tables for any use present special problems to users of screen readers.

Coding Practices:

- For data tables, identify row and column headers. Use `<td>` to identify data cells and `<th>` to identify headers. [Priority 1]
- For data tables that have two or more logical levels of row or column headers, use markup to associate data cells and header cells. Use `<thead>` , `<tbody>` , and `<tfoot>` to group rows and use `</col>` and `<colgroup>` to group columns. [Priority 1]
- Do not use tables for layout unless the table makes sense when presented in a linear (across rows and down columns) fashion. Otherwise, provide an alternative presentation. [Priority 2]
- If a table is used for layout, do not use any structural markup for the purpose of visual formatting -- do not use the `<th>` element to cause the content of a (non-table header) cell to be displayed centered and in bold. [Priority 2]
- Provide text summaries for tables where practical. [Priority 3]

Guideline 6. Ensure that pages featuring new technologies transform gracefully.

Ensure that pages are accessible even when newer technologies are not supported or are turned off. Although content developers are encouraged to use new technologies, they should know how to make their pages still work with older browsers and people who choose to turn off features.

Coding Practices:

- Organize documents so they may be read without style sheets. For example, when an HTML document is rendered without associated style sheets, it must still be possible to read the document. [Priority 1]
- Ensure that equivalents for dynamic content are updated when the dynamic content changes. [Priority 1]
- Ensure that pages are usable when scripts, applets, or other programmatic objects are turned off or not supported. If this is not possible, provide equivalent information on an alternative accessible page. [Priority 1]
- Ensure that dynamic content is accessible or provide an alternative presentation or page. [Priority 2]

Guideline 7. Ensure user control of time-sensitive content changes.

Ensure that moving, blinking, scrolling, or auto-updating objects or pages may be paused or stopped. Some people with cognitive or visual disabilities are unable to read moving text quickly enough or at all. Movement can also cause such a distraction that the rest of the page becomes unreadable for people with cognitive disabilities. Screen readers are unable to read moving text. People with physical disabilities might not be able to move quickly or accurately enough to interact with moving objects.

Coding Practices:

- Until user agents allow users to control flickering, avoid causing the screen to flicker. [Priority 1]
- Until user agents allow users to control blinking, avoid causing content to blink [Priority 2]
- Until user agents allow users to freeze moving content, avoid movement in pages. [Priority 2]
- Until user agents provide the ability to stop the refresh, do not create periodically auto-refreshing pages. [Priority 2]
- Until user agents provide the ability to stop auto-redirect, do not use markup to redirect pages automatically. [Priority 2]

Guideline 8. Ensure direct accessibility of embedded user interfaces.

Ensure that the user interface follows principles of accessible design: device-independent access to functionality, keyboard operability, self-voicing, etc. When an embedded object has its "own interface", the interface -- like the interface to the browser itself -- must be accessible. If the interface of the embedded object cannot be made accessible, an alternative accessible solution must be provided.

Coding Practices:

- Make programmatic elements such as scripts directly accessible or compatible with assistive technologies [Priority 1 if functionality is important and not presented elsewhere, otherwise Priority 2.]

Guideline 9. Design for device-independence.

Use features that enable activation of page elements via a variety of input devices. Device-independent access means that the user may interact with the user agent or document with a preferred input (or output) device -- mouse, keyboard, voice, head wand, or other.

Coding Practices:

- Provide client-side image maps instead of server-side image maps except where the regions cannot be defined with an available geometric shape. [Priority 1]
- Ensure that any element that has its own interface can be operated in a device-independent manner. [Priority 1]
- For scripts, specify logical event handlers rather than device-dependent event handlers. [Priority 2]
- Create a logical tab order through links, form controls, and objects. [Priority 3]

- Provide keyboard shortcuts to important links, form controls, and groups of form controls. [Priority 3]

Guideline 10. Use interim solutions.

Use interim accessibility solutions so that assistive technologies and older browsers will operate correctly. For example, changing the current window or popping up new windows can be very disorienting to users who cannot see that this has happened.

Note. The following checkpoints apply until user agents (including assistive technologies) address these issues. These checkpoints are classified as "interim", meaning that the Web Content Guidelines Working Group considers them to be valid and necessary to Web accessibility as of the publication of this document. However, the Working Group does not expect these checkpoints to be necessary in the future, once Web technologies have incorporated anticipated features or capabilities.

Coding Practices:

- Until user agents allow users to turn off spawned windows, do not cause pop-ups or other windows to appear and do not change the current window without informing the user. [Priority 2]
- Until user agents support explicit associations between labels and form controls, for all form controls with implicitly associated labels, ensure that the label is properly positioned immediately precedits its control on the same line. [Priority 2]
- Until user agents (including assistive technologies) render side-by-side text correctly, provide a linear text alternative (on the current page or some other) for all tables that lay out text in parallel, word-wrapped columns. [Priority 3]
- Until user agents handle empty controls correctly, include default, place-holding characters in edit boxes and text areas. [Priority 3]
- Until user agents (including assistive technologies) render adjacent links distinctly, include non-link, printable characters (surrounded by spaces) between adjacent links. [Priority 3]

Guideline 11. Use W3C technologies and guidelines.

Use W3C technologies and follow accessibility guidelines. Where it is not possible to use a W3C technology, or doing so results in material that does not transform gracefully, provide an alternative version of the content that is accessible.

Coding Practices:

- Use W3C technologies when they are available and appropriate for a task and use the latest versions when supported. [Priority 2]
- Avoid deprecated features of W3C technologies. [Priority 2]
- Provide information so that users may receive documents according to their preferences (e.g., language, content type, etc.) [Priority 3]
- If, after best efforts, you cannot create an accessible page, provide a link to an alternative page that uses W3C technologies, is accessible, has equivalent information (or functionality), and is updated as often as the inaccessible (original) page. [Priority 1]

Guideline 12. Provide context and orientation information.

Provide context and orientation information to help users understand complex pages or elements. Grouping elements and providing contextual information about the relationships between elements can be useful for all users. Complex relationships between parts of a page may be difficult for people with cognitive disabilities and people with visual disabilities to interpret.

Coding Practices:

- Title each frame to facilitate frame identification and navigation. [Priority 1]
- Describe the purpose of frames and how frames relate to each other if it is not obvious by frame titles alone. [Priority 2]
- Divide large blocks of information into more manageable groups where natural and appropriate. [Priority 2]
- Associate labels explicitly with their controls. [Priority 2]

Guideline 13. Provide clear navigation mechanisms.

Provide clear and consistent navigation mechanisms -- orientation information, navigation bars, a site map, etc. -- to increase the likelihood that a person will find what they are looking for at a site. Clear and consistent navigation mechanisms are important to people with cognitive disabilities or blindness, and benefit all users.

Coding Practices:

- Clearly identify the target of each link. Link text should be meaningful enough to make sense when read out of context -- either on its own or as part of a sequence of links. Link text should also be terse.[Priority 2]
- Provide metadata to add semantic information to pages and sites. For example, indicate the document's author, the type of content, etc.[Priority 2]
- Provide information about the general layout of a site (e.g., a site map or table of contents). [Priority 2]
- Use navigation mechanisms in a consistent manner. [Priority 2]
- Provide navigation bars to highlight and give access to the navigation mechanism. [Priority 3]
- Group related links, identify the group (for user agents), and, until user agents do so, provide a way to bypass the group. [Priority 3]
- If search functions are provided, enable different types of searches for different skill levels and preferences. [Priority 3]
- Place distinguishing information at the beginning of headings, paragraphs, lists, etc. [Priority 3]
- Provide information about document collections (i.e., documents comprising multiple pages.). Another way to create a collection is by building a zipped archive of the multiple pages. [Priority 3]
- Provide a means to skip over multi-line ASCII art. [Priority 3]

Guideline 14. Ensure that documents are clear and simple.

Ensure that documents are clear and simple so they may be more easily understood. Consistent page layout, recognizable graphics, and easy to understand language benefit all users. They help people with cognitive disabilities or who have difficulty reading. Ensure that images have text equivalents for people who are blind, have low vision, or for any user who cannot or has chosen not to view graphics.

Coding Practices:

- Use the clearest and simplest language appropriate for a site's content. [Priority 1]
- Supplement text with graphic or auditory presentations where they will facilitate comprehension of the page. [Priority 3]
- Create a style of presentation that is consistent across pages. [Priority 3]

It is impossible in this tutorial to give examples of all of the above coding techiques to assist in Web page accessability. A comprehensive set of example HTML code, however, is provided at the W3C Web site: Techniques for Web Content Accessibility Guidelines.

There are a number of Web sites that provide software for testing pages against accessibility guidelines. For example, bobby.watchfire.com permits you to enter the URL of your page and receive a report on its compatibility with W3C and other guidelines.

Section 508 Requirements

Under Federal law, U.S. agencies must make their Web sites accessible to people with vision and hearing impairments, with limited dexterity, and with other disabilities. Section 508, an amendment to the Rehabilitation Act of 1973, mandates that people with disabilities be given comparable access to Web-accessible government information as other users; the Act also pertains to organizations which receive funds from the federal government.

There are sixteen general rules for accessible Web pages in the Section 508 Standards. In brief, the rules for accessible Web pages are:

- **Text Tags:** A text equivalent for every non-text element shall be provided.
- **Multimedia Presentations:** Equivalent alternatives for any multimedia presentation shall be synchronized with the presentation.
- **Color:** Web pages shall be designed so that all information conveyed with color is also available without color, for example from context or markup.
- **Readability:** Documents shall be organized so they are readable without requiring an associated style sheet.
- **Server-Side Image Maps:** Redundant text links shall be provided for each active region of a server-side image map.
- **Client-Side Image Maps:** Client-side image maps shall be provided instead of server-side image maps except where the regions cannot be defined with an available geometric shape.
- **Data Tables 1:** Row and column headers shall be identified for data tables.
- **Data Tables 2:** Markup shall be used to associate data cells and header cells for data tables that have two or more logical levels of row or column headers.
- **Frames:** Frames shall be titled with text that facilitates frame identification and navigation.
- **Flicker Rate:** Pages shall be designed to avoid causing the screen to flicker with a frequency greater than 2 Hz and lower than 55 Hz.
- **Text-Only Alternatives:** A text-only page, with equivalent information or functionality, shall be provided to make a web site comply with the provisions of this part, when compliance cannot be accomplished in any other way. The content of the text-only page shall be updated whenever the primary page changes.
- **Scripts:** When pages utilize scripting languages to display content, or to create interface elements, the information provided by the script shall be identified with functional text that can be read by assistive technology.
- **Applets and Plug-Ins:** When a web page requires that an applet, plug-in or other application be present on the client system to interpret page content, the page must provide a link to a plug-in or applet.
- **Forms:** When electronic forms are designed to be completed on-line, the form shall allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.
- **Navigation Links:** A method shall be provided that permits users to skip repetitive navigation links.
- **Time:** When a timed response is required, the user shall be alerted and given sufficient time to indicate more time is required.

Details about these guidelines along with links to other resources for Section 508 compatibility are at the Federal government site www.section508.gov/.

2.5: Accessibility is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

2.6: Web Site Development

Web Site Development Job Roles

The task of developing a major Web site is rarely undertaken by a single individual. In most cases, development teams are established and work is divided among experts or individuals with certain job roles. The following are some job roles that are typically necessary for the successful development of a large scale Web development project. In smaller projects, a single individual may take on multiple roles.

Project Manager

Manages the Web site development project and coordinates all team activities. The project manager creates the project schedule and plan. The project manager must ensure that all project milestones are met and that all outcomes are met.

Information Architect

The information architect clarifies the mission and goals of the Web site, determines the functionality of the site, helps define site organization, layout, and navigation.

Marketing or Public Relations Representative

The PR representative works with the Web developers to help define the look and feel for the Web site that aligns with the organization's marketing goals. PR representatives may also assist with editing the Web site content.

Multimedia Designer

Assists with determining the appropriate graphics, colors, and layout for the site. Creates page layout and designs graphics and other multimedia content.

Database Administrator

Creates databases, tables, stored procedures, writes SQL statements, and maintains control and access to databases.

Network or Systems Administrator

Configures, maintains, and secures the Web server. Installs and maintains hardware and systems level software.

Web Developer

Writes the HTML code and client or server-side scripts.

Like any significant technology development, Web sites are developed by following a project development methodology or step-by-step plan that encompasses the life cycle of a project from start to finish. Web sites are often developed using a variation of the traditional Systems Development Life Cycle (SDLC). The Web Systems Development Life Cycle consists of the following phases:

- Conceptualization - Identify Opportunity, What is the motivation for the site, Identify audience, Determine purpose and goals of the site
- Analysis - Determine functional requirements, determine environmental requirements, determine content requirements, estimate costs, review competitors' sites
- Design - choose site organization, storyborad the design, create page layout, document each page
- Production - select Web authoring tool, organize site files, develop and test components
- Testing - test site on different browsers, test with different screen resolutions, test from different locations, test using mobile devices, accessibility testing, site validation
- Launch - Publish site to live production Web server
- Maintenance - enhancements, modifications, additions to the site over time
- Evaluation - review site to ensure that it meets goals

Web Site Hosting

A Web site hosting service is a type of Internet hosting service that allows individuals and organizations to make their own Web site accessible via the World Wide Web. Web hosts are companies that provide space on a server they own or lease for use by their clients as well as providing Internet connectivity, typically in a data center.There are various types of web hosting solutions. Depending on your needs, you could choose from one of several web hosting solutions and therefore it is important to research and evaluate the services to select the most appropriate option. Some of the most common types of web hosting services available include:

- Shared Hosting -Shared web hosting (also called virtual hosting) is the most common type of hosting service used. Under this hosting, the Web site is placed on the same server as many other sites, ranging from a few to hundreds or thousands.The Hosting company manages the server to ensure uptime, upgrading hardware and software.
- Dedicated Hosting - dedicated web hosting solutions allow a client to lease the complete computer and have access to all its resources to host one or multiple websites. The user gets his or her own Web server and gains full control over it, however, the user typically does

not own the server.

- Virtual Private Server (VPS) Hosting - divides server resources into virtual servers, where resources can be allocated in a way that does not directly reflect the underlying hardware. Each server looks and acts like a dedicated server.
- Managed Web Hosting - the user gets his or her own Web server but is not allowed full control over it, however, they are allowed to manage their data via FTP or other remote management tools
- Free Web Hosting - offered by different companies with limited services, sometimes supported by advertisements, and often limited to basic Web Hosting services

A number of issues should be considered when choosing a Web host, including bandwidth, disk storage space, technical support, availability of server-side scripting languages (ASP.NET, PHP, Java, Coldfusion), database server availability (MySQL, MS SQL Server), type of Web server (IIS, Apache), and the server operating system (Windows, Linux). The host may also provide an interface or control panel for managing the Web server and installing scripts as well as other services like e-mail. Some hosts specialize in certain software or services (e.g. e-commerce). They are commonly used by larger companies to outsource network infrastructure to a hosting company.

Domain Name Selection

A domain name is an identification label that defines a realm of administrative autonomy, authority, or control on the Internet, based on the Domain Name System (DNS). An important purpose of domain names is to provide easily recognizable and memorizable names to numerically addressed Internet resources (i.e., www.maconstateit.net instead of 74.208.73.193). The right to use a domain name is delegated by domain name registrars which are accredited by the Internet Corporation for Assigned Names and Numbers (ICANN), the organization charged with overseeing the name and number systems of the Internet. After determining the availablity of a domain name, you can register the domain name through a domain name registrar or through your Internet Hosting Company. The best way to determine whether a domain name is available is to use the Domain Name Look up Tool provide by Whois.

The following are basic guidelines to following when selecting a domain name:

- Select a domain name that is as short as possible
- Select a domain name that is as generic as possible, but related (resale value, trademarks etc.)
- Avoid using hyphens, numbers or misspellings in the domain name
- A domain name that is easy to remember
- A domain name that is easy to spell
- A .com domain is universally recognized and remembered. The newer extensions such as .biz, .lastname etc will take a few years to be accepted.
- Avoid doubling up of letters - e.g. www.websiteengine.com. Instead use www.websitengine.com.

Web Site Marketing

After your Web site is hosted and accessible via the World Wide Web, various marketing techniques can be incorporated to help promote and increase access to your site. Some common and easy to implement Web marketing techniques are listed below:

- Keyword Rich Page Titles
- META Description and Keywords Tags
- Include ATL attribute with the `<img/>` tag
- Create a Site Map
- Submit Site to Directories and Search Engines
- Use Social Networking to Promote Web Site
- Use Online Forums

An example of a keyword rich page tile and meta description and keyword tags is shown below:

```
<head>

    <title>  Web Development Tutorials - XHTML, CSS, JavaScript, ASP.NET, PHP  </title>

    <meta name="description" content="Web Development tutorials for ASP.NET, XHTML, JavaScript,"
        DHTML, and XML"/>

    <meta name="keywords" content="ASP.NET 2.0, XHTML, ASP, Xml, HTML, JavaScript, DHTML,
        Tutorial, Web Development, Active Server Pages" /> "

</head>
```

Figure **2.6.1**: Title element with keywords.

2.6: Web Site Development is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

2.7: Mobile Web Development

First generation computers relied on vacuum tubes. These early computers were typically very large, sometimes as big as a room, and relatively slow by modern standards. As Moore's Law has continually progressed over the years we've now got nearly the breadth of all human knowledge available at our fingertips every day. They've gone by a few different names over the years but typically when we say Smartphone we recognize the distinction between these small hand-held devices and what is *just a mobile phone*. Typically a smart phone has much more capability in what it could do compared to a mobile phone. Now (2015) we have a wide range of mobile devices consisting of smartphones, tablets, and watches that access websites and run software applications.

Device Resolutions

On a desktop computer, the user can typically set a resolution that works best for them. The monitor itself may have a specific aspect ratio in which many specific resolutions may apply. Typically a maximum resolution will make items smaller, but clearer. If you use an resolution smaller than the maximum supported items may look larger relatively but possibly less clear. In the late stages of CRT monitors' prevalence most monitors seemed to only go as high as 1024x768 pixels but today many modern High Definition (HD) monitors display at 1920x1080 pixels or higher.

Smartphones (and tablets) come in a wide range of sizes and resolutions. Many smartphones have resolutions smaller than 1024x768 (an old CRT monitor) but since the phone's size is much smaller than a CRT's screen size we get very clear images even at these smaller resolutions.

Responsive Design with Media Queries

Using Media Queries is a technique that allows certain CSS to apply under specific circumstances. This helps designers of websites to adjust CSS properties of the site to create a better browsing experience for the user.

To start using Responsive Design:

- Set the Viewport using into the `<head>` section of your HTML page. This sets the width of the browser window for the Media Query.
- Create a Media Query to hold a set of CSS that can apply at specific ranges. W3C Schools suggests using a Mobile First approach so that designs can start small and get more complex as more room is opened up at larger widths.

Mobile First Example

This examples shows a bar that is 20px tall and changes colors based on the width of the browser. It starts as green and changes as the width gets larger. Try resizing your browser window to see the bar change colors.

```
<style>
	.colorChanger { width:60%; margin:0 15%; height:20px; background-color:green; }
	@media only screen and (min-width:600px) {
		.colorChanger { background-color:orange; }
	}
	@media only screen and (min-width:900px) {
		.colorChanger { background-color:blue; }
	}
	@media only screen and (min-width:1200px) {
		.colorChanger { background-color:red; }
	}
	@media only screen and (min-width:1500px) {
		.colorChanger { background-color:yellow; }
	}
</style>

<div class="colorChanger">

</div>
```

Figure **2.7.1**: Code for resizing sections of a page with browser size.

Choosing Breakpoints

In the above example we wrote Media Queries at fairly random intervals (every 300px) to achieve the color change. These breakpoints aren't really standardized and requires the developer to use some judgement as to when a breakpoint is needed. In a mobile first design a developer may choose to add breakpoints to open up the page for more content or to rearrange the design into a 2 or 3 column layout.

It is a free and open internet to all. It is not suggested to write media queries to target specific device resolutions. Instead, write media queries along the range of width that make it look good across devices in the range. It may take actually testing these devices to make sure it performs as intended.

2.7: Mobile Web Development is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

3: Basic Document Layout

3.1: Structuring Page Content
3.2: Document Structure Tags
3.3: Paragraphs
3.4: Line Breaks
3.5: Headings
3.6: Horizontal Rules
3.7: List Structures
3.8: Linking Pages
3.9: Displaying Pictures

3: Basic Document Layout is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

3.1: Structuring Page Content

In this chapter we will look at some of the HTML tags that help define the **template** of a page. The template is an HTML structure that defines how your site looks across many pages. Each page in a website will be different but each page should follow a **common brand** as the other pages in the website. By defining and committing to a brand, users will be able to distinguish your website from other websites. If a page in your website does not follow the common brand users may feel inclined to leave your website. Think about it. If you were reading your local news site and you go to a page that doesn't look like it belongs to them, what do you do? Most would probably close the page and maybe leave your website altogether.

HTML describes and structures, CSS styles

The primary purpose of HTML tags is to structure and describe the text and graphic content of a Web page. The function of HTML tags is not styling and decorating content, but rather presenting content in such a logical and organized manner that allows the browser to understand and to recognize the information. Browsers render many HTML elements with a default style that proves useful in arranging content on a page; however, these default styles may be changed by using CSS.

In addition to structuring the content of a page, visual styles may also be applied. Different sizes, colors, or text fonts are some of the visual changes that can be made. Graphic images can also be resized and decorated. These tutorials will introduce styling methods that add such kinds of visual appeal to text and graphic elements. Styling elements is primarily done through CSS instead of through HTML. Before covering CSS and advanced styling, it is necessary to become familiar with basic HTML tags that can describe content and be used by browsers to provide some degree of arrangement and style to a page.

Editor versus Browser Displays

Web page design begins with a standard document setup that gives the page its overall structure. The image below reproduces the standard page template.

The ideal gas law is easy to remember and apply in solving problems, as long as you get the **proper values a**

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <title> page title goes here </title>
    </head>
    <body>
        *
        page content goes here
        *
    </body>
</html>
```

Listing 3.1.1 A web page template

Every page designed begins with the coding represented in the image above. The title of the page must be coded within the `<title>` tag. When the page is opened, the text will appear in the browser's "Title Bar." Informative content should be added

inside the <body> tag. Only the information appearing in the body of the document shows up in the browser window. Text and graphic images are placed inside the <body> tag, and are then surrounded by HTML tags to order and arrange them.

It is important to understand that formatting Web pages can not take place without the usage of HTML tags. To emphasize this point consider the heading and three paragraphs coded in the Notepad editor in Figure 2-1. The information in the browser should be displayed in this same general format.

Figure **3.1.1**: Unformatted web document appearing in the Notepad editor ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

As shown above, a heading appears at the top of the document, and all subsequent paragraphs are separated from each other by single blank lines. When entering information into the editor, care must always be taken to ensure that the information entered will appear in the browser in an orderly and readable fashion. Editor formatting has little to do with how the entered information appears as a Web page, but it is still a valuable practice to develop to avoid such careless mistakes as interchanging words like deprecating and defecating.

When the document exemplified above is saved as an HTML file (.htm or .html) and opened in the browser, it is displayed as shown in the below Figure 2-2.



Figure **3.1.2**:Unformatted Web page appearing in the browser. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The appearance of the information is very different from how it is displayed in the editor. Of course, the reason for this is that no HTML tags appear in the document to guide the browser in laying out the text on the page. Therefore, the browser treats the information as one continuous string of text. All blank spaces and line breaks appearing in the editor are collapsed to single blank spaces separating the words. The result is a single block of text on the page.

Formatted Text Content

It is an easy step to return to the Notepad editor and include some basic HTML tags to arrange the text for display in the browser. In this case, as shown in Figure 2-3, paragraph tags (coded as <p> container tags) surround the separate paragraphs and a header tag (coded as an <h2> container tag) surrounds the heading line.

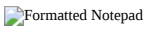


Figure **3.1.2**: Formatted web document appearing in the Notepad editor. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Now, when the page is opened in a browser, these tags cause the text blocks to be displayed as separate paragraphs as shown in Figure 2-4. The formatting is far from fancy, but it does illustrate the point that the layout of page elements is exclusively controlled by the HTML tags that appear in the document.



Figure **3.1.3**: Formatted Web page appearing in the browser. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The next several topics introduce a basic set of HTML tags that bring order and structural formatting to page content. The initial focus is on text arrangement. Later topics will introduce basic tags for adding graphic images and links. For the time being, learn the structural aspects of page design, and begin becoming comfortable with the styling techniques introduced in the following tutorial.

3.1: Structuring Page Content is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

3.2: Document Structure Tags

HTML 5 includes a new set of semantic document structural tags to define different parts or structure of a web page. A semantic element clearly describes its meaning to both the browser and the web developer.

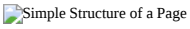


Figure **3.2.1**: A simple template "MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0

Some of the most common HTML 5 structural elements are described below.

The `<header>` tag

The `<header>` element is used as a container for page heading content. The tag typically contains heading tags (`<h1>` - `<h6>`), a logo image, or other content that introduces the content. It is possible to have more than one `<header>` tag per page; however, the `<header>` cannot be coded within a `<footer>` or another `<header>` tag.

The `<nav>` tag

The `<nav>` tag is a block level element used to define major blocks of navigation links or navigation menus.

The `<main>` tag

The `<main>` tag is used to serve a container for major page content that is not repeated in other parts of the page. A web page can include only one `<main>` element. Additionally, it cannot be included as a child element within a `<header>` , `<nav>` , `footer>` , `<article>` , or `<aside>` tag.

The `<section>` tag

The `<section>` tag defines generic sections in a web page such as headers, footers, or any other sections of the document. In this context, a section is a thematic grouping of content.

The `<article>` tag

The `<article>` element represents a complete, or self-contained, composition in a document, page, application, or site that is, in principle, independently distributable or reusable.

The `<aside>` tag

The `<aside>` tag is a block level element that defines content aside from the content it is placed in. The aside content should be related to the surrounding content.

The `<footer>` tag

The `<footer>` tag is a block level element that defines footer information for an entire web page or section of the document. The content of the `<footer>` typically consists of contact information, copyright, links, or logos. In some cases, a web page may contain multiple `<footer>` tags.

The `<figure>` tag

The `<figure>` tag specifies self-contained content, like illustrations, diagrams, photos, or code sections. By default, this is a block level element with 40px right and left margin settings.

The `<figcaption>` tag

The `<figcaption>` is a block level tag that defines a caption for a `<figure>` element discussed in the previous section. The `<figcaption>` tag is normally coded as the first or last child element of the `<figure>` tag.

3.2: Document Structure Tags is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

3.3: Paragraphs

The <p> Tag

Most Web pages are text documents. In written languages such as English, letters are grouped into words, words are organized into sentences, and sentences are structured into cohesive paragraphs. HTML provides a way for content to be marked as a paragraph. By default, browsers render text marked as a paragraph with a single blank line preceding and following the text. Text inside a paragraph is also aligned to the left margin of the page and is word-wrapped inside the browser window.

In order to format a block of text as a paragraph, the <p> (paragraph) tag surrounds the enclosed text. The general format for this tag is given shown below.

```
<p> Text </p>
```

Any amount of text can appear between the opening and closing tags. When displayed in the browser, the enclosed text is single spaced and preceded and followed by a single blank line to separate it from surrounding page content. The <p> tag is classified as a container and block-level element.

You can see the effects of <p> tags in the Web page shown above which displays text as three separate paragraphs. These tags are added in the editor as shown below.

```
<p>
 A Web page begins with preliminary sections of code th
 HTML5 document. Also included is a title which appears
 Title Bar.
</p>

<p>
 All content that appears in the browser window is code
 of the page. Text information, graphic images, links,
 are added to the page and are surrounded by HTML tags
 them for presentation.
</p>

<p>
 It is important to note that content is arranged on th
 HTML tags coded in the document. Otherwise, all conten
 unformatted string of text.
</p>
```

The text is now easily readable because organization is brought to its information content. Note that back-to-back paragraphs are separated by a single blank line even though each <p> tag itself produces a blank line before and after a paragraph. The browser collapses into a single blank line: the one that follows a paragraph and the one that precede the next.

> *Note that on this and subsequent pages complete coding for a Web page may not be shown in order to focus on the code being discussed. Still, it is assumed that the code shown is surrounded by the page preliminaries down through the* `<body>` *tag and by the closing* `</body>` *and* `</html>` *tags.)*

The `<blockquote>` Tag

The `<blockquote>` tag offers a way to indent a block of text, to offset it from surrounding content for special emphasis as would be done for a quoted citation from which the tag gets its name. This container tag indents its enclosed text a fixed number of pixels (approximately 40 pixels) from the left and right margins. Its enclosed text is word-wrapped and preceded and followed by a blank line. The general format for the `<blockquote>` tag is shown below.

```
<blockquote>
   text
</blockquote>
```

According to HTML 5 standards, the `<blockquote>` is used to mark content that is quoted from another source. In practice, the `<blockquote>` should be used only in the correct context to mark a block quotation. For our purposes, it currently serves as a convenient way to indent any block of text.

In the following example, a `<blockquote>` tag offsets the middle paragraph from surrounding text. Browser output is shown in Figure **3.3.1**.

```
<p>
   Here are three blocks of text. This first paragraph is
   standard paragraph tag and is blocked at the left marg
</p>

<blockquote>
   This second block of text is formatted with a blockquo
   left and right margins approximately 40 pixels from th
</blockquote>

<p>
   This final block of text is coded like the first one.
   standard paragraph tag and is blocked at the left marg
</p>
```

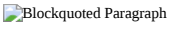


Figure **3.3.1**: Applying the `<blockquote>` tag to a paragraph. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Similar to the way it appears in the browser, the above editor coding shows the blockquoted text offset with blank lines and indented. As you know, this editor formatting is not required and has no effect on browser output that is controlled solely by HTML tags. Nevertheless, it is good practice to arrange editor coding to resemble browser output the best way possible for visual indication of the display relationships among sections of page content. It makes it easier to spot missing or miscoded tags when editor coding is well spaced and indented.

Nested Blockquotes

In the following example, `<blockquote>` tags are used to offset three blocks of text. The middle one is further indented inside its outer blockquoted paragraphs. Browser output is shown in Figure **3.3.2**.

```
<p>
 Here are five blocks of text. This first block is format
 paragraph tag and is blocked at the left margin of the p
</p>

<blockquote>
   This second block of text is formatted with a blockquo
   left and right margins approximately 40 pixels from th

    <blockquote>
     This block of text is also surrounded by blockquote
     indented within the margins produced by its outer bl
    </blockquote>

   This block of text is aligned like the second block of
   inside the outermost blockquote tag.
</blockquote>

<p>
 This fifth block of text is coded like the first one. It
 a standard paragraph tag and blocked at the left margin.
<p>
```

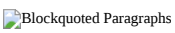


Figure **3.3.2**: Text formatted with nested `<blockquote>` tags. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

In this example, the two `<blockquote>` tags are nested; that is, a `<blockquote>` tag appears inside a `<blockquote>` tag.The outer `<blockquote>` tag surrounds and indents its three contained text blocks approximately 40 pixels from the left and right margins of the page. The inner `<blockquote>` tag indents its enclosed text an additional 40

pixels from the margins established by the outer tag. Any additional nested tags further indent their enclosed text from the previous margins.

When nesting `<blockquote>` tags inside one another, make sure that opening and closing tags are properly paired, and that inner tags are fully closed inside their next outer-most enclosing tags. As in the case with `<p>` tags, back-to-back `<blockquote>` tags have their preceding and following blank lines collapsed to a single blank line.

Remember that the `<blockquote>` tag should not be used to just indent text. Instead it should only be used when displaying large blocks of quoted text within a web document. In a later tutorial, you will learn style sheet methods that can be used for indenting text.

Deprecated (OLD) Tags and Attributes

HTML 5 is a replacement for the older XHTML and HTML4 languages. In addition to adding many new elements and features, HTML 5 retains many of the tags found in previous HTML versions. Still, many older styling attributes are no longer supported officially by HTML 5. Many of these attributes may still be recognized by browsers which support compatibility with older HTML versions. For example, the `<p>` tag historically included the align attribute to control text alignment. For instance, by setting this attribute to `<p align="center">` the text on each line of a paragraph is centered rather than aligned at the left margin.

In HTML 5, the align attribute is not supported. It is recommended that you refrain from using it and other styling attributes which have been phased out. Instead, newer style sheet declarations should be used to produce the same effect. Still, deprecated attributes are described in these tutorials since you may encounter them on existing Web pages even though most attributes are not valid under HTML 5 standards.

Certain HTML tags are also deprecated under HTML 5. Tags that are being phased out are presented since you will likely encounter them in existing pages. These tags may display properly in modern browsers, but newer alternatives should be used in their place. Furthermore, HTML 5 changes the contextual meaning of some older tags. These tags should now be used in the context where they are appropriate, instead of simply being used for styling purposes.

Deprecated (OLD) align Attribute

By default, paragraph text displayed in a browser is word wrapped and blocked at the left margin of the page. It is possible, however, to block text at the right margin, to center it, or to justify all lines by expanding them to both margins. These alignment options are specified by coding the align attribute inside a block-level tag. The general format for coding this attribute along with its possible values are:

```
align="left|center|right|justify"
```

The keyword align is assigned one of the quoted alignment values. For example, the following code centers the heading of the previous example page and justifies all paragraph lines by expanding them to the right margin. Browser output is shown in Figure **3.3.3**.

```
<h2 align="center"> Web Page Layout </h2>

<p align="justify">
 A Web page begins with preliminary sections of code that
 HTML5 document. Also included is a title which appears i
 Title Bar.
```

```
</p>

<p align="justify">
 All content that appears in the browser window is coded
 of the page. Text information, graphic images, links, an
 are added to the page and are surrounded by HTML tags to
 them for presentation.
</p>

<p align="justify">
 It is important to note that content is arranged on the
 HTML tags coded in the document. Otherwise, all content
 unformatted string of text.
</p>
```

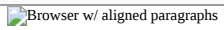


Figure **3.3.3**: Web page formatted with aligned paragraphs in browser.

Be aware that the align attribute is deprecated under current HTML5 standards. It is presented here as a quick and easy way to align paragraphs and will serve as a convenient, but temporary, styling method until you learn how to code and integrate style sheets on your Web pages.

3.3: Paragraphs is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

3.4: Line Breaks

The
 Tag

Coding paragraphs with <p> tags is the most common method of structuring text on a Web page. Most Web pages are text pages, and paragraphs are convenient and readable methods of text presentation. Other varieties of text structuring are nonetheless available.

The
 (line break) tag causes the browser to end a line of text and to continue display on the next line in the browser window. It does not, as in the case of paragraphs, leave blank lines before or after the ended text line. The general format for the break tag is shown here:

>
>
> *General format for
 tag. Note that in HTML5 the / is optional.
 is fine but note that it does not have a closing </br> .*

The
 tag is **not a container tag**. It does not enclose text and it does not have an accompanying closing tag. HTML5 elements that do not enclose text and have no accompanying closing tag are referred to void elements. This empty tag is simply placed within the text wherever a line break should occur. The tag is handy for displaying lists of items, lines of verse, or other groupings of individual, single-spaced text lines. For example,the following code uses line breaks to end individual lines of verse packaged inside a <blockquote> tag.

```
<p>
 Here is a tale about Mary and a pesky little lamb that f
 anywhere and everywhere she went.
</p>

<blockquote>
   Mary had a little lamb, <br />
   Its fleece was white as snow; <br />
   And everywhere that Mary went, <br />
   The lamb was sure to go.
</blockquote>

<p>
 Mary had an awkward social life. It's awfully difficult
 trailing around after you all the time.
</p>
```

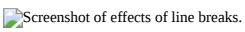


Figure 3.4.1: Using line break for single-spaced output. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The four lines of verse are enclosed inside a blockquoted paragraph to offset and indent them from surrounding paragraphs. Each line of verse appears on a separate text line on a single space below the preceding line.

Multiple Line Breaks

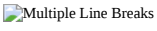
With `<p>` and `<blockquote>` tags, a single blank line appears before and after the enclosed indented text block. When `<br />` tags are inserted to end lines of text, no spacing appears between the lines. This is the correct way to use the `<br />` tag. Historically, multiple `<br />` tags were often used to insert extra blank lines in order to increase vertical spacing between lines of text or between other content appearing on the page. This practice is no longer preferred under HTML 5, and should be replaced with style sheets where necessary.

Below is a recoding of the previous page to leave an additional blank line before and after the blockquoted verse. These blank lines are produced by coding `<br />` tags to force additional line breaks. Browser output is shown in Figure **3.4.2**.

```
<p>
   Here is a tale about Mary and a pesky little lamb that
   anywhere and everywhere she went. <br /> <br /> <br /
</p>

<blockquote>
   Mary had a little lamb, <br />
   Its fleece was white as snow; <br />
   And everywhere that Mary went, <br />
   The lamb was sure to go.
</blockquote>

<p>
  <br />
 Mary had an awkward social life. It's awfully difficult
 trailing around after you all the time.
</p>
```



3.4.1 Using line break for single-spaced output. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Each `<br />` tag inserts an additional line break on the page. Therefore, you can code multiple `<br />` tags back-to-back to produce multiple blank lines down the page.

Notice that it requires three `<br />` tags at the end of the first paragraph to match the number of blank lines given by the single `<br />` tag at the beginning of the last paragraph. This is a peculiarity of the way the browser renders the tag at the beginning and end of paragraphs. You may have to experiment to determine the number of tags to code to produced the desired number of blank lines. Still, it is preferable to control vertical spacing with other block level elements or style sheets.

3.4: Line Breaks is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

3.5: Headings

The Heading Element

Text appearing on a Web page is displayed in a default font style and size unless specified otherwise. Most browsers display text in the Times New Roman font face, or font family at approximately 12-point size. In order to add visual variety to your pages, you will likely want to choose other font faces and sizes for different sections of your document. This is a topic that is taken up in more detail later; however, there are simple ways to change text sizes to permit you to add various levels of headings to your pages.

The `<hn>` (heading) tag is a block-level and container tag that encloses a string of text for display in one of six heading styles. Its general format is shown in Figure **3.5.1**.

```
<hn> Heading Text </hn>
```

The number n in the tag ranges from 1 (largest) to 6 (smallest). These six heading levels are coded in the following example and displayed in the browser as shown in Figure **3.5.1**. A standard paragraph is displayed for comparison.

```
<h1> This is Heading Level 1 <h1>
<h2> This is Heading Level 2 <h1>
<h3> This is Heading Level 3 <h1>
<h4> This is Heading Level 4 <h1>
<h5> This is Heading Level 5 <h1>
<h6> This is Heading Level 6 <h1>
<p> This is a normal text paragraph. </p>
```



Figure **3.5.1**: Browser display of `<hn>` tags. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Headings are displayed in bold characters in the default font face. They appear on a line by themselves preceded and followed by a single blank line. Thus, the `<hn>` tag cannot be used to change text sizes. In the middle the tag cannot be used to change text sizes in the middle of a line or middle of a paragraph since it forces a line break before and after its enclosed text. It is used for block-level, stand-alone text, and is most often used as content headings to identify and set apart sections of a Web page.

HTML 5 Heading Semantics

While heading tags do provide a convenient way to style text, they are designed to mark the heading for a section of content on the page. Headings with lower numbers are considered higher in rank, and content on a page should be structured with the appropriate header ranking. This listing illustrates this use of headings to section content.

```
<h1> North America </h1>
<p> North America is a large continent which contains many countries </p>
<h2> The United States of America </h2>
<p> The United States of America is a country located in North America </p>
<h3> The State of Georgia </h3>
<p> Georgia is one of fifty states found in The United States of America </p>
```

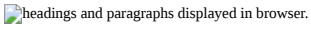


Figure **3.5.2**: Heading and paragraphs displayed in browser. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The main section, "North America", is marked with an <h1> tag to indicate that it is the start of a major section on the page. Next, "The United States of America" is marked with an <h2> tag to indicate that it begins a subsection of "North America". Finally, "The State of Georgia" is marked with an <h3> tag to indicate that it begins a subsection of "The United States of America". In this way, tags can be used to section and subsection content on a page.

Deprecated align Attribute

Headings can be aligned to the left (default) or right, or they can be centered on the line by applying the

> *align="left|center|right"*

attribute to the <hn> tag. The following code, for example, centers a heading horizontally between page margins.

> *<h1 align="center">Centered Heading>><\/h1>*

Since align is a deprecated attribute under HTML 5 standards, it should be used only as a temporary method to align headings. Later tutorials cover style sheet standards for aligning text horizontally on a page.

3.5: Headings is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

3.6: Horizontal Rules

The Horizontal Rule Element

The primary way to separate and identify sections of a Web page is by use of major and minor headings coded within tags. Paragraph level content within these sections can also can be visually separated by displaying horizontal rules between them. In addition to its visual properties, HTML 5 prescribes the horizontal rule for use when a thematic break among such content is required. The general format for this tag is shown in Figure 2-15.

> `<hr />`
>
> *Note that in HTML5 the* `/` *is optional.* `<hr>` *is fine but note that it does not have a closing* `</hr>`

The `<hr />` tag causes a line break with the rule appearing on a line by itself. The default rule is 2 pixels in height and spans the width of the browser window. The rule shown in Figure Runaway argument for \eatSpaces? is produced by the `<hr />` tag coded on a line by itself is:

> `<h2>` *Horizontal Rules* `</h2>`
>
> `<p>` *A horizontal rule is used to separate content sections of a page. The following is a default rule.* `</p>`
>
> `<hr />`
>
> `<p>` *The rule is 2 pixels in height and spans the width of the browser window. It is preceded and followed by blank lines.* `</p>`

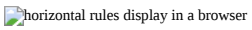


Figure Runaway argument for \eatSpaces?: Default horizontal rule. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

In a later tutorial you learn how to style a horizontal rule by adjusting its height, width, color, alignment, and display characteristics. For present purposes, the default rule can be effective in visually breaking up text without starting a new section.

Deprecated `<hr />` Attributes

Historically, the `<hr />` tag included the following attributes to style the rule. These attributes are deprecated, but many browsers still recognize them.

> *align="left|center|right"*
> *size="n"*
> *width="n|n%"*
> *noshade*
> *color="color"*

The align attribute positions the rule to the left (default), centered, or to the right across the line. The size attribute sets the height of the horizontal rule in pixels. The width attribute sets the rule's width in pixels or as a percentage of the window width. Rules can be

displayed as a solid bar by coding noshade (without an associated value) and assigned a color with the color attribute. The color value is given as a color name or hexadecimal value. We will discuss such assignments in a later tutorial.

The following rule is given by the code:

```
<hr align="center" size="5" width="50%" noshade color="red">
```

Since style attributes within the tag have been deprecated under current standards, they should not be used in HTML5 documents. Instead, they should be avoided in favor of the newer styling methods that will be discussed later.

3.6: Horizontal Rules is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

3.7: List Structures

Listing Elements

HTML 5 provides text structuring in the form of lists. You can produce lists of bulleted items, lists of numbered items, and lists of terms and definitions. The first two lists resemble single-spaced lines of text with the addition of prefixed bullets or numbers. The latter list is similar in display to a series of blockquoted paragraphs.

Unordered Lists

An **unordered list** is a series of items preceded by bullet characters and set off from surrounding text by blank lines. The list is single spaced and indented from the left margin. Unordered lists should be used when the order of each item in the list is not important. An example unordered list:

- List Item #1
- List Item #2
- List Item #3

An unordered list is created with the block-level `<ul>` container tag enclosing listed items identified with block-level `<li> <ul>` container tag enclosing listed items identified with block-level `<li>` (list item) container tags. The general format for an unordered list is:

```
<ul>
  <li> list item </li>
  <li> list item </li>
  ...
</ul>
```

For example, the bulleted list shown above is given by the following code.

```
<ul>
  <li> List Item #1 </li>
  <li> List Item #2 </li>
  <li> List Item #3 </li>
</ul>
```

Items in the list are single spaced and preceded by a bullet character. If text for a list item is wider than the width of the page it is word wrapped and indented inside the bullet character. Items can be enclosed inside `<p>` tags or `<br />` tags can be coded between items to increase line spacing between them. The following list, for example, surrounds list items with `<p>` tags to leave blank lines between entries.

```
<ul>
  <li><p> This is the first item in the list. Text follow
 character is word wrapped inside the bullet. Paragraph t
 leave blank lines between items in the list. </p></li>

  <li><p> This is the second item in the list. Text follo
```

```
  character is word wrapped inside the bullet. Paragraph t
  leave blank lines between items in the list. </p></li>
</ul>
```

Browser output for the above html is:

- *This is the first item in the list. Text following the bullet character is word wrapped inside the bullet. Paragraph tags are used to leave blank lines between items in the list.*
- *This is the second item in the list. Text following the bullet character is word wrapped inside the bullet. Paragraph tags are used to leave blank lines between items in the list.*

Nested Unordered Lists

Unordered lists can be nested inside each other. For example, a bulleted list appearing inside a second bulleted list appearing inside a third bulleted list is produced by the following code and is displayed in the browser as shown after the html code below. Note that a nested unordered list is part of the list items of its container list. That is, the `<ul>` ... `</ul>` tags for the nested list appear inside a pair of `<li>` ... `</li>` tags of `<ul>` ... `</ul>` tags for the nested list appear inside a pair of `<li>` ... `</li>` tags of the enclosing list.

```
<ul>
    <li> List Item 1 </li>
    <li> List Item 2
        <ul>
        <li> List Item 2a </li>
        <li> List Item 2b
            <ul>
            <li> ListItem 2b1 </li>
            <li> List Item 2b2 </li>
            </ul>
        </li>
        </ul>
    </li>
    <li> List Item 3 </li>
</ul>
```

- *List Item 1*
- *List Item 2*
  - *List Item 2a*

- *List Item 2b*
  - *ListItem 2b1*
  - *List Item 2b2*
- *List Item 3*

Each nested list is further indented inside its container list. Also, different bullet characters are used for nested lists. By default, a disc character marks the outer-most list, a circle marks the next inner-most list, and a square marks the inner list. Notice that when lists are contained inside other lists that no blank lines surround the interior lists as they do when a single list appears within the normal flow of text on the page.

Deprecated *type* Attribute

The type="disc|circle|square" attribute can be coded inside the opening `<ul>` tag in order to specify the style of bullet character to use if different from the default disc character. HTML5 standards do not allow the use of the type attribute and suggest using CSS for specifying bullet characters.

Ordered Lists

An ordered list is a series of items preceded by sequence numbers and set off from surrounding text by single blank lines. By default, the list is numbered with decimal numbers beginning with 1 and numbered consecutively through the last item in the list. The list is single spaced and indented from the left margin in the same way as an unordered list. Ordered lists should be used when the order of each list item is important. An example ordered list is shown below.

1. *List Item #1*
2. *List Item #2*
3. *List Item #3*

The html to create that ordered list is here:

```
<ol>
    <li> List Item #1 </li>
    <li> List Item #2 </li>
    <li> List Item #3 </li>
</ol>
```

Items in the list are single spaced and word-wrapped inside the numbering character. `<li>` can contain other tags like `<p>`, `<blockquote>`, `<h1>`, and other tags. It is important to remember that `<ol><li>` and `<ul><li>` is a structure

Nested Ordered Lists

Ordered lists can be nested inside each other, with subordinate lists indented from the next outer-most list. All nested ordered lists use the same decimal numbering system beginning with decimal 1. For example, the following code produces the numbered lists.

```
<ol>
    <li> List Item 1 </li>
    <li> List Item 2
```

```
        <ol>
        <li> List Item 2.1 </li>
        <li> List Item 2.1 </li>
        </ol>
    </li>
    <li> List Item 3 </li>
</ol>
```

1. *List Item 1*
2. *List Item 2*
    1. *List Item 2.1*
    2. *List Item 2.2*
3. *List Item 3*

Note that when ordered lists are contained inside other ordered lists that no blank lines surround the interior lists as they do when a list appears within the normal flow of page text.

Ordered List Attributes

A ***type*** attribute can be coded inside the opening `<ol>` tag in order to specify one of five different numbering characters. The attribute value can be `<ol type='1'>` for decimal numerals (the default), `<ol type='A'>` for upper-case letters, `<ol type='a'>` for lower-case letters, `<ol type='I'>` for upper-case Roman numerals, and `<ol type='i'>` for lower-case Roman numerals. The tag `<ol type='A'>`, for example, produces the following list of alphabetically ordered items.

```
<ol type='A'>
    <li> List Item 1 </li>
    <li> List Item 2 </li>
    <li> List Item 3 </li>
</ol>
```

A. *List Item 1*
B. *List Item 2*
C. *List Item 3*

You can choose the beginning sequence value of an ordered list by coding the optional `<ol start='n'>` attribute for the `<ol>` tag. Here, *n* must be a valid integer. For instance, a starting sequence value is needed, when an ordered list is interrupted by other elements on the page.

As shown in the output below, two consecutively sequenced lists are separated by a paragraph. The first list begins with "A" and is sequenced through "E" since there are five items in the list. The second list needs to begin its sequence with the sixth letter "F". If a starting value is not specified, sequencing begins anew with "A".

This is the beginning of the list:

A. List Item A
B. List Item B
C. List Item C
D. List Item D
E. List Item E

This is a continuation of the list:

F. List Item F
G. List Item G
H. List Item H
I. List Item I
J. List Item J

Code for the above lists is shown below. The first ordered list uses upper-case letters (type="A") beginning with "A" and ending with "E". The second list overrides this default sequencing by specifying start="6" in its opening `<ol>` tag. Thus, the second list is (type="A") beginning with "A" and ending with "E". The second list overrides this default sequencing by specifying start="6" in its opening `<ol>` tag. Thus, the second list is ordered consecutively from "F" through "J".

```
<p> This is the beginning of the list. </p>

<ol type='A'>
  <li> List Item A </li>
  <li> List Item B </li>
  <li> List Item C </li>
  <li> List Item D </li>
  <li> List Item E </li>
</ol> >

<p> This is a continuation of the list. </p>

<ol type='A' start='5'>
  <li> List Item F </li>
  <li> List Item G </li>
  <li> List Item H </li>
  <li> List Item I </li>
  <li> List Item J </li>
</ol> >
```

You can reverse the order of the sequence of list markers through the use of the reversed attribute. This is a boolean attribute that can be applied to an `<ol>` tag in multiple ways. For instance, you could reverse the order of markers through adding the attribute itself without a value, such as in `<ol reversed>` . Optionally, you could also add a value to the attribute, and set it

equal to either "reversed" or a blank string "". For example, `<ol reversed='reversed'>` and `<ol reversed=''>` also accomplishes the same effect. This is true of other boolean attributes you may come across in future elements. All of these forms are valid under HTML5. Below is an example of ordered list with the attribute reversed.

> 3. *List Item 1*
> 2. *List Item 2*
> 1. *List Item 3*

Definition Lists

A **definition list** is a series of terms and definitions offset from surrounding text by blank lines. The terms in the list are blocked at the left margin, and definitions are indented and word wrapped on the following lines. An example definition list is shown below with the html to create the list.

> ***Term 1***
>
> *This is the Term 1 definition. The definition term appears on a line by itself and is followed by a definition text block. The definition is indented and word wrapped.*
>
> ***Term 2***
>
> *This is the Term 2 definition. The definition term appears on a line by itself and is followed by a definition text block. The definition is indented and word wrapped.*

A definition list is enclosed inside `<dl>` tags and contains one or more `<dt>` (definition term) tags listing the items to be defined. Each definition term has an associated `<dd>` (definition description) tag enclosing the definition for the term.

The definition list shown above is produced with the following code.

```
<dl>
    <dt> Term 1 </dt>
        <dd> This is the Term 1 definition. The definitio
       a line by itself and is followed by a definition t
       The definition is indented and word wrapped. </dd>
    <dt> Term 2 </dt>
        <dd> This is the Term 2 definition. The definitio
       a line by itself and is followed by a definition t
       The definition is indented and word wrapped. </dd>
</dl>
```

When displayed in the browser, items in the list are single spaced with no blank lines appearing between the terms. If you wish to include additional line spacing, you can code `<p>` tags surrounding the definitions or `<br />` tags between them.

A definition list, of course, can be used for purposes other than defining terms. Any situation where you have a name and value relationship among content you can employ a definition list. Another example where a definition list might be appropriate is for a frequently asked questions, or FAQ, section of a web site.

3.7: List Structures is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

3.8: Linking Pages

Hyperlinks

The original purpose of the World Wide Web was to provide a mechanism for non-linear access to information. In order to activate this type of retrieval, information is packaged as a set of Web pages, and then **hypertext links**, or **hyperlinks** are created to navigate from one page to another in whatever sequence the reader prefers. In a later tutorial, the varieties of page links are discussed more fully. Here you are introduced to the basic concept of linking and shown how to set up text links for navigation between pages.

Most often a link from one page to another is made by creating a text "hot spot" on one page that, when clicked with the mouse, opens a second page which replaces the first page in the browser window. The string of text that is made sensitive to a mouse click can be a single word or several words, usually suggestive of the contents of the destination page. Normally, a second link appears on the linked page for return to the original page. It is considered poor Web citizenship to leave visitors at the end of a series of links with no way to get back to the starting page.

In the illustration in Figure 2-26 two text links cause navigation to different pages. On the linked pages a text link provides for return to the original page.

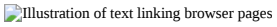


Figure **3.8.1**: Text links between pages. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

You can test these links by clicking the "Linking Web Pages" text link below. The original page is opened in a separate browser window. From there, you can link to and return from two different pages.

Linking Web Pages

The <a> Tag

The most common type of link is a clickable word or phrase that transfers directly to the target page. A text link of this sort is created by surrounding the text string with an `<a>` tag (also called an **anchor** tag), and specifying the location of the page to which to link. The `<a>` tag is classified as a container and inline-level tag. The basic format for coding this tag is:

```
<a href="url">linking text</a>
```

The `href` (hypertext reference) attribute specifies the URL of the linked page. Links can be made to other pages at the local Web site or to pages at remote sites anywhere on the World Wide Web.

If the linked page is in the same location – in the same directory – as the linking page, then the URL is simply the page name. This local link is called a *relative* link since the path to the linked page is relative to the location of the linking page.

When linking to a remote site, the URL must be a full Web reference including the `"http://"` protocol followed by the domain name of the site. This external link is called an **absolute** link since the path to the linked page is a full Web address.

The `<a>` tag surrounds a string of text that is to be made click-sensitive and that activates the link. Usually the text is descriptive of the destination link. It can be a single letter, a word, a phrase, or any amount of text that can reasonably serve as a link. When this text is clicked, the linking page is immediately replaced by the linked page in the browser window.

The `<a>` tag is an in-line tag. It does not add any new lines or spaces before or after the content which it wraps.

Local Links

When creating your own Web site, most of the links will be between your own pages. The easiest way to store these pages for linking is to save them in the same location. That is, in the same disk directory or on the same diskette or storage unit. In this case the URL specified in the `<a>` tag is simply the name of the page.

In the example in Figure 2-26 there are three Web pages named `Home.htm`, `Organize.htm`, and `Supplement.htm`. The `Home.htm` page contains text links to the other two pages and is coded as follows.

```
<h2>Linking Web Pages</h2>

<p>
  Navigation between pages of a Web site is made possible
  links. One or more words on a "linking" page are made s
  clicks that take the reader to a different "linked" pag
  reasons for breaking up a topic into separately linked
</p>

<p>
  <a href="Organize.htm">Topical Organization</a><br>
  <a href="Supplement.htm">Supplemental Information</a>
</p>

<p>
  When creating links between Web pages it is important t
  links to the linking pages. The page author should not
  at the end of a series of links with no direct way to r
</p>
```

Notice the line containing the code

```
<a href="Organize.htm">Topical Organization</a>
```

An `<a>` tag surrounds the string of text "Topical Organization" to convert it into a hyperlink to open the **Organize.htm** page in the browser window. Only the name of this page is required as the **href** URL since the linked page appears in the same directory as the **Home.htm** page from which the link is made. Similar coding links to the **Supplement.htm** page. When either of the two links is clicked, the associated page is retrieved and loaded into the browser, replacing the **Home.htm page**.

When either `Organize.htm` or `Supplement.htm` are showing in the browser window, it is easy to return to the `Home.htm` page by clicking the Back button on the browser's Tool Bar. However, it is a better navigation solution to code your own internal links between pages. In other words, both the `Organize.htm` and `Supplement.htm` pages should include, as shown here, their own links back to `Home.htm`.

Coding for the `Organize.htm` page is shown below. Here the string of text "Return to Home" is made into a link by surrounding it with an `<a>` anchor tag. A click on this link reopens the `Home.htm` page in the browser window. Similar coding appears on the `Supplement.htm` page to return to the home page.

```
<h2>Topic Organization</h2>

<p>
  One of the reasons for breaking up a Web topic into sep
  that there simply may be too much information for inclu
  page. For the same reason that books are divided into c
  are divided into sections and sections are divided into
  topic may need to be divided into logical units that ar
  Rather than being presented as a single continuous stre
  is organized into related chunks of information that ar
  overall structure and that provide for isolated reading
  subtopics.
</p>

<p><a href="Home.htm">Return to Home</a></p>
```

The manner in which you set up links between your pages is at your discretion. To a large extent, it depends on how you subdivide your site into separate pages and the way in which the separate topics relate to one another. Make sure there are no "dead end" pages where visitors arrive with no way to get back to where they came from.

Remote Links

If you know the URL of a Web site anywhere on the World Wide Web you can make a link to that site. This URL is normally in the form

`http://domain name`

where the `http` protocol prefixes the site location (domain name). This type of absolute reference takes you to the home page of that site. The following link code, for example, is used to open the home page at *www.google.com*.

```
<a href="http://www.google.com">Go to Google</a>
```

Go to Google

To find out the URL of a Web site simply go there and read the URL that appears in the Address box of your browser. Copy this address into the `href` attribute of your `<a>` tag and add the text to make a clickable link.

If you are linking to a particular page at a remote site then the domain name is followed by a path through the directory structure to the specified page. For instance, the following URL accesses a page at *sports.yahoo.com* containing links to various business sites and services.

```
<a href="http://sports.yahoo.com/nfl/teams/">NFL Teams</a
```

NFL Teams

This page is located in the `teams` subdirectory of the `nfl` directory of the site. The URL is taken from the browser's Address box for the displayed page.

Link Colors

Depending on your browser's settings, a text link shows up in different colors as a visual clue to the status of the link. Before a linked page is visited for the first time its text link displays in blue. After the page is visited, the link changes to purple. You can tell at a glance whether or not you have previously visited a page.

Unvisited Link
Visited Link

In a later tutorial you will learn how you can change these default colors for links that you code.

The target Attribute

By default, linked pages are opened in the same browser window as the linking page subsequently replacing it. When linking to a remote site, a risk is that your visitors will navigate the remote site and eventually lose contact with your site. They can repeatedly click the browser's Back button to get back to your site, but if they have wandered around the remote site to any great extent then backing up to your site may be impractical. You end up losing your visitors to the remote site.

One way around this problem is to **always** link to remote sites in a separate browser window. It is quite easy to open a page in a different window. Simply add the attribute `target="_blank"` to your `<a>` tag as shown below for the above link to the Google site.

```
<a href="http://www.google.com" target="_blank">Go to Goo
```

With this technique, visitors can browse the remote site as much as they wish in a second browser window, and by closing that window return immediately to your linking page in the original window.

Creating text links as described here represent only a couple of the ways in which hyperlinks can tie together multiple Web pages. In a later tutorial, other varieties of links and linking methods are discussed.

3.8: Linking Pages is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

3.9: Displaying Pictures

In addition to presenting text content, Web pages can display graphic images. Later tutorials discuss this topic more fully. Here, basic methods of placing pictures on a page are introduced.

Graphic Formats

There are three popular graphic formats used for Web images. **GIF** (**Graphics Interchange Format**) is the most widely supported. Pictures saved in this format have the `.gif` file extension. GIF format can display images in black and white, grayscale, or color. It is most often used for line drawings, logos, icons, and other "spot color" images, and is usually not for "continuous color" images such as photographs. Incidentally, the acronym GIF is pronounced "jiff."

The **PNG (Portable Network Graphics)** format was designed as a replacement for the GIF format. PNG, pronounced "ping," has the advantage over GIF of improving image quality while offering better image compression. It does not, however, support animated images. PNG files use the `.png` extension.

The **JPEG** (**Joint Photographic Experts Group**) format, pronounced "j-peg," is designed for storing photographic images. It can represent millions of colors, and uses compression techniques to compact all of color information into small file sizes. Pictures saved in this format usually have the `.jpg` file extension.

You can display pictures taken with digital cameras or scanned from photographs, or you can copy images from public clip art collections on the Web. If you have available any of the popular graphic editing software packages, then it is usually a simple matter to edit, resize, and convert digital photographs or scanned images to Web formats. Be sure that images are saved in GIF, PNG, or JPEG format for display on your Web pages. Other formats may display properly in the browser, but their large file sizes may make it impractical for visitors to download these images during Web access.

Capturing Images

Some of you may have the talent to create your own graphic images; most of you probably do not. Therefore, you may need to locate existing illustrations that complement the text content on your pages. The Web is a rich source of graphics, and many pictures are available for capturing. You can use your favorite search engine to locate "clip art" sites with millions of images to choose from. The screen display in Figure **3.9.1** shows a Google search finding over 32 million sources of clip art.

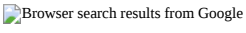


Figure **3.9.1**: Search results for clip art sites. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Once you have found the picture you want to download, it is easy to capture it for your use. As shown in Figure **3.9.2**, right-click on the picture and choose "Save Picture As..." from the pop-up menu. In the Save dialog box choose the location on your local PC to download the image. The location can be in the same directory as the Web page on which it will be displayed, or you can create a separate subdirectory for storing media files. You can also rename the picture during its saving if you wish. Be alert to images that are copyrighted. You need permission to use these.

3.9.2 Downloading an image from the Web

The <img> Tag

Image formats GIF, PNG, and JPEG are placed on Web pages by using the in-line `<img>` tag. The general format for this tag is shown below.

```
<img src="url" alt="text"/>
```

An `<img>` tag is coded on the Web page *at the location* at which the image is to be displayed. The `src` attribute gives the source, or location, of the image file. If the file is in the same directory as the Web page on which it is displayed, then only the file name is needed as the `src` URL.

The `alt` attribute is required under HTML5. Its `"text"` value gives a brief pop-up description of the image when the mouse is moved on top of the picture. This text string is displayed when users have graphic displays turned off in their browsers, thus giving a text description of missing pictures. Also, the text can be vocalized by special reader software used by visually impaired visitors who otherwise might not be able to see or focus clearly on images.

Note that the `<img>` tag is an empty tag; it does not use a pair of opening and closing tags since it does not surround any content. It only marks the page location of an image. By default, the picture is aligned at the left margin of the page.

The JPEG image shown in Figure **3.9.3** is assumed to be located in the same directory as the Web page that displays it. The image is placed on the page at the location of the `<img>` tag with the accompanying code. (Move your mouse pointer on top of the image to view its pop-up `alt` box.)

```
<p><img src="Stonehenge.jpg" alt="Picture of Stonehenge"/></p>
```

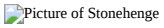


Figure **3.9.3** Code and browser output of image displayed with `<img>` tag.

Notice that you do not physically place the image on the Web page. Rather, you include an `<img>` tag containing a URL address for the image file. When the Web page is loaded into the browser this URL is used to locate the image file, which is then transmitted separately to the browser for display on the page. This is why image file sizes are important considerations. Displaying an image requires an additional data transmission from the Web server – one for each image appearing on the page. Therefore, try to keep the number of images and their sizes as small as possible in order to reduce the number and length of transmissions needed to fill a Web page.

More information about placing images on Web pages is presented in later tutorials. For now, you should be able to modestly decorate your pages with your own pictures or those copied from clip art collections.

Using Graphics as Links

Graphic images can be used as links in the same way as text strings. In this case, an `<img>` tag is surrounded by an `<a>` tag to make the image click sensitive and to link to the page or site coded in the `href` attribute.

The following image was copied from the *www.google.com* Web site and saved to a local Web directory. It is placed on the page and made into a clickable link to the site that opens in a separate browser window.

```
<a href="http://www.google.com"><img src="Google.gif" alt
```



Notice that the above image is surrounded by a border. In much the same way as text strings used as links are underlined, graphic images used as links are displayed with a border. The border is a visual indication that the picture is a link. You can eliminate the border with the deprecated attribute `border="0"` coded in the `<img>` tag. More current techniques for working with borders are described in a later tutorial.

Creating a Media Subdirectory

As you produce more and more Web pages, you will get to the point where your Web directory starts to become overloaded and disorganized. Preferably, documents and picture files that relate to different Web sites or applications should be kept together inside their own separate subdirectories. This arrangement makes it easier to locate and work with related files.

One of the common ways to begin this organization is to create a separate subdirectory within the main Web directory just for your graphic files. In this way, all of your Web pages remain in the main Web directory and all supporting files reside in the subdirectory. Figure **3.9.4** shows a typical arrangement of Web directories, with a `Media` subfolder inside the main Web folder.



Figure **3.9.4**: Web directory structure. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

If you create this `Media` subfolder, then the relative URL for the `src` attribute of `<img>` tags needs to reflect this directory structure. For example, if `Page1.htm` in the accompanying illustration displays `Pic1.gif` located in the `Media` subfolder, then its `<img>` tag needs to be coded as:

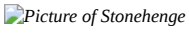
```
<img src="Media/Pic1.gif" alt="Image Pic1"/>
```

This coding indicates that, relative to the directory location of `Page1.htm`, the `Pic1.gif` image is located one level down in the directory hierarchy, inside the `Media` subdirectory. In a later tutorial, the organization of Web pages and supporting files is covered in more detail. At present, you can begin to bring order to your Web files by placing images in a separate subdirectory.

Deprecated Image Alignment

Horizontal placement of an image can be controlled by enclosing the `<img>` tag inside a `<p>` tag which aligns the image to the left, to the right, or centered on the line. Paragraph alignment can use the deprecated `align="left|center|right"` attribute of the `<p>` tag. The picture below, for example, is centered on a line by itself and is preceded and followed by blank lines with the enclosing `<p>` tag.

```
<p align="center"><img src="Stonehenge.jpg" alt="Picture
```



Later you will learn more about placing and aligning images on a page. For present purposes consider use of the paragraph's `align` attribute only as a quick fix for aligning images.

Image width and height Attributes

A downloaded image may not be the exact size you need for your page. The best solution is to open it in a graphic editing program in order to size or crop the picture to the dimensions you need prior to placing it on the page. An alternative is to resize the picture dynamically, during browser display, by setting the `<img>` tag's **`width`** and **`height`** attributes:

```
width="n"
height="n"
```

These *n* sizes are given in pixels. You need only to set one *or* the other dimension, but not both, to maintain proportionality. The browser automatically adjusts the other dimension to keep the picture in correct proportion.

The original size of the downloaded image shown below is 480 pixels (width) by 641 pixels (height). It is resized during browser display with the tag:

```
<img src="Flower.jpg" alt="Picture of flower" width="200"
```

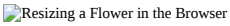


Figure **3.9.5**: Image resizes with browser window. (Copyright; author via source) ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

By resetting the display width only, the browser adjusts the height setting to ensure that proper proportions are maintained. Even though the image is resized by the browser, the original image file size remains the same. For large images, this can mean excessive time to display an image since the entire file must be transmitted to the visitor's computer prior to resizing it on-the-fly for screen display.

3.9: Displaying Pictures is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

4: Basic Document Styling

4.1: Applying Styles
4.2: Margin Styles
4.3: Alignment Styles
4.4: Width and Height Styles
4.5: Basic Color Styles
4.6: Basic Color Styles
4.7: ID and Class Selectors

4: Basic Document Styling is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

4.1: Applying Styles

A large part of HTML5 coding involves the application of styling characteristics to page elements. Styling includes the positioning of elements on the page along with such visual appearances as font faces, sizes, colors, and other display characteristics. There are two basic ways to control the layout and appearance of page elements: through tag attributes and through style sheets. These methods are described briefly here and are elaborated and illustrated throughout the tutorial.

Attributes

Formatting styles can be applied through attributes coded within HTML tags. Particular tags have particular attributes that are specific to them and that control the appearance of whatever page content they contain. As a quick example, a horizontal rule is displayed across the page wherever the `<hr>` tag is coded. This tag produces the following line that is often used to separate paragraph level content on page. (the default color is black, but we used blue to ensure it would standout)

Figure **4.1.1**: Default Horizontal Rule. (("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0))

The above line is the default display of the tag. It can take on a different appearance by coding attributes inside the tag. The size attribute controls the depth of the line; the width attribute gives its length; the color attribute sets its color, and the align attribute specifies its horizontal position on the line (to the left of the page, to the right of the page, or centered on the page).

Tag attributes are coded using the inline style method (described below) and the "style" tag. The style tag uses CSS properties in HTML to set such things as color, size, background changes, height and width. Below is an `<hr>` tag coded with four attributes to control its displayed appearance as shown in Figure **4.1.2**.



Figure **4.1.2**: Red horizontal rule 50% of the width of the Web page. (("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0))

The line is 10 pixels in height ("height:10px;"), 50% of the width of the browser window ("width: 50%;"), red in color ("background-color:red"), and centered horizontally on the page (align="center").

```
<hr style="height:10px; align: center; width: 50%; background-color:red" >
```

Thus, using the style tag with attribute values give styling characteristics to display the rule differently from its normal default style. As you proceed through this tutorial, particular attributes associated with particular tags are presented as styling options for those tags.

Under HTML5 standards, the style tag is encourage instead of the older general format of: attribute="value", which has been deprecated. The movement is towards use of Cascading Style Sheets, rather than setting attributes directly within HTML5 elements.

Cascading Style Sheets

The preferred way to apply styling to page elements is with Cascading Style Sheets (CSS). A style sheet is a set of style properties and accompanying style values that describe the appearance of HTML5 elements to which they are applied. There are three methods of creating style sheets: an in-line style sheet appears inside the tag to which its style declarations apply; an embedded style sheet is a separate style section of a Web page which applies its styles to all designated tags on a page; and a linked style sheet is an external document containing style settings that apply to all pages that link to it.

In-line Style Sheets

in-line style sheet is coded inside a tag to apply styling to that particular tag. It is coded in the general format:

```
<tag style="property: value [; Property:value] ..."> </tag>
```

One or more property:value pairs give style settings within a style attribute coded in the tag. Multiple style properties settings are separated from each other with semicolons, and spaces can be included between the settings to help with readability.

To some extent, the properties and values that can be coded for a tag depend on the particular tag being styled. For instance, style settings for a horizontal rule can include height, width, color, and text-align properties, identical in effect to the deprecated size, width, color, and align attributes they replace. These properties are coded as an in-line style sheet as shown in Figure **4.1.2** above. A style attribute appears inside the `<hr>` tag and specifies a set of properties and values to apply to the tag when it is displayed in the browser.

When applied as in-line style sheets, the properties and values pertain only to the tag in which they are coded. This means that if there are several `<hr>` tags on a page, each would need to include the same in-line style sheet to share the same style. This may not present a problem with only a few tags; however, with many tags it can be tedious and error-prone to code the same style sheet in all of the individual tags. A way is needed to declare style settings one time and to share these settings among multiple tags.

Embedded Style Sheets

To avoid duplicate coding of in-line style sheets, an embedded style sheet can be used. An embedded style sheet is defined within a `<style>` tag, normally coded in the `<head>` section of the page. Within this `<style>` section is a list of tag names, called selectors, to which style declarations (properties and values) are assigned. Once these declarations are made, they are automatically applied to the identified tags wherever they are located on the page.

Older HTML languages required the additional `<style>` tag attribute type="text/css" to be included in order to specify the type of content surrounded by `<style>` tags; however, in HTML5 the type automatically defaults to "text/css" when omitted, making it unnecessary to include. The general format for the `<style>` section of a document is shown in the code block below.

```
<style> selector {Property:value [; Property:value] ...}
   ...
</style>
```

A selector is a tag name (without the enclosing `<` and `>` symbols). Style properties and values for the tag are enclosed within a list of style declarations, separated by semicolons, and all enclosed within a pair of braces `{ }` . For example, the following code specifies style declarations for the `<hr>` tag by supplying the same style settings as in the previous in-line style sheet.

```
<head>
    <title> Embedded Style Sheet Example </title>

    <style>
       hr { height:10px; width:50%; color:red; text-align:center; }
    </style>
</head>
```

The hr selector is associated with four property:value declarations to style a horizontal rule. Once these styles are defined within an embedded style sheet, they are applied automatically wherever the browser encounters an `<hr>` tag in the document. It is not necessary to code separate in-line style sheets within each and every `<hr>` tag. The tag itself carries the styling described in the embedded style sheet.

Although shown above with a single selector for the `<hr>` tag, an embedded style sheet can contain any number of entries depending on how many different tags are to be styled. The example is also coded on a single line. Many Web page authors prefer to code property:value settings on separate lines for ease of reading and editing as shown here:

```
<style>
     hr {
```

```
            height:10px;
            width:50%;
            color:red;
            text-align:center;
          }
    </style>
```

The typed format of style sheet entries does not affect their styling. Like HTML code, style sheet code is in free format: it can be arranged in any fashion as long as the codes and punctuation are correct.

Whether to employ an embedded style sheet or multiple in-line style sheets is a matter of coding efficiencies and personal choice. Normally, if a style is applied to multiple occurrences of a tag it involves less coding and fewer errors to code an embedded style sheet by declaring the style one time for sharing by all designated tags. One the other hand, if a style is applied only once to a single tag, then there is convenience in coding an in-line style sheet for that one particular tag. Later, strategies are presented for dealing with various styling situations.

Linked Style Sheets

If your Web site contains multiple pages,then a third style sheet alternative is probably the best solution. It permits you to conveniently apply the same styles to all pages without having to duplicate in-line or embedded style sheets on each page.

A linked style sheet is a separate document containing property:value settings in the same format as an embedded style sheet. The only difference is that the linked document does not include `<style>` tags surrounding the entries. For example, a linked style sheet document containing specifications for the `<hr>` tag described above includes the entries shown here:

```
    Stylesheet.css (document)

    hr {
        height:10px;
        width:50%;
        color:red;
        text-align:center
    }
```

This separate document is created with a text editor and saved as a standard text file, usually with the file extension .css to identify it as a style sheet document. It is a common practice to save the document under the name Stylesheet.css in the same directory as the Web pages to which it applies.

A linked style sheet document contains style settings that are applicable to all pages of the Web site. Therefore, all pages that use the style sheet must "link" to it to make the styles available to those pages. This linkage occurs with a `<link>` tag coded in the `<head>` section of the page. The general format for the `<link>` tag is

```
<link
  href="url"
  rel="stylesheet"
>
```

The href (hypertext reference) attribute gives the location of the linked style sheet. If the style sheet appears in the same directory as the Web page to which it applies, then this entry is simply the name of that document. The rel attribute specifies the relationship of the external document to the current page (always "stylesheet"). Just as with the `<style>` tag, the attribute type="text/css" historically was necessary to include in order to tell the browser what type of resources was being linked. Under HTML5, when the type tag is omitted, it automatically defaults to "text/css", making its inclusion unnecessary.

If you have a style sheet document named Stylesheet.css that is located in the same folder as your Web page. You link the Web page to this document using the following HTML code.

```
<head>
    <title> Linked Style Sheet Example <title>
    <link href="Stylesheet.css" rel="stylesheet">
</head>
```

Now the Web page has available to it all of the style settings included in the Stylesheet.css document. This linked style sheet can serve to replace embedded or in-line style sheets that otherwise would appear on individual Web pages. As in the case with embedded style sheets, any tags identified with selectors in a linked style sheet carry with them the declared styles.

Style Sheet Comments

It is usually a good idea to place comments in your CSS document to describe its various sections. Comments are general descriptions or explanations of CSS code. These serve as useful reminders of the purposes or contents of sections of code when you or someone else returns at a later time to edit the document.

CSS comments are coded between a /* and */. Comments can span multiple lines. The example below demonstrates the use of CSS comments.

```
/* Set Page Font Family */

body {font-family:arial}

/* You can also comment an entire block during testing

.comment {font-weight:bold}

*/
```

Besides their use for including comments in a CSS document, comment tags can be used to temporarily suspend browser display of a section of code. This purpose is often useful in "debugging" your page, that is, in trying to discover errors by selectively removing sections of code from display until the problem is isolated.

Applying Style Sheets

It would not be uncommon to employ all three types of style sheets for a single Web page. A linked style sheet would contain styles that are common to all Web pages; an embedded style sheet would contains styles that are pertinent to the particular page; and in-line style sheets would apply to individual tags for which common styling is not needed. The browser handles these multiple style sheets in the following fashion.

- First, any linked style sheets are applied to the identified tag selectors on the page.
- Second, any embedded style sheets are applied. If the same tag selectors appear in both the linked and embedded style sheets then embedded styles override or augment linked styles.
- Third, any in-line style sheets are applied. If these settings pertain to the same tags that appear in either linked or embedded style sheets, then in-line styles override or augment both linked and embedded styles.
- The general principle in force is that any lower-level style setting takes precedence over an equivalent higher-level style setting. In-line style sheets take precedence over embedded style sheets which, in turn, take precedence over linked style sheets.

Assume, for example, that a linked style sheet contains the following style declarations for horizontal rules.

```
hr {
        height:1px;
        width:50%;
        color:red;
        text-align:center;
    }
```

All pages that link to this style sheet display horizontal rules in this style. Now assume that one particular page needs differently styled rules, say, ones that are blue rather than red. An embedded style sheet is coded on this single page in order to override the color declaration in the linked style sheet:

```
<style>
   hr {color:blue;}
</style>
```

All horizontal rules on this page are blue; however, they retain the height, width, and text-align properties from the linked style sheet. Tags on this page inherit the properties of the linked style sheet unless modified by the embedded style sheet.

Assume further that one particular horizontal rule on this page needs to be green and extend to the width of the page. An in-line style sheet is added to this particular rule.

```
     <hr style="color:green; width:100%;">
```

Green styling overrides the blue color that is inherited from the embedded style sheet, and a width of 100% overrides the 50% width inherited from the linked style sheet. This particular rule, however, retains its 1 pixel height and centered alignment inherited from the linked style sheet.

Style inheritance is what makes style sheets "cascading." Linked styles are inherited by, or cascade across, all Web pages that link to them; embedded styles are inherited by, or cascade across, an entire Web page containing a `<style>` section; and in-line styles are inherited by, or cascade across, individual tags containing in-line style sheets.

Throughout these tutorials, style properties and values are introduced and illustrated as in-line style sheets with discussion on how they can be elevated to embedded on linked style sheets for wider application. By the end, you should be at ease with style sheets and with the many ways they offer to design Web pages to meet your most exacting design requirements.

Validating CSS

The W3C maintains a free CSS Validation Service available at http://jigsaw.w3.org/css-validator that can be used to validate CSS code and check it for syntax errors. The CSS validation tool can help identify code that needs to be corrected quickly and indicate which style rules a browser likely to consider valid.

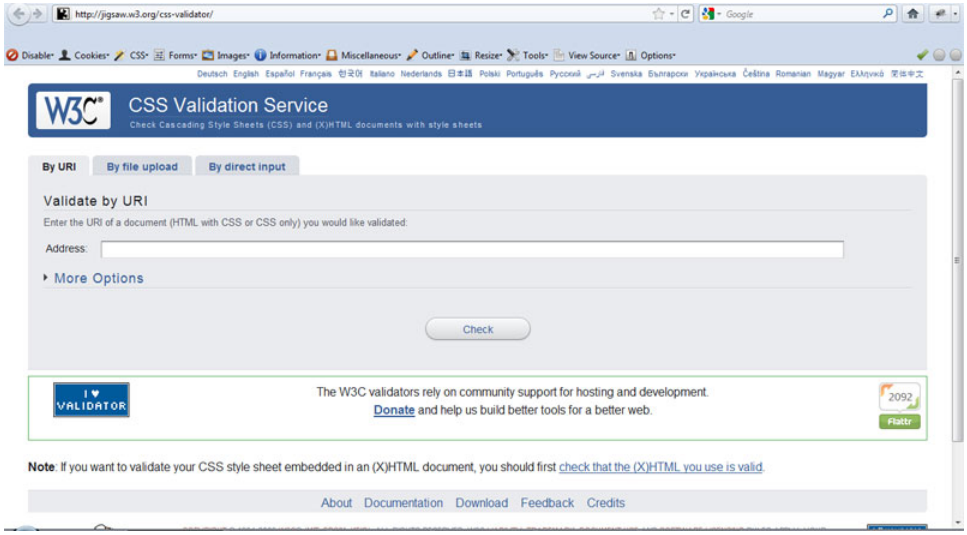


Figure **4.1.3**: W3C CSS Validation Service. (("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0))

Figure \(\PageIndex{5}\): W3C CSS Validation Service. (("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0))

There are three options for validating CSS.

1. You can enter the URL for the external or linked .css file.
2. You can upload a CSS file from your local PC.
3. You can paste your CSS into a text box.

4.1: Applying Styles is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

4.2: Margin Styles

Margin Styles

When text is displayed in the browser window it spans from border to border with about a quarter of an inch margin surrounding the text on the page. You may have noticed these margins on Web pages created earlier. They are shown in the simple single-paragraph Web page in Figure **4.2.1**.

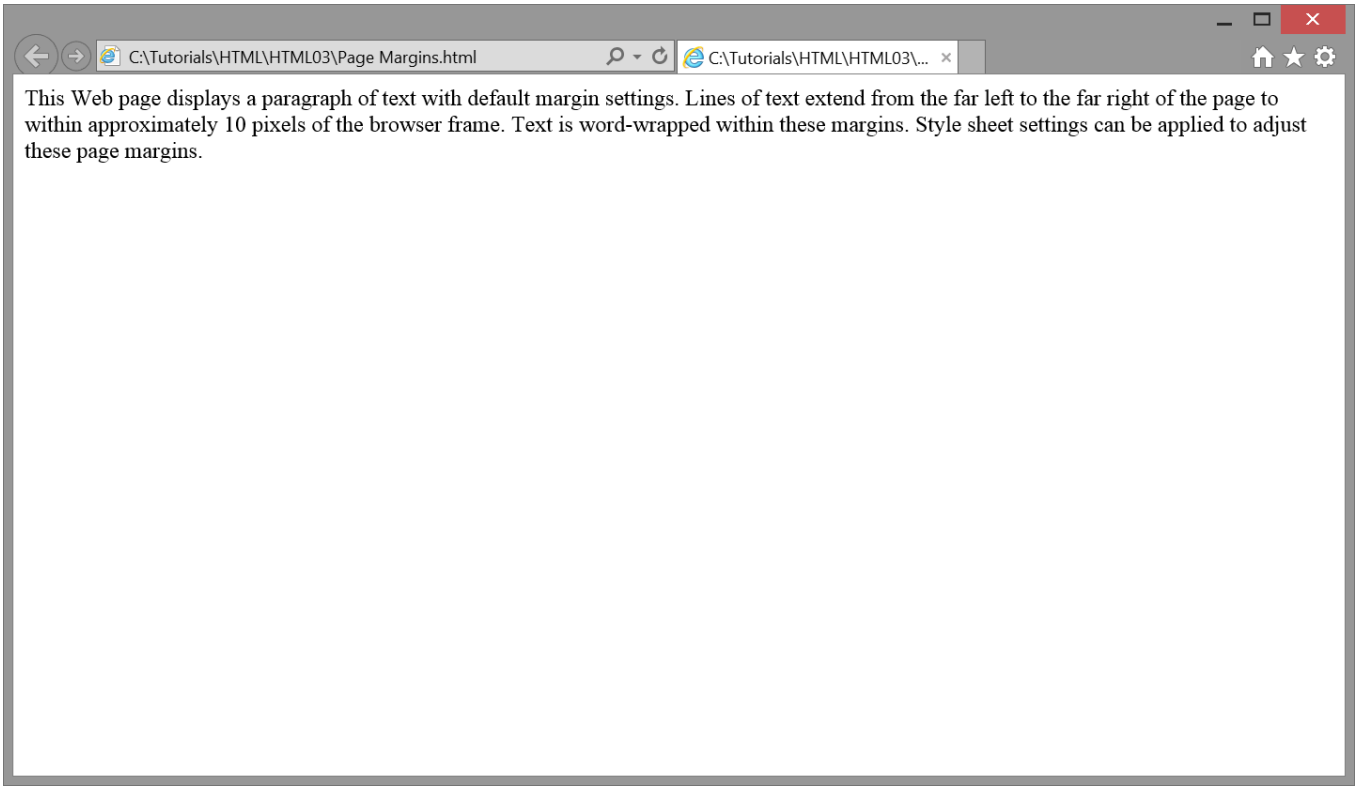


Figure **4.2.1**: Web page with default margins. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Margin Style Properties

There are five style properties that can be used to set margins around page elements. These properties specify the amount of space to leave between the outside edges of the element and any surrounding content. When applied to the `<body>` tag, they give the amount of white space that needs to be left around any content displayed on the page.

Property	:	Values
margin margin-top margin-right margin-bottom margin-left	:	*n*px (pixels) - A pixel represents a single dot on a monitor. The number of dots per inch depends on the screen resolution. *n*% (percent) - A percent specifies a value relative to the current value. *n*em (ems) - One em is equal to the font-size for the current font. *n*pt (points) - A point is 1/72 of an inch. auto

Figure **4.2.2**: Margin properties and values. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The **margin** property sets the same margin width around *all* sides of the element; the **margin-top, margin-right, margin-bottom,** and **margin-left** properties set margins individually for each of the four sides. The **auto** setting causes margins to revert to

their default widths when overriding previous margin settings.

Measurement Values

The margin properties, like various other style properties, require a measurement value. In most cases, you are required to specify whether the measurement is in pixels (**px**), a percentage (**%**) of the width of the browser window, an em (**em**), or point (**pt**). If you do not specify the measurement type, then pixels are assumed.

Applying Margins with an In-Line Style Sheet

You can set margins surrounding the content on a Web page by applying a style sheet to the `<body>` tag. For example, the following in-line style sheet sets a 25-pixel margin around the entire page by using the margin property, and by specifying 25 pixels of white space between the contents on the page and the borders of the browser window.

```
<body style="margin:25px;">

<p>  This Web page displays a paragraph of text within margin settings of
25 pixels surrounding the page. These margins are produced by an in-line
style sheet applied to the body of the document. Lines of text extend
between and are word-wrapped within these left, right, top, and bottom
margin settings. </p>

</body>
```

This styling produces the page display shown in Figure **4.2.3**. The text paragraph appears less crowded than in Figure **4.2.1** which makes it a more inviting read. Shortening the lines makes the text more readable as well.

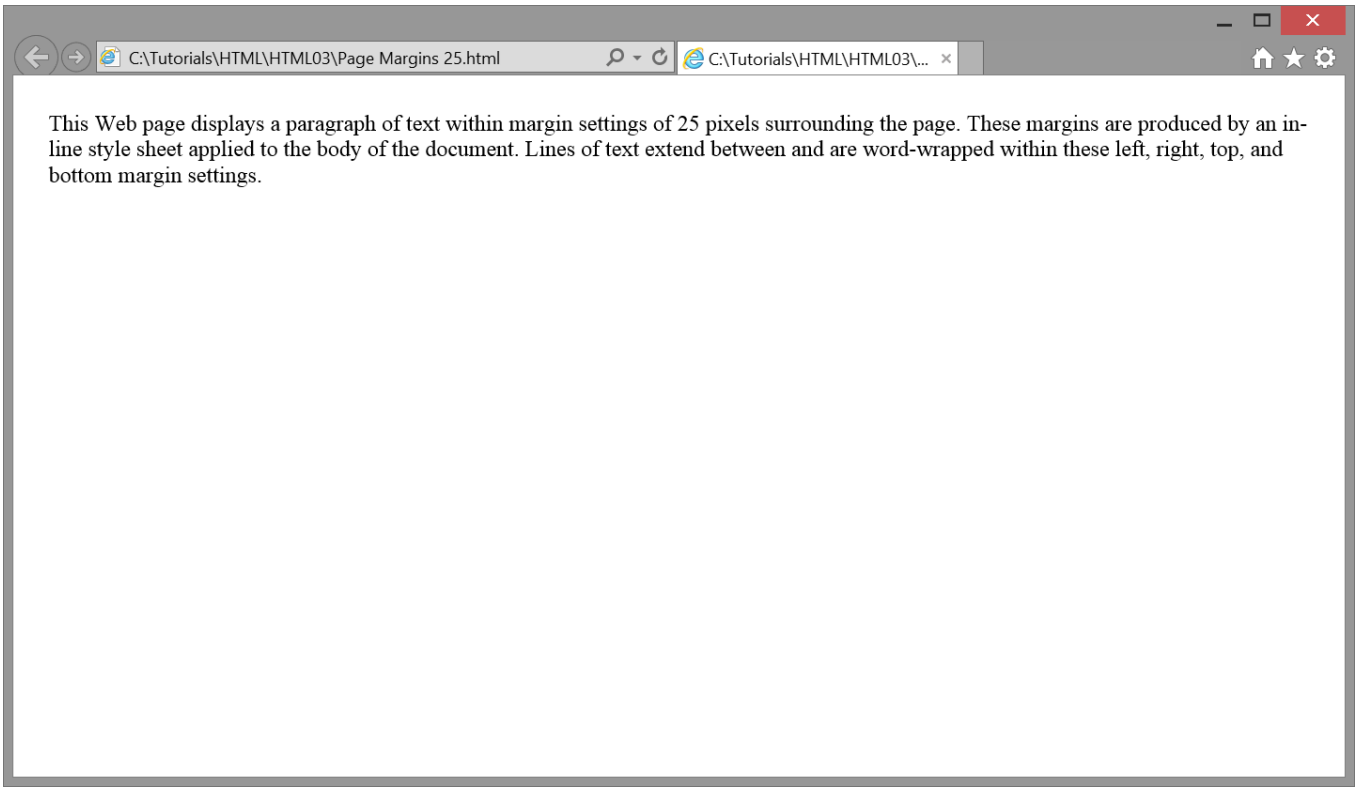


Figure **4.2.3**: Web page with 25-pixel margins surrounding the page. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Applying Margins with an Embedded Style Sheet

In the above example, the `<body>` tag includes an in-line style sheet to set margins for this particular page. Instead, you can code an embedded style sheet to produce the same results. The following `<style>` entry supplies a body selector along with a margin style property and value to apply to this tag.

```
<head>
    <title>  Page Margins </title>
    <style>
       body {margin:25px;}
    </style>
</head>

<body>

    <p>  This Web page displays a paragraph of text within margin settings of
   25 pixels surrounding the page. These margins are produced by an embedded
   style sheet applied to the body of the document. Lines of text extend
   between and are word-wrapped within these left, right, top, and bottom
   margin settings. </p>

</body>
```

The `<body>` tag, through the body selector, takes on the 25-pixel margin setting without having to code a style sheet inside the tag itself. Furthermore, by using an embedded style sheet, style settings are isolated within a separate section of the page which allows styles to be easily located and changed if necessary. In fact, your first instinct, should be to create an embedded style sheet adjusted to in-line or linked style sheets as situations dictate.

Applying Margins with a Linked Style Sheet

If you have a Web site with multiple pages and all pages need to share the same body margins, then a linked style sheet should be used. Place body styling in this separate document and link to it from all pages to which the style should apply.

The following code shows a Stylesheet.css document in which margin styling for the `<body>` tag is specified. This document appears in the same Web directory as the pages which share this style.

```
Stylesheet.css

       body {margin:25px;}
```

Once this style sheet document has been created any Web page can apply its margin settings by linking to it. Code to adopt this linked style sheet is shown below.

```
<head>
    <title> Any Page </title>
    <link  href="Stylesheet.css" rel="stylesheet" >
</head>
...
```

Applying Individual Page Margins

In the above examples, the margin property is applied to set the same margin widths surrounding all four side of the page. Instead, you can selectively apply different widths to each of the sides by using individual margin properties. In the following embedded style sheet these different margin settings are applied to a page.

```
<style>
     body {margin-top:50px; margin-left:30px; margin-right:30px; margin-bottom:50px;}
</style>
```

Note that the properties can appear in any order without affecting their application. Be sure that property:value pairs are connected with a colon (:) and that declarations are separated with semicolons (;).

The single margin property also provides a quick way to specify individual margins. The previous example could have been written using just the margin property in the following way

```
<style>
       body { margin: 50px 30px 50px 30px; }
</style>
```

With this shortcut method the order of values is important. Each length corresponds to the top, right, bottom, and left margins, in that order. Providing the margin with just two values also provides a neat shortcut. The previous code can further be reduced simply to:

```
<style>
       body { margin: 50px 30px; }
</style>
```

Here, the top and bottom margins will have a length of 50px, and the left and right margins will have a length of 30px. Again, the order matters for this shortcut to correctly work.

Vertical Margins

It was mentioned previously that multiple `<br>` tags can be used to increase the amount of vertical spacing on a page. Each `<br>` tag beyond the one used to end a line of text produces an additional blank line down the page.

This same effect can be achieved with margin-top and margin-bottom style settings. By specifying these margin widths at the top and bottom of a text block, the given number of pixels of blank space are inserted before and after the text, and produce an effect similar to inserting blank lines. For instance, the following code inserts additional spacing before and after an indented paragraph by increasing its top and bottom margins with an in-line style sheet. The effect is the same as shown previously for a blockquoted paragraph offset with additional `<br>` tags. Browser output is shown in Figure **4.2.4**.

```
 <p> Here is a tale about Mary and a pesky little lamb which followed her
anywhere and everywhere she went. </p>

     <p  style="margin-left:40px; margin-right:40px;
            margin-top:50px; margin-bottom:50px;" >
    Mary had a little lamb, <br>
    Its fleece was white as snow; <br>
    And everywhere that Mary went, <br>
    The lamb was sure to go. <br>
     </p>
```

```
<p> Mary had an awkward social life. It's awfully difficult to date with sheep
trailing around after you all the time. </p>
```

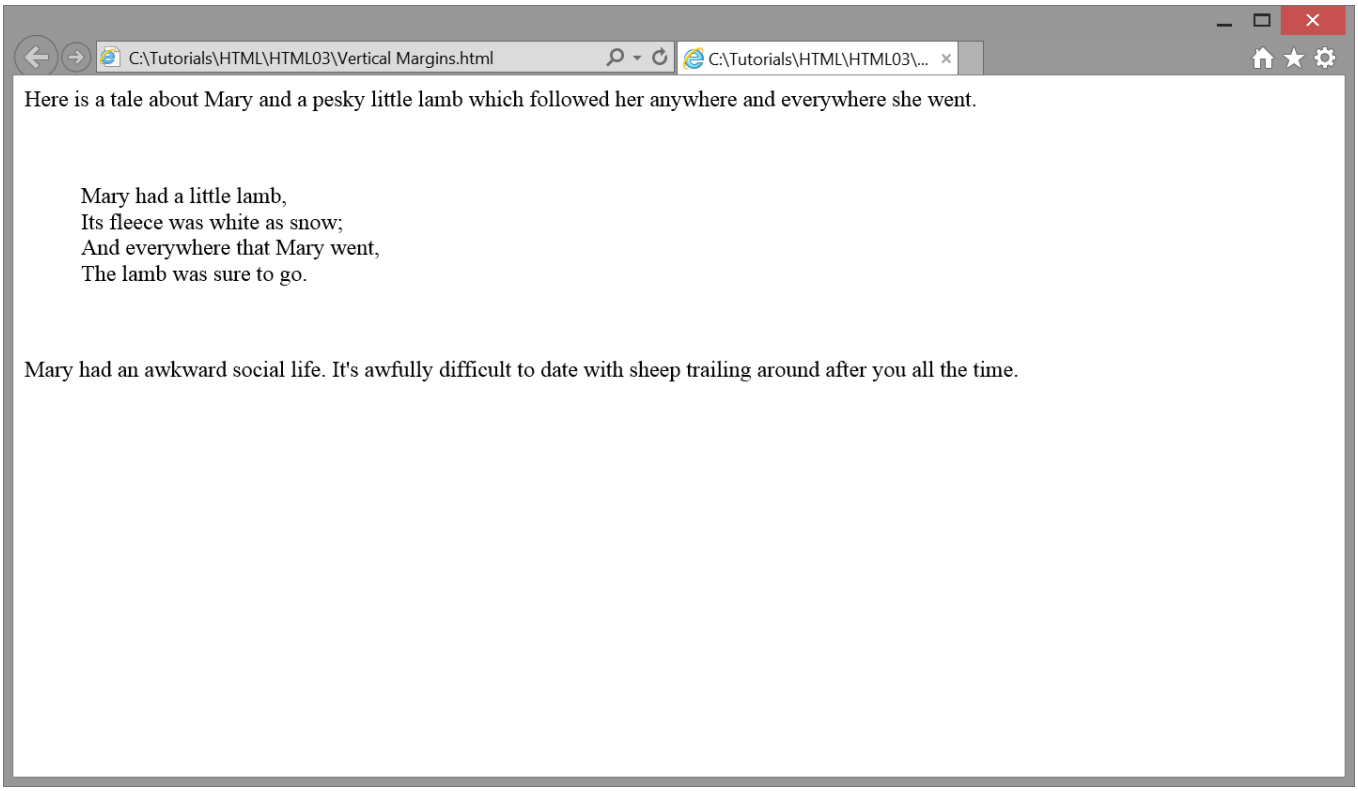


Figure **4.2.4**: Using margin settings to increase vertical spacing. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

While it is possible to use either multiple `<br>` tags or extended margin settings to produce vertical spacing, HTML5 guidelines specify that the
tag should only be used to provide line breaks among content, not simply to generate extra blank vertical space. Moreover, greater pixel precision is gained by applying margin-top and margin-bottom styles.

As you encounter additional tags throughout these tutorials, keep in mind that they can be assigned margin properties to adjust the amount of space between their contents and surrounding elements on the page. Margin settings are one of the main techniques used to reduce the crowding of content on a Web page. There will be a recurring need to apply margin properties to all manner of HTML tags.

Paragraph Margins

Margin styles are applicable to many HTML tags, not just to the `<body>` tag. In fact, any container tag can take on margin settings. The `<p>` tag is case in point. By declaring margin, margin-top, margin-right, margin-bottom, and/or margin-left style properties for this tag, a paragraph can have its margins adjusted on all sides of the text block or on each side individually.

The following code displays three paragraphs, the middle one of which uses an in-line style sheet to set its left and right margins to 40 pixels.

```
<p> Here are three paragraphs. This first paragraph is formatted with a
standard paragraph tag with default style settings. </p>
```

```
 <p  style="margin-left:40px; margin-right:40px;" > This second paragraph is
formatted with the style setting "margin-left:40px; margin-right:40px;" to
adjust its left and right margins to offset the paragraph from surrounding
paragraphs. This styling produces a paragraph similar in style to a
blockquote tag. </p>

 <p> This third paragraph is coded like the first one. It is enclosed within
a standard paragraph tag containing default styles. </p>
```

Browser output of this code is shown in Figure **4.2.5**.

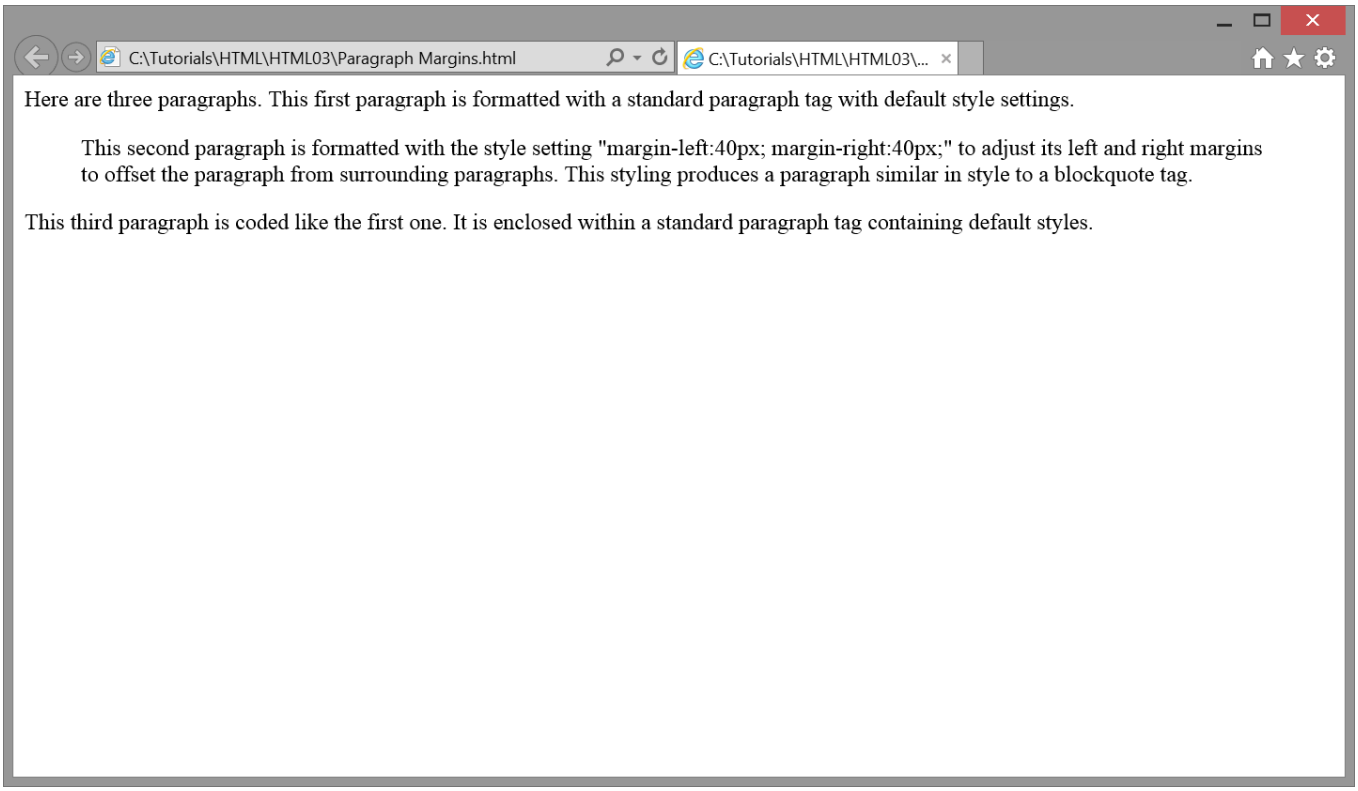


Figure **4.2.5**: Copy and Paste Caption here. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

An in-line style sheet is used in this example since margin settings are applicable only to the single paragraph. There is no need to define an embedded style sheet or to create a linked style sheet document since this styling is not shared across all paragraphs.

4.2: Margin Styles is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

4.3: Alignment Styles

Elements on a Web page are normally blocked at the left margin of the page or at the left margins established by their margin or margin-left style properties. It is often the case, that you may wish to change this default alignment, and align a page element at the right margin, center it horizontally on the page, or "float" the element to the left or right to permit text flow around the sides of the element.

The text-align Style

The text-align style property permits left, center, right, or justified alignment of text within elements. Its property settings are shown in Figure 4.3.1.

Property	:	Values
text-align		left center right justify

Figure 4.3.1: Text alignment property and values. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The text-align style property is replacement for the deprecated align attribute previously used for `<p>` , `<hn>` , and `<hr>` tags. From this point forward, all horizontal alignment of text should be done with the text-align property.

Paragraph Alignment

A `<p>` tag surrounds a block of text which has its individual lines blocked at the left margin of the page or at the left margin established by the paragraph's margin or margin-left style setting. A text-align style can be applied to the paragraph to change this default left-alignment display.

The following code creates four paragraphs that each have an in-line style sheet to apply different text alignments. Browser display of these paragraphs shown here:

```
<p style="text-align:left;">
   This paragraph is formatted using the style="text-align:left;" style
   setting. Each line of text is blocked at the left margin of the page and
   wraps at the right margin of the page. This is the default alignment
   style for a paragraph. </p>

    <p style="text-align:center;">
   This paragraph is formatted using the style="text-align:center;" style
   setting. Each line of wrapped text appears centered between the left and
   right page margins. </p>

    <p style="text-align:right;">
   This paragraph is formatted using the style="text-align:right;" style
   setting. Each wrapped line of text is aligned at the right margin of the
   page. </p>

    <p style="text-align:justify;">
   This paragraph is formatted using the style="text-align:justify;" style
   setting. All wrapped lines of text except the last line are expanded to reach
```

```
    both left and right margins. Since the final line is not long enough to
    wrap, it is not justified between the margins of the page. </p>
```

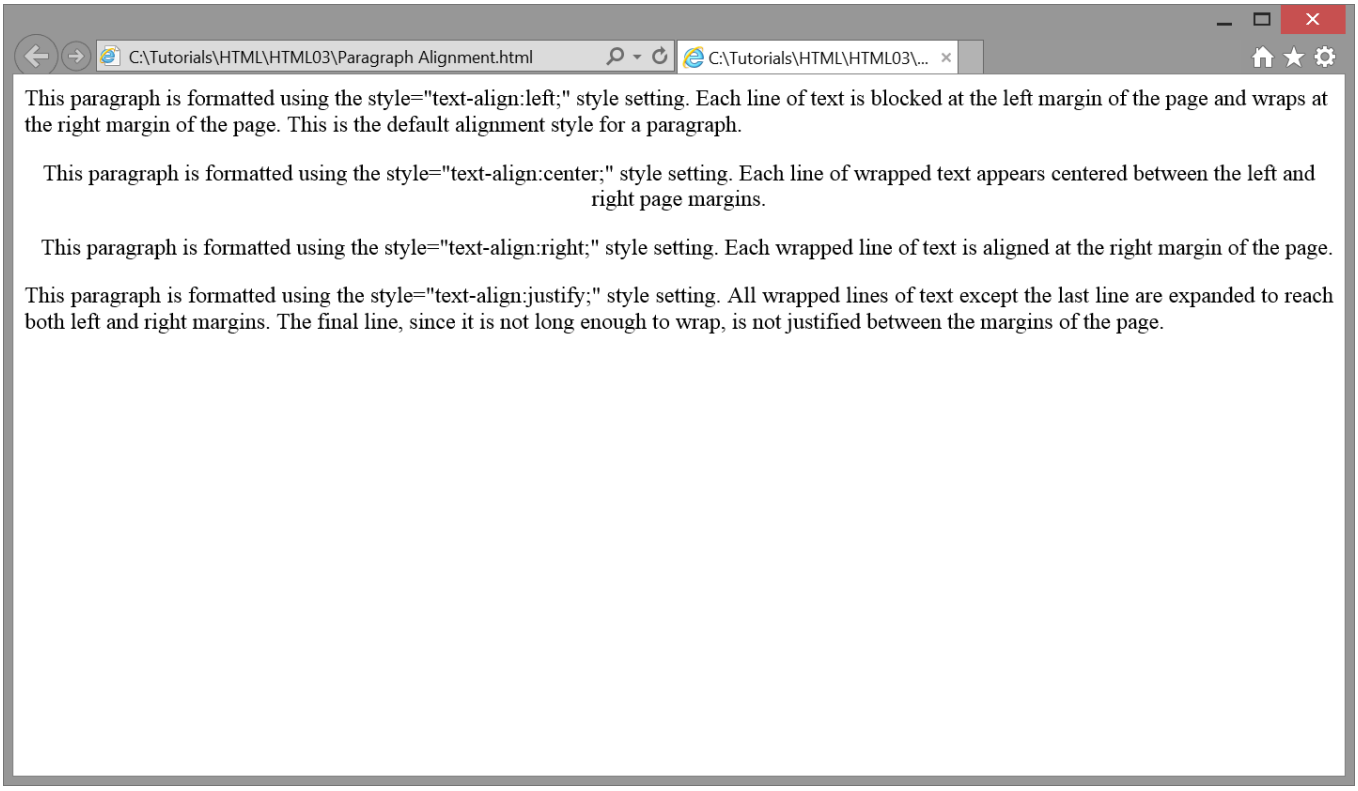


Figure **4.3.2**: Various paragraph alignments displayed in browser window. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

In this example, all four paragraphs have a different styling. Thus, `<p>` tags are styled with in-line style sheets rather than with an embedded or linked style sheet which would produce common styling for all paragraphs. Keep in mind, that text alignment which should be shared by *all* paragraphs on a page should appear one time only in an embedded or linked style sheet. If it is decided that all paragraphs on a page will have justified lines, then an embedded style declaration for the `p` selector is appropriate.

```
<style>
     p {text-align:justify;}
    </style>
```

If it is further decided that all paragraphs on all pages of a Web site will have justified paragraphs, then this embedded style sheet can be promoted to a linked style sheet for sharing among all pages.

Heading Alignment

Text alignment styles can be applied to `<hn>` tags to place headings at the left, in the center, or to the right of a page. In the following example, the text-align property is applied to an `<h2>` heading with an in-line style sheet to center it horizontally on the page. This style property replaces the older align="center" attribute previously used to align headings.

```
<h2 style="text-align:center;"> Centered Heading <h2>
```

It is normally the case that various headings used on a Web page will have common styles in order to enforce a common look for the page. Therefore, it is common practice to include heading alignment in an embedded style sheet for sharing among all like headings. The following code gives alignment styling for three heading sizes that appear on a page. Furthermore, top margins are increased to leave additional vertical blank space between the headings and preceding text.

```
<style>
     h1 {text-align:center; margin-top:20px;}
     h2 {text-align:left;   margin-top:20px;}
     h3 {text-align:left;   margin-top:20px;}
</style>
```

Although text alignment is not required for the above `<h2>` and `<h3>` tags (since left is the default alignment), these settings are included in the style sheet to make them explicit and to documentation the settings. Also, since all heading alignments appear in this embedded style sheet, any changes to heading alignments can be made at this single location for propagation across all headings on the page.

Centering Images

Image alignment takes place by first converting the image from an inline element to a block-level element using the CSS `display` property, and setting its value to block. Next, the CSS properties `margin-left` and `margin-right` styles are applied with a value set to `auto`. For example, coding to center an image on the page is given in the following in-line style sheet.

```
<p><img style="display:block;margin-left:auto;margin-right:auto;" src="Stonehenge.jpg
```

Property	:	Values
display		block inline

igure **4.3.3**: Display property and values. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)



Figure **4.3.4**: Centered alignment of an image. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Image resources have an inherent width value. When margin-left and margin-right are both set to auto, the browser calculates the amount of space available, minus the width of the image, and divides it evenly between the left and right margins. For other block-level elements which have no inherent width length, you must specify a width in order to horizontally center the element.

Floating Images

Alternative to positioning a picture on a line by itself is to place it at the left or right margin and permit word wrap around it. The picture is said to "float" to the left or right of its accompanying text. Floating an image makes use of the float style property.

Property	:	Values
float		left right none

Figure **4.3.5**: Float property and values. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The picture in Figure **4.3.6** appears at the right margin of the page (floated right). The accompanying text paragraphs wrap around the image. All styling is contained in an embedded style sheet.

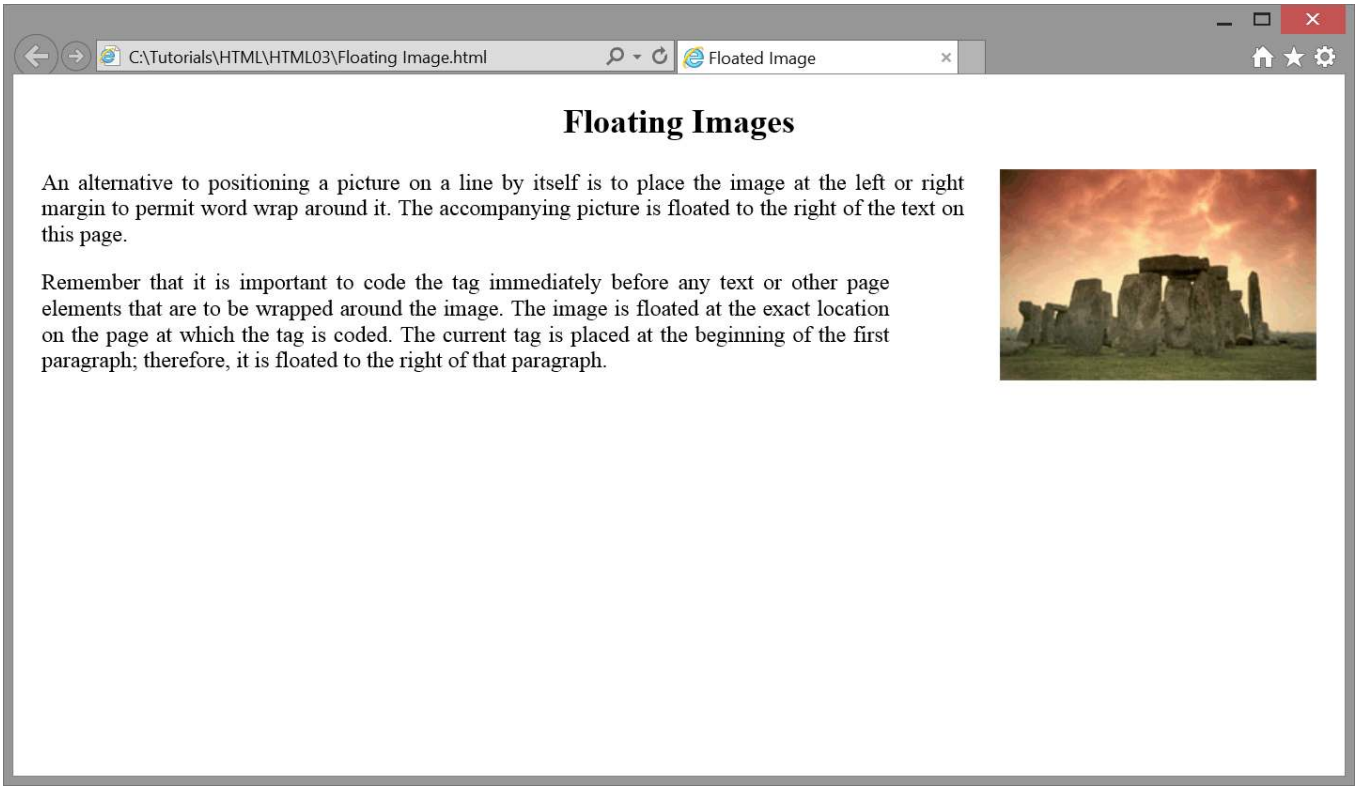


Figure **4.3.7**: A floated image. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

```
<head>
      <title> Floated Image </title>
      <style>
        body {margin:20px;}
        p    {text-align:justify;}
        h2   {text-align:center;}
        img  {float:right; margin-left:25px;}
      </style>
    </head>

    <body>
    <h2> Floating Images </h2>
```


https://eng.libretexts.org/@go/page/93802

```
 <p>
 <img src="Stonehenge.jpg" alt="Picture of Stonehenge">
An alternative to positioning a picture on a line by itself is to place the
image at the left or right margin to permit word wrap around it. The
accompanying picture is floated to the right of the text on this page. </p>

 <p> Remember that it is important to code the  <img>  tag immediately
before any text or other page elements that are to be wrapped around the
image. The image is floated at the exact location on the page at which the
 <img>  tag is coded. The current tag is placed at the beginning of
the first paragraph; therefore, it is floated to the right of that
paragraph. </p>
 </body>
```

Notice that the float property is associated with the img selector in the above style sheet. It is a property of the `<img>` tag -- the tag to be floated -- unlike the case where an image is placed on a line by itself and is aligned by the text-align property of its enclosing `<p>` tag.

Images that are floated to the left or right of the page must be coded immediately before any text that wraps around the image. In the above example, the `<img>` tag appears at the beginning of the paragraph that wraps around the image. No line breaks appear between the image and its surrounding content. The `<img>` tag is assigned the float:right style property to place it at the right margin of its enclosing paragraph. It also has a `margin-left: 25px` style setting in order to leave white space between the picture and its surrounding text on the left.

Images can be floated to the left or right of the page; they cannot be placed in the middle of the page with word wrap around both sides. When floating images, it is usually a good idea to include additional white space between the image and its surrounding text by extending its margins. Like most style properties, the float property can be applied to many other tags. In later tutorials you will learn how to float text containers to the left or right of a page.

In the above example, all tags are styled with an embedded style sheet. Although not shown here, any other `<p>` , `<h2>` , and `<img>` tags appearing on this page would take on these same styling characteristics.

Wrapping Text Around Images

When images are floated to the left or right of the page, subsequent text wraps around the image. This is illustrated in Figure **4.3.8** where an image floats to the left so that a caption for the image wraps to its right. At the same time, a follow-on paragraph also wraps around the image.

Here is a captioned image that is floated to the left so that its caption appears on its right.

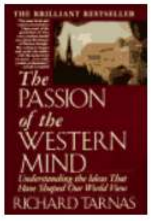


Figure 1.
The Passion of the Western Mind.
This following paragraph also wraps to the right of the picture, since it is coded in the line below the caption.

Figure 3-16. A floated image with text paragraphs wrapped around it.

Figure **4.3.8**: A floated image with text paragraphs wrapped around it. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The clear Property

Although any subsequently coded text wraps around a floated image, you might not want this to happen. For instance, in the following display the picture caption wraps to the right of the image; however, the follow-on paragraph, rather than being wrapped around the image, appears below the image.

Here is a captioned image that is floated to the left so that its caption appears on its right

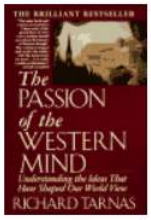


Figure 1.
The Passion of the Western Mind.

This following paragraph appears below the picture rather than wrapped in-line below the caption.

Figure **4.3.9**: A floated image with a follow-on paragraph appearing below the image. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Code to produce this display is shown below.

```
  <p> Here is a captioned image that is floated to the left so that its caption
appears on its right.  </p>

  <div>
  <img src="book.gif" alt="Book picture" style="float: Left(); margin-right:10px;
  <b> Figure 1.  </b>   <br>
  <i> The Passion of the Western Mind  </i> .
  </div>

  <p style="clear:left"> This following paragraph appears below the picture
rather than wrapped in-line below the caption.  </p>
```

You can ensure that a page element always appears at the left or right margin of the page by assigning it the clear style property. The value of this property can be left, right, both, or none (default).

Property	:	Values
clear		left right both none (default)

Figure **4.3.10**: Clear style property and values. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

In the previous example, the follow-on paragraph is given the in-line style setting clear:left;. This means that the paragraph is displayed on the next available line that has a "clear" left margin. Therefore, the paragraph begins on the line immediately below the picture, on the first line whose left margin is not occupied by another element.

Any time a page element is floated to the left or right of the page, you may need to ensure that a subsequent text or graphic element appears below the floated element rather than being displayed as part of the wrapped content. The value of the clear property assigned to this non-wrapped element depends on the margin location (left or right) of the floated element. The setting clear:both ensures that neither margin is occupied by a page element before displaying the so-styled element.

The text-indent Style

Previous examples of left-aligned paragraphs show them in "blocked style" wherein all lines of text are blocked at the left margin. It is common practice to indent the first line of a paragraph as a visual clue that this is the beginning of a new paragraph. First-line

indention is not really necessary when paragraphs are separated by blank lines as is the case with Web pages. Still, you may wish to apply this formatting to soften the look of your pages.

The **text-indent** style property is used to indent the first line of a paragraph. The amount of indention is given in number of pixels or as a percentage of the width of the page.

Property	:	Values
text-indent		*n*px *n*%

Figure **4.3.11**: Text indention property and values. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The following code indents the first line of two paragraphs: the first by 25 pixels and the second by 5% of the width of the page. Browser output is shown in Figure **4.3.12**.

```
<p style="text-indent:25px;"> This paragraph is formatted using style="text-
  indent:25px;". The first line of the paragraph is indented 25 pixels from the
  left margin. The remaining lines of the paragraph are blocked at the left
  margin and wrap at the right margin. </p>

 <p style="text-indent:5%;"> This paragraph is formatted using style="text-
  indent:5%;". The first line of the paragraph is indented 5% of the page width
  from the left margin. The remaining lines of the paragraph are blocked at
  the left margin and wrap at the right margin. </p>
```

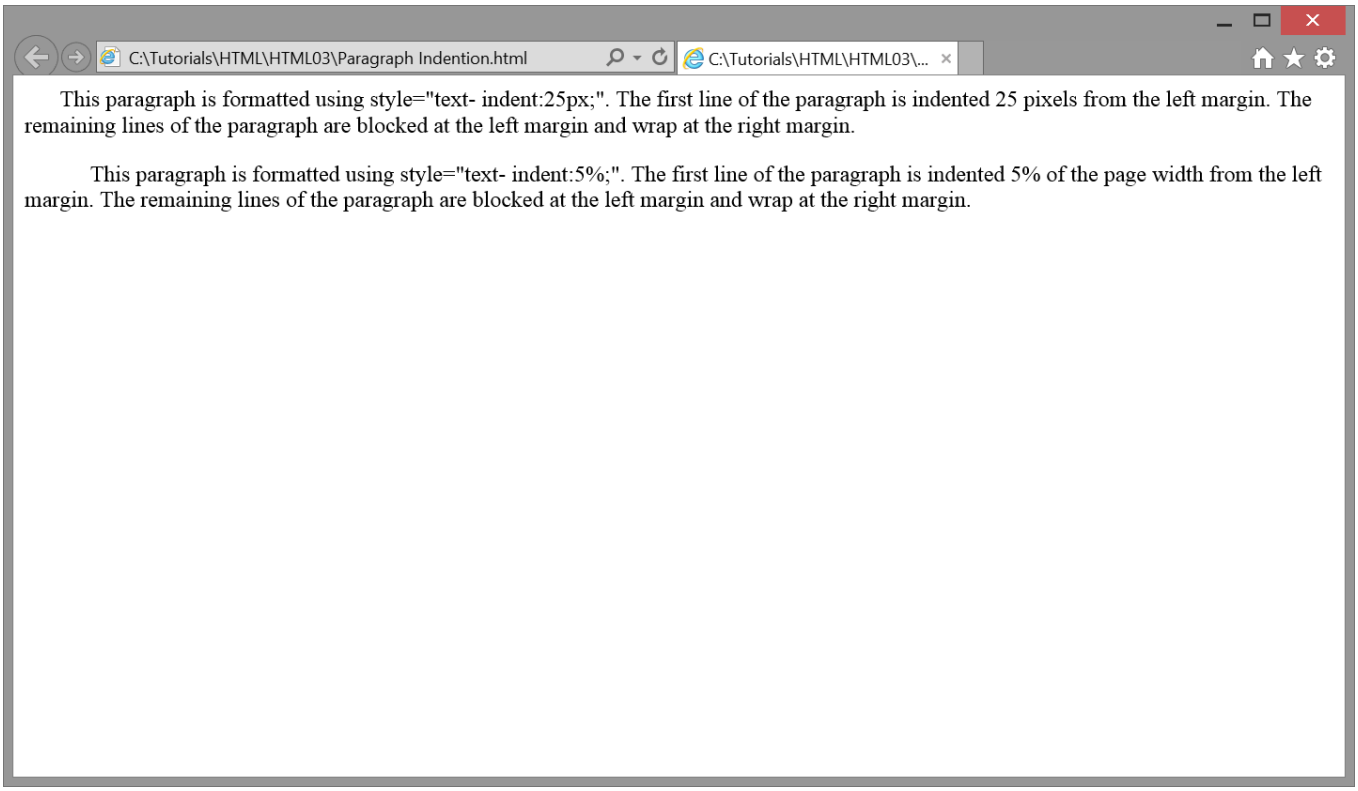


Figure 4.3.12: Text indentations applied to paragraphs. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

In this example, both pixel and percentage values produce approximately the same amount of first-line indention. When using a percentage measure for paragraph indention, remember that this value is based on the width of the page. Different amounts of indention will occur depending on the width of the browser window. In order to maintain a fixed amount of indention regardless of window size, use a pixel or em value rather than a percentage.

4.3: Alignment Styles is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

4.4: Width and Height Styles

There are two general **width** and **height** style properties that can be used to size any page element. The width property gives the width of an element in pixels or as a percentage of the page width. The height property sets the element's height in pixels or as a percentage of page height.

Property	:	Values
width		*n*px *n*%
height		

Figure **4.4.1**: Width and height properties and values. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

In subsequent tutorials, these properties are applied to a wide assortment of page elements. For this initial introduction, widths and heights are applied to horizontal rules and graphic images to illustrate their use.

Rule Widths and Heights

As described previously, paragraph level content can be visually and thematically separated by displaying a horizontal rule between them. This is done with the `<hr>` tag to draw a line across the page.

The default rule is 2 pixels in height and spans the width of the browser window. You can change these dimensions by coding width and height styles for the `<hr>` tag. This sizing is shown by the following in-line style sheet and browser display.

```
<hr style="width:75%; height:7px; text-align:center;">
```

When specifying widths of page elements, it is often preferable to code them as percentages. In this way, the widths always appear in correct proportion to other elements on the Web page regardless of whether the user adjusts the size of the browser window. If a horizontal rule is sized to less than the width of the browser window, then it also can be aligned with text-align:left, text-align:center, or text-align:right to position it horizontally on the line. In the previous example, the rule is 75% of the width of the page and is centered.

Normally all horizontal rules on a page will be of the same style. Therefore, it makes sense to code this styling in an embedded or linked style sheet so that it can be coded one time for application to all rules on the page or all rules on the site. The following code is an example of how horizontal rules can be standardized for a page with an embedded style sheet.

```
<style>
   hr { height:1px; width:50%; text-align:center; }
</style>
```

With this styling, all `<hr>` tags on the page produce rules that are 1 pixel in height, 50% of the width of the browser window (irrespective of the window size), and centered horizontally on the page.

Image Widths and Heights

It is often the case that graphic images are not the exact widths or heights needed to fit comfortably on your Web page. This is a recurring problem if you do not possess, or do not know how to use, imaging software to resize the image prior to placing it on the page. However, by applying width and height style properties to these images they can be resized on-the-fly when displayed in the browser.

The actual size of the image shown in Figure **4.4.2** is 162 x 108 pixels. When displayed on the page, its width is set to 100 pixels with an in-line style sheet.

```
<img src="Stonehenge.jpg" alt="Picture of Stonehenge" style="width:100px;">
```



Figure 4.4.2: A dynamically resized image. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

When resizing images, set either the width or height property, usually not both. The browser automatically adjusts the other dimension to maintain proper proportion. Of course, you can set both properties independently if you know their correct resized dimensions or if you wish to create special visual effects with disproportionate widths and heights.

Even though an image can be automatically resized when displayed in the browser, it still retains its original file size. An extremely large image still can take up considerable disk space and can cause delays in downloading prior to its display. Ideally, you should edit a large image to reduce its actual size to the dimensions of its page display rather than using width and height styling to reduce only its display size.

An image should not normally be displayed at larger dimensions than its original size. When increasing the size of an image you risk losing resolution and producing a fuzzy "pixelated" image.

Recall from a previous tutorial that the `<img>` tag has deprecated width and height attributes for resizing images. These attributes should be replaced by their comparable width and height style properties for concordance with HTML5 standards. Although their names are the same, the way in which attributes and styles are applied is different.

The width and height style settings can be applied to almost any HTML5 element by default, not just to `<hr>` and `<img>` tags as in the previous examples. In subsequent tutorials, there will be recurring occasion to resize other page elements to fit them with exact dimensions on the page.

4.4: Width and Height Styles is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

4.5: Basic Color Styles

Colors can be applied to various page elements with the **color** and **background-color** properties. Color values can be given by a color name, a hexadecimal value representing the combination of red, green, and blue hues to be applied, or by an rgb(red,green,blue) functional form where each hue is given as a decimal or percentage value. For example, color specifications can take the following forms:

```
color:red
color:#FF0000
color:#F00
color:rgb(255,0,0)
color:rgb(100%, 0%, 0%)
```

These color assignments are discussed more fully in a later tutorial. For now, you can use common system color names to bring color styling to your pages. There are 16 basic color names recognized as standard Windows colors and shown in Figure 4.5.1. These colors can be applied to page elements with the style settings color:name or background-color:name coded in the style sheet associated with the element.

Property	:	Values
text-align		aqua, black, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow
background-color		

Figure 4.5.1: Basic color properties and names. (("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0))

Text and Background Colors

Text colors are applied by assigning a color value in the style sheet associated with the container tag enclosing the text. For instance, a heading can be displayed in blue by assigning this color name in an in-line style sheet for the `<hn>` tag.

```
<h1 style="color:blue;">A Blue Heading</h1>
```

If an entire paragraph is to have a particular text color, then that color name can be applied with a style sheet associated with its `<p>` tag.

```
<p style="color:red;">...red paragraph text...</p>
```

In certain cases, you may wish to display all text on a page in a particular color. This is easily done by assigning the color property to the `<body>` tag. The following code displays all text on a page in green by using an embedded style sheet associated with the body selector.

```
<style>
   body {color:green;}
</style>
```

Any text container can also be given a background color by applying the background-color property. Just make sure that the color chosen is complementary to the text color and does not make it difficult to read the text. In the following style sheet, yellow is used as the page background color behind its green text.

```
<style>
 body {color:green; background-color:yellow;}
</style>
```

Rule Colors

The same background-color styling can be applied to horizontal rules to change their filled in color. In the following example, rules are displayed with various style properties to produce different colors, sizes, and page positioning. At the same time, the foreground (text) color of the page is blue and the background color is yellow. Browser output is shown in Figure **4.5.2**.

```
<body style="color:blue; background-color:yellow;">
<p>
 A blue horizontal rule 10 pixels in height, 300 pixels in width, and
 aligned at the right margin:
</p>

<hr style="background-color:blue; height:10px; width:300px; text-align:right;">
<p>
 A red horizontal rule 2 pixels in height, 75% of the page width, and
 centered:
</p>

<hr style="background-color:red; height:2px; width:75%; text-align:center;">

<p>
 A green horizontal rule 10 pixels in height; 25% of the page width, and
 aligned at the left margin:
</p>

<hr style="background-color:green; height:10px; width:25%; text-align:left;">
</body>
```

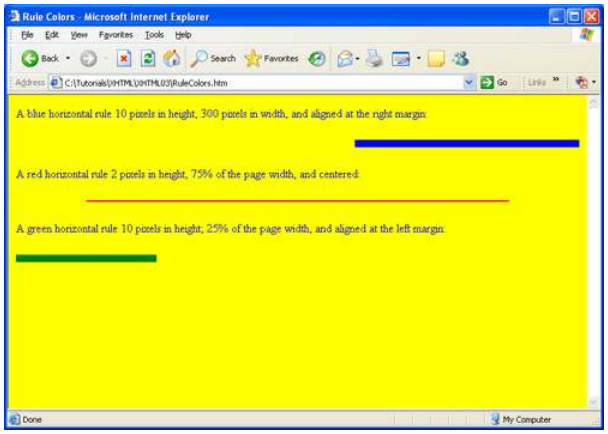


Figure **4.5.2**: Styled horzontal rules. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

In this example, all rule styling is applied with in-line style sheets. The individual rules have different settings for which an embedded style sheet with a single hr selector is not appropriate. Body styling also uses an in-line style sheet. Although, in this

case, an embedded style sheet with a body selector could have been used to apply foreground and background colors.

The color and background-color properties can be applied to all HTML5 elements. Many different color combination can be used to produce a full spectrum of colors. Extended discussion of color values and styling is saved for a later tutorial. Use of the basic color names can get you started in introducing color variety to your pages.

4.5: Basic Color Styles is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

4.6: Basic Color Styles

Up to this point, styles have been applied to individual page elements by associating style sheets with their tags. This technique works fine in most cases; however, there are styling situations not covered by this method.

For example, within a paragraph you might wish to apply a style to a particular letter, word, phrase, or other text string located inside the paragraph. Applying a style sheet to the `<p>` tag does not give you the ability to select which "substring" of the paragraph to style. All characters in the paragraph receive the same styling.

Another example is a consecutive group of paragraphs to which you want to apply the same style. As you may have expected, you can duplicate and apply the same in-line style sheet individually to the separate `<p>` tags to give them the same appearance. Nonetheless, the ability to apply the same style to the paragraphs as a group without having to style them individually would be more convenient.

For these and other styling situations involving strings of text less than a paragraph in length and blocks of text longer than a paragraph, HTML5 provides "marker" tags to identify and isolate particular sections or subsections of a page to which styling can be applied.

The `<span>` Tag

A **`<span>`** tag is an in-line tag placed around text for the purpose of identifying a string of characters to which this tag's style sheet is applied. The tag can enclose a single letter, a word, a phrase, a sentence, or any other substring of text for the purpose of identifying it for the application of styling. As an in-line tag, the `<span>` tag surrounds a string of text enclosed inside a block-level container.

In the following paragraph, the words "RED" and "BLUE" are isolated with `<span>` tags as words to which these tags apply their color properties. When this paragraph is displayed in the browser, the two words are rendered in their associated colors.

```
<p> This paragraph contains the word <span style="color:red;"> RED </span>
that is surrounded by a <span> tag to apply this color property to the word. In
this sentence the word <span style="color:blue;"> BLUE </span> is given that
color. </p>
```

A `<span>` tag is nothing more than a marker to isolate text to which its style sheet can be applied. It has no built-in formatting characteristics of its own or semantic meaning. It does not force a line break nor does it produce a blank space. Therefore, it can be used in-line within the normal flow of text simply to style its enclosed content. Naturally, the style that is applied must be appropriate to the enclosed text string. For instance, it would not be appropriate to apply the text-indent property to indent the "first line" of a single word enclosed by a `<span>` tag.

The `<div>` Tag

A `<div>` (division) tag is a similar type of marker to the `<span>` tag. Its purpose is to enclose and designate a collection of page elements for application of styling to the enclosed set without providing any semantic meaning. The `<div>` tag is a block-level tag because it encloses other tags, and more importantly, it forces a line break on the page by creating a line break before and after its enclosed content.

Use of the `<div>` tag is illustrated by the following two paragraphs. Both paragraphs are styled the same. However, instead of coding these styles with in-line style sheets in both `<p>` tags, the paragraphs are surrounded by a `<div>` tag which applies these styles. The paragraphs inherit the text-indent and text-align styles of the enclosing `<div>` tag. Browser output of this formatting is shown in Figure **4.6.1**.

```
<div style="text-indent: 25px; margin-left:30px; margin-right:30px; Text-align: justi

  <p>
```

```
    This paragraph has first-line indention of 25 pixels and is justified
    between its margins.  This paragraph's margin-left and margin-right
    properties are equal to 0px (default value), but its containing div has left
    and right margins of 30px.
  </p>

  <p>
    This paragraph also has first-line indention of 25 pixels and is justified
    between its margins.  This paragraph's margin-left and margin-right
    properties are also equal to 0px (default value), but its containing
    div has left and right margins of 30px.
  </p>

</div>
```

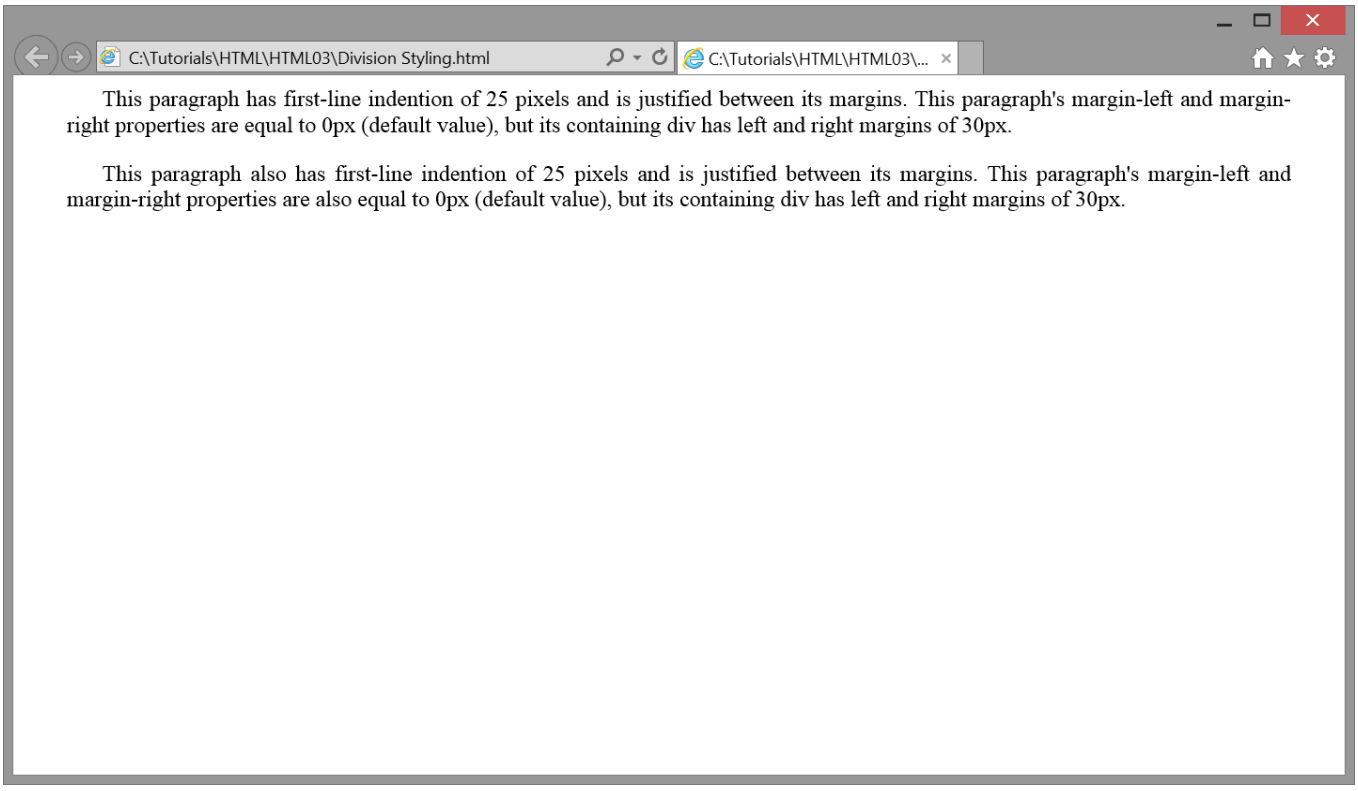


Figure **4.6.1**: Paragraphs styled with enclosing `<div>` tag. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The `<div>` tag does not have any visible formatting characteristics of its own other than the fact that it creates a line break before and after its enclosed content. These line breaks are not evident in the above example since `<p>` tags themselves produce their own line breaks. Such line breaks are collapsed together with those produced by the `<div>` tag.

Since the `<div>` tag, like the `<span>` tag, provides great flexibility in enclosing and styling page content, there are numerous occasions throughout these tutorials where spans and divisions are used to apply formatting to a wide assortment of page elements.

4.6: Basic Color Styles is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

4.7: ID and Class Selectors

Style Selectors and Classes

Use of embedded style sheets has been suggested as the normal way to apply page-wide styles to tags. Where particular tags need to take on special styles different from these page-wide styles, in-line style sheets are used to modify the styling.

Where practical, embedded style sheets should always be the preferred styling method, even for application of special one-time-only styling. Embedded style sheets localize style settings within the single `<style>` section of the page, thereby making them more accessible than if scattered across multiple in-line style sheets. Also, embedded style sheets can be easily promoted to linked style sheets for site-wide application. Discussed below are ways to apply special styling to page elements by using embedded style declarations rather than in-line style sheets.

Simple Selectors

By way of review, most embedded styling can be done with individual tag selectors and their associated style declarations. These simple selectors declare default tag styling for an entire page. The general format for a simple selector

```
selector {property:value [; property:value] ...
```

Any number of selectors can be combined with any number of style properties within an embedded style sheet. For example, the following simple selectors set page-wide formatting for page margins, headings, and paragraphs. Anywhere one of these tags is encountered on the page, the browser applies the associated style.

```
<style>
   body       {margin:20px; color:black;}
   h1         {color:blue; text-align:center;}
   h2         {color:blue; text-align:left;}
   p          {text-align:justify; text-indent:25px;}
</style>
```

There will likely to be occasions where these page-wide styles need to be modified for particular tags. For example, one or more of the paragraphs in the document may require different alignment or indention, or a particular heading may use a different style to emphasize it differently from other headings. Although you can use in-line style sheets to override these embedded style declarations, there are ways to designate exceptional styling within the embedded style sheet itself.

ID Selectors

One way to apply special styles to individual page elements is to provide the element with a unique identifier. After which, special styling can be applied only to the element with that identifier that is within the embedded style sheet.

For instance, assume that common paragraph indention and alignment are adopted for an entire Web page with the embedded style sheet declaration shown above. That is, all paragraphs have 25-pixel indentions and are text justified with first-line indents of 25 pixels. Now assume that one particular paragraph on the page requires different styling. This single paragraph needs to be offset from surrounding paragraphs with 25-pixel left and right margins and it should not have first-line indention.

The first step in styling this particular paragraph is to assign it an **id** value. An id is a unique identifier for the tag. It can be any name you choose using combinations of alphabetic and numeric characters. In the following code, **id="Special"** is assigned to the paragraph requiring special formatting.

```
<p  id="Special" > This paragraph requires special styling different from regular
   paragraphs on the page. It should be offset 25 pixels from both margins and not
   have first-line indention. </p>
```

With an id assigned to this `<p>` tag it can be designated for special styling by using an ID selector in the embedded style sheet. The general format for an ID selector is

```
#id{property:value [; property:value]...}
```

Styling is applied to the element on the page that contains an id="value" property, where "value" matches the #id name. Thus, an ID selector can be added to the above style sheet to format the special paragraph differently from regular paragraphs:

```
<style>
  body       {margin:20px; color:black;}
  h1         {color:blue; text-align:center;}
  h2         {color:blue; text-align:left;}
  p          {text-align:justify; text-indent:25px;}
  #Special  {text-indent:0px; margin-left:25px; margin-right:25px;}
 </style>
```

The syntax #Special refers to the tag with the id value (#) of "Special". The standard 25-pixel first-line indention given by the p simple selector is reset to 0 pixels for this particular paragraph. Also, the paragraph is given left and right margin settings different from normal paragraphs. Nevertheless, its text remains justified as inherited from standard paragraph styling given through the p selector. In other words, all paragraphs inherit the styling given by the p simple selector unless these styles are modified or supplemented by an #id selector.

ID Selectors can be used for identifying and styling any HTML tag. Just assign a unique id="value" attribute to the tag and add the ID Selector #value to the stylesheet. ID Selectors should be unique and assigned to only one tag in the HTML document. In this way, the id property in tags serve to uniquely identify them for specific reference in stylesheets.

Styling Spans and Divisions - TOP

You should *not* style `<span>` or `<div>` tags with simple selectors in an embedded style sheet. These tags often appear multiple times on a page to isolate and style different portions of text and different collections of page elements, all of which usually require a different styling. Thus, you cannot associate one particular style with these tags without reducing their flexibility in styling other spanned text or divisions that require a different styling. In other words, do **not** code the following simple selectors in embedded style sheets.

```
<style>
 div     {property:value}
 span    {property:value}
</style>
```

However, by using ID selectors you *can* apply different styles to different `<span>` and `<div>` tags without committing them to one particular style.

In the following example, two paragraphs are given the same style by enclosing them inside a `<div>` to which special styling is applied. The style effects only this single division through its unique ID selector. Styling of other divisions is not affected since they do not have this id value. At the same time, two `<span>` tags take on their unique stylings through their unique id values, and leave any other stylings of spanned text unaffected.

```
 <style>
    #Justify   {text-align:justify;}
    #Red    {color:red;}
    #Blue     {color:blue;}
```

```
    </style>

    <div id="Justify">
        <p> This paragraph is styled by inheriting the styling of its container
       division which is formatted with the  <span   id="Red" >" Justify "</span> styl

        <p> This paragraph is also styled by inheriting the styling of its container
       division. In addition,  <span  id="Blue" > BLUE </span>  styling is applied to
       one of its words. </p>
    </div>
```

Style Classes

By defining a **class selector** a general-purpose style can be declared for broad application to any tags needing to share the style. A class selector is defined in an embedded style sheet as shown below

```
.class {property:value [ ;property:value] ...}
```

Rather than naming a tag or using an id as the selector, a ***class*** name is supplied, *preceded by a period.* This *.class* selector can be a name of your choosing, but cannot include spaces or special characters. Any combination of style properties and values can be associated with this style class.

The following embedded style sheet includes a style class named .Offset. Once this class has been defined, it is assigned to any tag with the **class**="*class*" attribute. Below, the style class is applied to a paragraph by assigning it to the `<p>` tag. This paragraph takes on the styling of the .Offset class, in this case overriding normal styling given by the p simple selector.

```
<style>
 p       {margin:20px;}
 .Offset {margin-left:30px; margin-right:30px;}
</style>

<p   class="Offset" > This paragraph requires special styling different from normal
paragraphs on the page. It has left and right margins of 30 pixels. </p>
```

(Note that the class assigned in the tag does *not* include the period that is necessary when declaring the *.class* selector in the style sheet.)

A style class is a general-purpose style that can be applied to any type of tag simply by assigning the class to the tag. Furthermore, unlike the ID selector, the class selector can be assigned to any number of tags. Thus, any paragraphs, divisions, or other block-level tag types can reflect the above styling by assigning them to the .Offset class. Of course, the tag to which the class is applied must be receptive to the particular properties and values declared for the class.

Style classes are particularly relevant to `<span>` and `<div>` tags for styling text strings and blocks of code without committing these tags to one particular style. The tags become carriers for many different styles simply by assigning different style classes to them. The following code is an example of applying various style classes through various tags to produce the page display shown in Figure **4.7.1**.

```
<head>
      <title> Class Styles </title>

    <style>
```

```
    .Offset  {margin-left:25px; margin-right:25px; text-align:justify;}
    .Red     {color:red;}
    .Blue    {color:blue;}
    .Rule    {height:2px; width:75%; text-align:center; color:green;}
  </style>

  </head>
  <body>

  <hr  class="Rule" >

  <div  class="Offset" >

    <p> Here is a paragraph that takes on the formatting given for its enclosing
   division. Within this paragraph the word  <span  class="Red" > RED </span>  has i
   own style class applied. </p>

    <p> This paragraph also is styled by its enclosing division. Within this
   paragraph the word  <span  class="Blue" > BLUE </span>  has its own style class
   applied. </p>

  </div>

  <hr  class="Rule"  >

  </body>
```

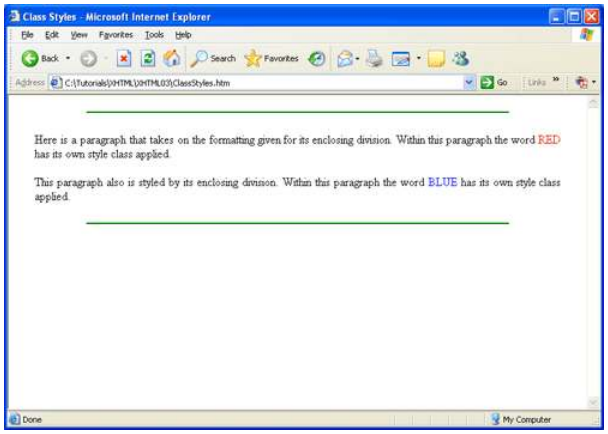


Figure **4.7.1**: Application of class styles. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Both of the above paragraphs are styled by enclosing them inside a division to which the .Offset style class is assigned. Thus, the paragraphs inherit division styling. In this case, both paragraphs are offset from the page margins and the text is justified. This technique has the same effect as declaring an ID selector for division styling (div#Offset) and applying the style through this ID selector (`<div id="Offset">`). However, the .Offset class is more flexible since it can be associated with other multiple tags. Any tags on the page can take on the .Offset style by assigning them to this class.

Within each paragraph are individual words that take on color styling. These words are enclosed inside `<span>` tags through which colors are applied by assigning the tags to either the .Blue or .Orange style class. Incidentally, if any other sections of the page — a paragraph, a heading, whatever — need to be displayed in one of these colors, then their enclosing tags can be assigned these same style classes. Any normal stylings of these tags are supplemented with assigned colors.

Both horizontal rules are 75% of the width of the page, 2 pixels in height, green, and centered. The choice is made to define a style class to represent this styling and to assign all `<hr>` tags to this class. Of course, these styles could be defined for the simple hr selector without using a class; however, the choice is made to create classes for all stylings and to assign tags to these classes according to what is required. It would not be unusual to find an embedded style sheet composed exclusively of style classes that are selectively applied to all varieties of tags requiring those styles.

CSS Media Types

CSS Media Types allow you to specify how documents will be presented in different media such as the browser screen, on paper, hand held devices, and other types of common media. The ability to control page layout in different types of media can be helpful because a document usually needs a larger font-size on a screen than on paper; and sans-serif fonts are easier to read on the screen, while serif fonts are easier to read on paper. Among other things, CSS Media Types also allow you to customize your page for various types of accessibility devices. Guidelines for creating accessible sites will be discussed later in another tutorial.

The table below lists the different CSS Media Types and describes how they are used.

Media Type	Description
all	Used for all media type devices.
braille	Used for braille tactile feedback devices.
embossed	Used for printing on braille printers.
handheld	Used for small handheld devices.
print	Used for printers.
projection	Used for projected presentations, like slides.
screen	Used for computer screen
speech	Used for speech and sound synthesizers.
tty	Used for terminals or devices with limited display capabilities.
tv	Used for televisions.

Figure **4.7.2**: Table of CSS Media Types. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The @media media type { } rule allows different style rules for different media in the same style sheet.

```
<head>
      <title> Class Styles </title>

    <style>
   @media screen {
     p {font-size:11pt;font-family:Arial;}
   }

   @media print {
     p {font-size:10pt;font-family:Times New Roman;}
```

```
}
 </style>

 </head>
 <body>

 <p> This example tells the browser to display a
11 point Arial font on the screen, but if the page is
printed, it will be in a 10 point Times New Roman font.
 </p>

 </body>
```

When the page is viewed in the browser, the text is rendered in 11 point Arial. When the page is printed, the font size is 10 points and in Times New Roman.

Styling Strategies

By now, you are probably confused. You have a choice of using linked style sheets, embedded style sheets, in-line style sheets, or some combination of the three. When using linked or embedded style sheets you have a choice of using simple tag selectors, ID selectors, or style classes. Later in this tutorial, additional styling options are presented. What's a person to do about all these options?

Given the various approaches to styling, a reasonable strategy to follow is outlined below. You may developed your own preferred approach as you become more familiar with style sheets, but the following outline is a good beginning to sort out the options.

- Use an embedded style sheet as the primary styling method. This permits isolation of styles within a single section of the page thereby making it easier to locate and change styles. It also promotes embedded styles to a linked style sheet for site-wide application.
- Within the embedded style sheet, use simple tag selectors to apply common, page-wide styling to all tags of the same type.
- Define style classes to apply styling through general-purpose tags such as `<span>` and `<div>` tags. Also, use style classes when two or more tags require formatting different from or supplementary to that given by their simple tag selectors.
- For single tags requiring unique styling, assign an id value to the tag and apply its styling through an ID selector in the embedded style sheet.
- Use an in-line style sheet where one-time styling is required and there is overriding convenience in having the style sheet coded inside the tag. Too many in-line style sheets are difficult to track down for changes; plus, identical in-line styles have to be changed in multiple locations, leading to possible oversight and errors.

Although preference is given to using embedded style sheets -- and by extension, linked style sheets the -- these tutorials often use in-line style sheets as instructional devices to introduce and describe style properties. The assumption is, however, that in-line styles will be elevated to embedded or linked styles when pages are "put into production."

This tutorial has given you a first introduction to style sheets. There are many more style properties to come and many other variations on those introduced here. The importance of style sheets cannot be over-emphasized. With deprecation of most tag formatting attributes, style sheets have become the primary technique for arranging and displaying page content. The use of in-line, embedded, and linked style sheets has become the essence of HTML5 coding.

4.7: ID and Class Selectors is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

5: Text Formatting

5.1: Font Settings
5.2: Text Properties
5.3: Font Style Tags
5.4: Text and Background Colors
5.5: Special Characters

5: Text Formatting is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

5.1: Font Settings

When text is displayed in a browser, it appears in a default font face, size, style, and color. Most browsers use the Times New Roman font face at approximately 1 em (12-point) size and rendered in black. However, several style sheet settings permit you to change these default settings to bring a different look to your pages.

When working with standard print documents, it is common to use absolute units of measure when setting font size. Absolute units include **points** and **pixels**. With absolute units, the font size values render inconsistently across platforms and can't be resized by the browser. For this reason, relative units of measure should be used when setting the font-size for web documents. Relative units include the **em** and **percentages**. The common 12pt font size is equivalent to 1 em.

When working with text fonts, you need to keep in mind that the faces that are displayed in the browser are limited to those installed on the user's PC. Therefore, you need to be aware of what kinds of computers your visitors use and what type faces you expect to have been installed. It is best to stick with common faces unless your Web audience is known in advance. The following are common font faces in all versions of Windows and Mac, displayed here in 1 em size.

Windows font, Font family	Mac Equivalents
arial	helvetica
arial narrow (sans-serif)	arial narrow (sans-serif)
comic sans ms (cursive)	comic sans ms (cursive)
courier new (monospace)	courier new (monospace)
georgia (serif)	georgia (serif)
impact (sans-serif)	impact[5] (sans-serif)
tahoma (sans-serif)	geneva(sans-serif)
times new roman (serif)	times(serif)
verdana (sans-serif)	verdana (sans-serif)

Figure **5.1.1**: Common font faces. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Font Styles

There are six font style settings that can be used to add variety, interest, or emphasis to the text on a Web page. These property settings are listed in Figure **5.1.2**.

Property:	Value
font-family	Font name can be any system font or the generic font family category values serif sans-serif cursive monospace fantasy Multiple values can be specified in order of preference, separated by commas. The first located font face is applied: font-family:arial,verdana,sans-serif
font-size	Font size specified as a unit of measurement, normally points (pt). If no unit of measurement is supplied, the default is pixels (px). Other font-size units include em (em), and percentages (%).
font-style	Font style specified as normal *italic* oblique
font-weight	Font weight specified as lighter normal **bold** bolder or a value from 100 (lightest) to 900 (darkest). Not all number values in the range are recognized by all browsers.
font-variant	Font variant specified as normal small-caps
font	A shorthand method of specifying a set of font styles by assigning values separated by spaces in the *exact order*: font-style font-variant font-weight font-size font-family. Unassigned values can be missing from the listing: style="font:italic bold .84em times new roman"

Figure 5.1.2: Font style properties. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Any combinations of these styles can be applied to any text on the page. They can be associated with a `<body>` tag to apply to the document as a whole. They can be coded for a p selector to apply to all paragraphs. They can be coded as classes and applied with `<span>` tags surrounding letters, words, phrases, sentences, or other selected strings of text within a paragraph. They can be applied as classes through `<div>` tags to blocks of text.

The `font-family` property needs to be specified to change the browser's default setting from Times New Roman. When using an uncommon font family to display special font styles that may not be present on every user's computer, you should provide other, more common type faces from which the browser can choose. The font-family property gives these choices in a comma-separated list. The browser searches the computer system for the first matching type face and uses it to display the text. For instance, the following style sheet sets the standard font style for a page.

```
<style>
  body {font-family:helvetica, verdana, sans-serif}
</style>
```

Figure 5.1.3: Font family value choices. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

First, the browser attempts to use the Helvetica type face. If it is not present on the user's system, the browser looks for the Verdana type face. If it is not present, the browser uses whatever built-in non-serif type face is available. The generic font names serif, non-serif, and monospace will always locate a type face that is similar to Times New Roman, Arial, and Courier, respectively.

You also need to set the `font-size` property in order to change the browser's default 1em (12-point) size. As mentioned earlier, font-size is commonly given in point sizes; however, it is best to use a relative unit of measure such as the em or percentages. Absolute units of measure such as points (pt) can render inconsistently across platforms and can't be resized by the User Agent (e.g browser). Keep the usage of such units for styling on media with fixed and known physical properties (e.g print).

The following table shows the different types of `font-size` property values and the recommended uses.

Category	Values	Notes
Text Value	xx-small, x-small, small, medium (default), large, x-large, xx-large	A relative unit of measure. Scales well when text is resized in the browser. Text size options are limited.
Pixel Unit (px)	Numeric value with unit such as 12px	A relative unit of measure. Pixel display depends on screen resolution; may not scale well in every browser when the text is re-sized
Point Unit (pt)	Numeric value with unit such as 12pt	An absolute unit of measure. Use to configure print version of a web page. This measure may not scale in every browser when the text is resized.
Em Unit (em)	Numeric value with unit such as .75em	A relative unit of measure; Recommended by the W3C; Scales well when text is resized in the browser.
Percentage value (%)	Numeric value with unit such as 90%	A relative until of measure. Recommended by the W3C; Scales well when the text is resized in the browser.

The `font-style` property is a choice between normal and italic styles; oblique usually is rendered in italic as well. Browser default is the normal style.

The `font-weight` property is a choice between normal and bold. The values lighter and bolder are not commonly recognized in browsers; neither are all numeric values recognized. Normally, values between 100 and 500 are displayed in normal weight, and values between 600 and 900 are displayed in bold.

The `font-variant` style of small-caps changes all letters to upper-case in a slightly smaller font size. Specifying normal returns the text to standard display.

Font style declarations can make use of individual property settings, or values can be combined in the single `font` property as a shorthand way of expressing the declarations. That is, a set of style sheet declarations can be made as shown in Figure **5.1.4**.

```
{font-family:arial;
 font-variant:small-caps;
 font-size:.84em;
 font-weight:bold;
 font-style:italic}
```

Figure **5.1.4**: Font settings through use of individual style properties. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

or combined as shown in Figure **5.1.5**,

```
{font:italic small-caps bold 10pt arial}
```

Figure **5.1.5**: Font settings through use of a single font property. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

with values separated by spaces. When using the font property, not all values need to be specified, but they need to be in the order: font-style font-variant font-weight font-size font-family. A typical declaration could include, for example, only the `font-size` and `font-family` values: `{font:10pt arial}` .

The following code uses an embedded style sheet with combinations of font settings. In this example all font stylings use the shorthand `font` property. Browser display of this page is shown in Figure **5.1.7**.

```
TYPE html>

 lang="en">
>
itle> Example Font Styles </title>

e type="text/css">
ody    {font:1em arial; text-indent:25px; margin:20px}
ead    {font:bold 1.17em; text-align:center; text-indent:0px}
offset {margin-left:25px; margin-right:25px; text-indent:0px}
ap     {font:bold 2em}
ancy   {font:1.09em comic sans ms}
style1 {font:bold 1.17em times new roman}
style2 {font:bold 1em courier new; color:fuchsia}
le>

d>
>

class="head">Font Stylings</div>

pan class="cap">  T  </span> his page show various font stylings applied with an embedded style sheet.
 are applied to the entire page and to selected portions of text on the page. </p>

class="offset">

ese next two paragraphs are offset from surrounding paragraphs by their containment within a division.
vision has left and right margins of 25 pixels and neither of the paragraphs have first-line indents. </p>

ass="fancy"> This paragraph is styled with a class that changes the font face and size. Inside this paragraph
pan tag that styles  <span class="style1"> this string of text </span>  differently from the rest of the para
ition, this  <span class="style2"> word </span>  has its own special style. </p>

>

/>
l>
```

Figure 5.1.6: Font style settings for a page. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)



Figure 5.1.7: Application of font styles. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Styling for the `<body>` tag sets the default format for the entire page. Thereafter, various style classes are applied to selected portions of the text.

Nested Styles

The above page includes **nested** style sheets wherein an inner style setting inherits and then temporarily overrides, or modifies, the settings of an outer style. The outer `<body>` style, for example, sets page display in 12-point Arial type with page margins of 20 pixels and paragraph indention of 25 pixels. For instance, these settings remains in effect unless overridden by the enclosed `<div>` styles which reset margins for the two enclosed paragraphs to 25 pixels and reset paragraph indention to 0 pixels. These division styles are further overridden in one of the enclosed paragraphs wherein font styling is changed to 13-point Comic Sans MS while retaining division margins and indention. Within this paragraph, `<span>` tags are used to override paragraph styling for individual text strings that take on different font faces and sizes.

Style sheets can be nested inside one another so that styles cascade down any number of levels. Ending the style settings occurs in reverse order. This point is illustrated by the following code showing an indented structure of `<span>` tags that apply and then remove increasing font sizes by using in-line style sheets.

```
<p>
<span style="font-size:1em">
Here
    <span style="font-size:1.34em">
    are
        <span style="font-size:1.67em">
        styles
            <span style="font-size:2em">
            nested
            </span>
        inside
        </span>
    one
    </span>
another.
</span>
</p>
```

Figure **5.1.8**: Nested style settings using `<span>` tags. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Here are styles **nested** inside one another.

Figure **5.1.9**: Application of nested font styles using `<span>` tags.. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Although you are not likely to nest styles to this extreme, the point is that each closing `</span>` tag, beginning with the inner-most, closes its associated style, and causes the styling to revert to the next outer-most style until all styles have been ended. Just remember to keep the opening and closing styling tags properly paired.

The above example also reemphasizes the fact that `<span>` tags do not have formatting characteristics of their own. Although they are coded on separate lines to show the nesting structure in this example, they do not cause line breaks or spaces when displayed in the browser. The display effect is the same as if coding were entered as the following continuous string.

```
<p><span style="font-size:1em"> Here  <span style="font-size:1.34em"> are
<span style="font-size:1.67em"> styles  <span style="font-size:2em">nested
</span>  inside  </span> one  </span> another. </span></p>
```

Figure **5.1.10**: Alternate coding for nested style settings. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

In contrast, a `<div>` tag has the same styling purpose except that it produces a line break. The following code styles the above text string with `<div>` tags and produces the output shown in Figure **5.1.12**.

```
<div style="font-size:1em">
   Here
        <div style="font-size:1.34em">
           are
                <div style="font-size:1.67em">
                   styles
                        <div style="font-size:2em">
                           nested
                        </div>
                   inside
                </div>
           one
        </div>
   another.
</div>
```

Figure **5.1.11**: Nested style settings using `<div>` tags. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Here

are

styles

nested

inside

one

another.

Figure **5.1.12**: Application of nested font styles using `<div>` tags. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

are

styles

nested

inside

one

another.

Figure 4-5. Application of nested font styles using `<div>` tags.

You need to be alert to these spacing differences since `<span>` and `<div>` tags are used extensively to apply styling to page elements.

Deprecated `<font>` Tag

Although not recognized in XHTML or HTML 5, older settings font styles were with the `<font>` tag. This tag, like a `<span>` tag, encloses text to be displayed in a specified face, size, and color. The general format for the `<font>` tag and its attributes are shown below.

```
<font
    face="face1 [,face2] [,face3]"
    size="n|+n|-n"
    color="color"
    >
        ...text to be formatted...
    </font>
```

Figure **5.1.13**: Deprecated `<font>` tag. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The `face` attribute specifies a comma-separated list of up to three font faces that can be applied to the text enclosed by the `<font>` tag. Font sizes can be set or changed with the size attribute. The value of this attribute can range from 1 to 7, smallest to largest. If a size is not specified, the default system font size of 3 is used, which is approximately 1 em type size. Note that font sizes are not specified in point sizes. The size of the font can be changed relative to the current size by using a "+" or "-" modifier. That is, an attribute value of `size="+1"` sets the font size to one size larger than the current setting.

You can set a text color by using the `color` attribute. The color value is given as a color name or hexadecimal value as discussed later in this tutorial.

5.1: Font Settings is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

5.2: Text Properties

The previous font settings can be paired with other style sheet properties to apply additional formatting to strings of text. The following table lists these properties that can bring more variety to text displays. Some of the properties pertain to styling the font itself, while others pertain to the structural arrangement of text within its container.

Property:	Values
letter-spacing	Sets the horizontal letter spacing of an element's text. Values are: normal *n*px *n*em
line-height	Sets the line height of an element's text. A measurement in pixels, em's, or points is an actual height; a number or a percentage is a multiplier of the current font size. normal *n*px *n*em *n*pt *n* *n*%
text-align	Sets the horizontal alignment of text within an element. Values are: left center right justify
text-decoration	Sets the decoration of text within an element. Values are: none line-through overline underline blink
text-indent	Sets the indention of text within an element. Values are: *n*px *n*em *n*%
text-shadow	Adds a drop shadow to the text. Values in order include: offset-x offset-y, blur-radius Where offset-x, offset-y, and blur-radius are length values (px, em, etc...), and color is a valid color value. The values offset-x and offset-y are required, while the others are optional.
text-transformation	Sets the capitalization of tex within an element. Values are: none Capitalize lowercase UPPERCASE
vertical-align	Sets the vertical alignment of contents within an element. Values are: bottom top baseline middle sub super text-bottom text-top
white-space	Sets how white space inside an element is treated. Values are: **normal**

Property:	Values
	Sequences of whitespace are collapsed, including newline characters. Lines are broken as necessary to wrap the text inside the element. **pre** White space is not collapsed but preserved, newlines are treated as newlines. **nowrap** Treats whitespace the same as normal, but suppresses line breaks within text. **pre-wrap** White space is not collapsed but preserved, newlines break lines, and lines are broken as necessary to wrap the text inside the element. **pre-line** Whitespace is collapsed, but newlines break lines, and lines are broken as necessary to wrap the text inside the element.
word-spacing	Sets the amount of spacing between words in an element. Values are: normal *n*px *n*em

Figure 5.2.1: Text style properties and values. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

You have already been introduced to the text-align and text-indent properties. The former controls how individual lines of text are aligned within a block of text; the latter controls the amount of spacing at the beginning of the first line of text.

There are two properties which control the amount of horizontal spacing across a line. The **word-spacing** property gives the amount of space between words; the **letter-spacing** property sets the space between letters. These properties are used to stretch (using positive values) or compress (using negative values) words and letters horizontally by adding or removing space between them.

The amount of vertical spacing in a text block is given by the **line-height** property. Although pixel, em, or point measurements can be used, a numeric value better equates to familiar settings. It is a *multiplier* of the current font size; therefore, a declaration of line-height:1.5 equates to line-and-a-half spacing and line-height:2 equates to double spacing. Note that fractional values are valid. Thus, a setting of line-height:.8 narrows the space between lines to 80% of the normal distance.

There are three text properties that are better thought of as font properties since they affect the style of characters. The **text-decoration** property draws a line over the characters (overline), under the characters (underline), or through the characters (line-through). The **text-transfor**m property converts text to upper-case characters, lower-case characters, or capitalized (capitalize) words.

To give text a drop shadow, the text-shadow property can be used. You must supply the x and y offset (offset-x offset-y) of the shadow, where positive x offsets the shadow to the right, and positive y offsets the shadow downwards. You can optionally provide a blur-radius to increase the blur of the shadow. Finally, you can set a color for your shadow. Text can have more than one drop shadows applied at once using comma separated values. The following is an example of a drop shadow.

```
text-shadow: 3px 3px #ccc;
```

Figure 5.2.2: Example of how to add a drop shadow. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The **vertical-align** property has a number of different uses depending on which tag it is applied to. For example, as a font styling property it can convert characters to superscript (super) or subscript (sub) notation. As a text alignment property, it positions text relative to its container or relative to surrounding content. Examples of this property are given when discussing particular tags or styling needs.

Applying Text Properties

You can see the effects of some of these text settings in the page below and displayed in Figure 5.2.3. Output is designed to resemble a typed report.



Figure 5.2.3: Using text styling to produce a report. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Most of the styling appears in an embedded style sheet defining style classes which are applied with `<span>` and `<div>` tags. In-line style sheets are used in one particular section of the page for one-time styles that are not shared by other sections of the document. The default format of the page is given by the selector for the `<body>` tag: 1 1/2 line spacing, paragraph indention, and justified text alignment.

```
<!DOCTYPE html>

    <html lang="en">
    <head>
      <meta charset="utf-8">
      <title> Cognitive Profile </title>

    <style>
     body      {line-height:1.5; text-indent:20px; text-align:justify}
     .head1    {text-align:center; letter-spacing:1.5px; font:bold 1.4em;}
     .head2    {text-indent:0px; font:bold 1.2em;}
     .uline    {text-decoration:underline}
     .ital     {font-style:italic}
     .offset   {margin-left:20px; text-indent:0px; line-height:normal}
     .line     {text-indent:0px}
     .super    {font-size:0.7em; vertical-align:text-top}
     .footnote {line-height:normal; text-indent:20px}
    </style>

    </head>
    <body>

    <div   class="head1" > The Cognitive Profile Model </div>

    <p> The  <span class="uline">  Cognitive Profile Model </span>  is a self
   assessment instrument designed to help students identify their cognitive
   styles. It provides a measure of how students think, learn, and make
   decisions. </p>

    <p> The Model classifies cognitive styles along two bi-polar dimensions:
   Sensor-Intuitive and Thinker-Feeler. The quadrants formed by these two
   intersecting dimensions are labeled Sensor Thinker (ST), Sensor Feeler
   (SF), Intuitive Thinker (NT), and Intuitive Feeler (NF). Scores from the
   assessment are plotted on this grid as a point value in each of the
   quadrants. Connecting the points produces a quadrilateral shape that shows
   a combination of preferred learning styles. </p>

    <div class="offset">
      You probably have a dominant quadrant, or one quadrant in which you have
      the most area (your highest number). You can see that you have some
      area/preference/ability in each type of thinking, and you can strengthen
      those abilities by learning skills in each of those areas. <span class="super"> 1 </span>
    </div>

    <p>  <span class="head2"> Assessment Inventory </span>  </p>

    <p> The inventory consists of 60 pairs of words for which students choose the
   one that appeals the most or that describes them best. Examples of word pairs
```

```
include: </p>

 <div class="offset">
   <span style="width:  80px; text-align:Left">
    Practical <br />
    Facts <br />
    Doing <br />
   </span>
   <span style="width:  100px; text-align:center; font-weight:bold">
    4 3 2 1 1 2 3 4 <br />
    4 3 2 1 1 2 3 4 <br />
    4 3 2 1 1 2 3 4 <br />
   </span>
   <span style="width:  80px; text-align:Right">
    Emotional <br />
    Feelings <br />
    Talking <br />
   </span>
 </div>

 <p> A value between 1 (weak preference) and 4 (strong preference) is assigned
to one of the word pairs. The values are summed across all inventories and
plotted on the quadrant to produce the cognitive profile. </p>

 <div>  <span class="line">  _________________ </span>  </div>
 <div class="footnote">
   <span class="super"> 1 </span>  Krause, Lois Breur,  <span class="ital"> How We
  Learn and Why We Don't </span> , Cincinnati: Thomson Learning, 2000 (p. 19).
 </div>

 </body>
 </html>
```

Figure 5.2.4: Code to produce a report. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

A comment is probably in order about the following section of output, reproduced here with borders around the `<div>` and `<span>` areas to make them more obvious.

Practical
Facts
Doing
4 3 2 1 1 2 3 4
4 3 2 1 1 2 3 4
4 3 2 1 1 2 3 4
Emotional
Feelings
Talking

Figure 5.2.5: Embedded spans within a division. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

By default, a `<div>` tag extends the width of the page as shown by the outer border area enclosing the three blocks of text. Being a block-level tag the `<div>` tag also causes a line break to set off its content from surrounding text.

```
<div class="offset">
  <span style="width:80px; text-align:left">
   Practical <br />
   Facts <br />
   Doing <br />
  </span>
  <span style="width:100px; text-align:center; font-weight:bold">
   4 3 2 1 1 2 3 4 <br />
   4 3 2 1 1 2 3 4 <br />
   4 3 2 1 1 2 3 4 <br />
  <span style="width:80px; text-align:right">
   Emotional <br />
   Feelings <br />
   Talking <br />
  </span>
</div>
```

Figure **5.2.6**: Code to embed spans within a division. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

A `<span>` tag is an in-line tag with no line breaks or spaces before or after. Therefore, multiple `<span>` tags coded one after another are displayed side by side by the browser. Although the words and numbers inside the `<span>` tags appear on separate lines, the contents of the tags themselves appear side by side inside the enclosing division.

In this example the `<span>` tags are also given fixed widths and various alignments to space their contents evenly. This is an example of using the width property, used earlier to size horizontal rules and graphic images, to set exact dimensions for a block of text. Further examples of sizing text blocks are presented in a later tutorial.

Using Negative Measurements

When using pixel measurements it is often necessary to supply *negative* values to achieve desired results. The paragraphs shown in Figure **5.2.7** use -20px text indentions to produce an "underhung" paragraph format often used for bibliographic entries in reports.



Figure **5.2.7**: Using text styling to produce bibliographic entries. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

As shown in Figure **5.2.8**, styling is given by an enclosing `<div>` tag which applies the .biblio style class to its paragraphs. The left margin for all paragraphs is set to 20px. This setting indents the entire paragraph so that its -20px first-line indent extends the line to the left of the margin.

```
<style type="text/css">  \
 .head   {text-align:center; font-weight:bold; font-size:1.2em;}
 .biblio {margin-left:20px; text-indent:-20px}
 .book   {font-style:italic}
</style>

<div class="head">Bibliography</div>

<div class="biblio">
```

Figure 5.2.8: Code to produce underhung paragraphs. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Letter and Word Spacing

Recall that both letter-spacing and word-spacing properties can take on either positive or negative values. The lines in Figure 5.2.9 are set to different letter- and word-spacing values for comparison of effects.

This is a normal letter-spaced sentence.
This is a 2px letter-spaced sentence.
This is a -1px letter-spaced sentence.

This is a normal word-spaced sentence.
This is a 5px word-spaced sentence.
This is a -1px word-spaced sentence.

Figure 5.2.9: Positive and negative letter and word spacing.("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

5.2: Text Properties is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

5.3: Font Style Tags

Text characters can take on styles such as **bold**, *italic*, underscore, and others to highlight or emphasize letters and words. Although these styles can be achieved with style sheet settings, there are also stand-alone tags that can be used to directly enclose and format text characters. In an older version of HTML, these tags were used purely stylistically without imparting meaning to their content; however, HTML5 has re-purposed many tags to give semantic meaning to the content they enclose. As such, these tags should only be used in their proper context, and not simply for their stylistic properties. These tags are known as phrase elements.

Phrase Elements

Phrase Elements, or logical style tags, have standardized meanings in all browsers. One reason for using them alongside CSS is to provide accessibility features for individuals with disabilities. For example, people who are visually impaired may not benefit from simply styling a word through CSS in bold font because it renders text visually in bold characters, but does not ensure that a visually impaired individual will be able to see it. On the other hand, by using the visually comparable `<strong>` logical style tag, a person using special reader software *hears* the text with audio emphasis.

Logical Style Tag	Purpose	Browser Display
`<abbr>abbreviation</abbr>`	Indicates an abbreviated form or acronym. You may include the full text version through the use of the `title="` attribute.	abbreviation
`<b>bold</b>`	Used to mark text which attention is drawn to but without any added emphasis or importance.	**bold**
`<cite>citation</cite>`	Contains a citation or a reference to other sources.	*citation*
`<code>code</code>`	Designates a fragment of computer code.	`code`
`<dfn>definition</dfn>`	Indicates that this is the defining instance of the enclosed term.	*definition*
`<em>emphasis</em>`	Indicates emphasis.	*emphasis*
`<i>italic</i>`	Indicates text set in a different mood in a manner indicating a different type of text such as a technical term or transliteration.	*italic*
`<kbd>keyboard</kbd>`	Represents user input, typically through a keyboard.	`keyboard`
`<mark>mark</mark>`	Indicates a selection of text that is highlighted or marked in a document for reference.	mark
`<q>quote</q>`	Denotes text which is quoted from another source.	“quote”
`<samp>sample</samp>`	Designates sample output from computer programs, scripts, etc.	`sample`
`<small>small</small>`	Used to denote small print side comments, such as copyrights or disclaimers.	small
`<s>strikeout</s>`	Indicates text which is no longer accurate or relevant.	~~strikeout~~
`<strong>strong</strong>`	Indicates stronger emphasis.	**strong**
`<sub>subscript</sub>`	Creates a subscript, and should only be used where the subscript is vital to meaning of	subscript

Logical Style Tag	Purpose	Browser Display
	the text.	
`<sup>superscript</sup>`	Creates a superscript, and should only be used where the superscript is vital to meaning of the text.	superscript
`<var>variable</var>`	Indicates an instance of a variable or program argument.	*variable*

Figure **5.3.1**: Logical style tags. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The logical style tags are designed for situations in which the visual appearance of text in the browser needs to be augmented by other special renderings or information content. Using them where appropriate, helps promote web accessibility for everyone. It should be noted that these tags can also have style properties associated with them. For example, the style sheet declaration code `{color:red}` associates a color with the code selector. Therefore, any text marked as computer code with the `<code>` tag also displays in red.

Style Sheet Alternatives to Headings

Recall that the `<hn>` heading tags enclose strings of text for display in one of six heading styles. The number `n` in the tag ranges from 1 (largest) to 6 (smallest). Headings are displayed in bold characters in the default font face. Of course, you can apply a style sheet to a heading tag to modify its settings. For instance, the style declaration

```
<style>
  h2 {font-family:arial; font-style:italic}
</style>
```

Figure **5.3.2**: Styling a heading tag. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

displays the heading at size 2, but with italicized Arial font face overriding the normal heading display in Times New Roman. Other styles can be applied to the tag to display the heading with additional style characteristics that augment standard display of `<h2>` headings.

In the same way that logical font tags can be emulated with style sheets, so can headings be 'created with style sheets rather than using the special `<hn>` tags. The following stylesheet declares style classes identical to the six heading types. This coding is displayed in the browser as shown in Figure **5.3.3**.

```
<style>
  .head1 {font:2em bold}
  .head2 {font:1.5em bold}
  .head3 {font:1.125em bold}
  .head4 {font:0.95em bold}
  .head5 {font:0.8em bold}
  .head6 {font:0.625em bold}
</style>

<p class="head1">This is Heading Level 1</p>
<p class="head2">This is Heading Level 2</p>
<p class="head3">This is Heading Level 3</p>
<p class="head4">This is Heading Level 4</p>
<p class="head5">This is Heading Level 5</p>
<p class="head6">This is Heading Level 6</p>
```

Figure **5.3.3**: Styling heading tags. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Browser display of headings created with style declarations:

This is Heading Level 1

This is Heading Level 2

This is Heading Level 3

This is Heading Level 4

This is Heading Level 5

This is Heading Level 6

There are several alternatives for creating headings. You can either use `<hn>` tags (where n is a number 1-6) with their default stylings, tags with modified stylings, or forego tags altogether and create your own heading classes. No one option is a best solution in all cases.

Group Selectors

Major and minor headings on a Web page should share common styling as visual clues to their common function as section breaks in page content. While differing in sizes and alignments, headings often have the same font faces and colors. Use of `<hn>` tags makes it easy to control heading styles within embedded style sheets through use of **group selectors**.

A group selector is two or more tag selectors, *separated by commas*, that share in the same styling. For example, the following style sheet declares a common font face and color for a set of `<hn>` tags, and establishes a common visual style for all headings used on the page.

```
<style>
  h1, h2, h3 {font-family:verdana; color:goldenrod}
</style>
```

Listing 4-14. *Using group selectors to style a group of tags.*

The selector h1, h2, h3 is a group selector. This means that the associated style declarations apply to all tags in this list. Assume further that `<h1>` headings are centered for major content breaks, that `<h2>` and `<h3>` headings are left aligned, and that `<h3>` headings are displayed in italic style. These additional group and individual stylings can be added to the above style sheet to produce the display shown in Figure 4-14.

```
<style>
  h1, h2, h3 {font-family:verdana; color:goldenrod}
  h1         {text-align:center}
  h2, h3     {text-align:left}
  h3         {font-style:italic}
</style>
```

Listing 4-15. *Using group selectors to further style groups of tags.*

Major Heading

Side Heading

Side Heading

Figure 4-14. Browser display of headings styled with group selectors.

Group selectors are no different in effect from simple selectors. They just permit a shorthand way to declare shared styles. They also illustrate quite clearly the concept of style inheritance. In the above example, all three headings inherit the same font face and color from their participation in the overall h1, h2, h3 group selector; `<h2>` and `<h3>` headings inherit left alignment from their participation in the h2, h3 group selector. By taking advantage of such inheritance, it saves from having to code separate, duplicate styles for each and every tag selector.

Incidentally, it is not necessary that group selectors be composed of the same tag type. All tags that share the same style can be part of the group selector. For example, the group selector shown in Listing 4-16 indicates that all paragraphs, spans, and divisions on a page share the same type face and size.

```
<style>
  p, span, div {font-family:verdana; font-size:0.8em;}
</style>
```

Listing 4-16. *Using group selectors to style different tags.*

The `<pre>` Tag

Normally, no formatting of text takes place in the browser unless it is surrounded with HTML tags. Although editor code may be indented and line-spaced by using the Space Bar and Enter keys, all contiguous spaces and lines are collapsed to a single blank

space in the browser. There is one case, though, where text as formatted in the editor can be displayed in identical fashion in the browser.

A block of text surrounded by the `<pre>` ("preformatted" or "preserve" spacing) block-level tag is displayed in a monospace font in precisely the way it is formatted in the editor. That is, blank spaces and blank lines used to format HTML code in the editor are faithfully reproduced in the browser, space by space and line by line. The `<pre>` tag is often used to create simple tables where alignment of columns is produced with embedded blank spaces. For example, the following code produces a table that is displayed in the browser identical to the way it is typed in the editor.

```
<pre>
                  Table 1
              Sales by Region
-----------------------------------------------
Region/Year   2000     2001     2002     2003
-----------------------------------------------
East           35.2     37.4     39.8     40.0
West           28.0     25.6     27.4     29.8
South         102.3     86.1     98.6    100.2
North          10.5      8.6      9.8     10.4
-----------------------------------------------
</pre>
```

Listing 4-17. *Using the* `<pre>` *tag to retain editor spacing.*

Since all text lines are surrounded by the `<pre>` tag, columns can be aligned with the Space Bar and line breaks made with the Enter key. This alignment and spacing is possible because both the editor and browser display characters in a monospace font producing equally spaced characters. You need to make sure that your editor uses a monospace font such as Courier New in order to accurately space the text. Browser output is not shown here because it is identical to the table arrangement produced in the editor.

You can include minor styling of the monospaced text as long as the formatting does not change character sizes, thereby changing the amount of space between letters. For example, the following code, dresses up the previous table with bold and italic font styles along with a text color. When practicing, make sure that you align the information in the table properly before adding extra style tags that visually displace the aligned text when viewed in the editor. These extra tags inside the `<pre>` tag do not add spacing to the table when viewed in the browser.

```
<pre style="color:blue">
                  <b>Table 1</b>
              <b>Sales by Region</b>
-----------------------------------------------
<b><i>Region/Year   2000     2001     2002     2003</i></b
-----------------------------------------------
<b>East</b>          35.2     37.4     39.8     40.0
<b>West</b>          28.0     25.6     27.4     29.8
<b>South</b>        102.3     86.1     98.6    100.2
```

```
<b>North</b>          10.5       8.6       9.8      10.4
-----------------------------------------------
</pre>
```

Listing 4-18. *Adding styling to text formatted with the* `<pre>` *tag.*

```
                  Table 1
              Sales by Region
-----------------------------------------------
Region/Year    2000      2001      2002      2003
-----------------------------------------------
East            35.2      37.4      39.8      40.0
West            28.0      25.6      27.4      29.8
South          102.3      86.1      98.6     100.2
North           10.5       8.6       9.8      10.4
-----------------------------------------------
```

Figure 4-15. *Browser display of styled table formatted with* `<pre>` *tag.*

Although the above examples show coding for a data table, a later tutorial describes use of formatting tags especially designed for laying out tables. Although these table tags are the most common way to display rows and columns of tabular information, you can use the `<pre>` tag as a quick and easy way to accomplish similar displays.

5.3: Font Style Tags is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

5.4: Text and Background Colors

Browsers display black text on a white background unless different color style settings are applied. A background color for the page or for any text container can also be specified. Text and background colors can be designated in style sheets using hexadecimal values, RGB values, or color names.

Hexadecimal Colors

```
#rrggbb
```

A value of hexadecimal "00" (the smallest 2-digit hexadecimal value and equivalent to decimal 0) in the color position means the absence of the color; a value of "FF" (the largest 2-digit hexadecimal value and equivalent to decimal 255) means the color is at full intensity. Thus, a hexadecimal value of "#FF0000" represents the color red: "FF" in the "*rr*" position assigns full intensity to red; "00" in the "*gg*" and "*bb*" positions represent the absence of green and blue color intensities.

Likewise, a value of "#00FF00" is (lime) green and a value of "#0000FF" is blue. White is represented by all colors at full intensity: `"#FFFFFF"` ; black is the absence of color: `"#000000"` . In between these extremes are colors of every hue and intensity to over 16 million colors although not all colors can be displayed accurately on all computer monitors.

Some colors can also be represented in the shortened three digit format of `#rgb` . For example, the color #F00 is equal to the color #FF0000, and the color #1BF is the same as #11BBFF.

RGB Colors

An **RGB** color specification uses decimal values to indicate the combined intensities of red, green, and blue hues. Values can range from 0 (smallest) through 255 (largest). These RGB values are given in style sheets in the following format,

```
rgb(r,g,b)
```

using a decimal value to indicate the intensities of the three colors. Thus, coding an RGB value of rgb(255,0,0) indicates red, rgb(0,255,0) indicates (lime) green, and rgb(0,0,255) indicates blue. As in the case with hexadecimal values, a full spectrum of over 16 million colors can be represented by combinations of red, green, and blue intensities.

Color Names

Colors can also be specified with special color names. Most browsers recognize over 200 color names. The common set of 16 Windows colors is shown below by color names and their associated hexadecimal and RGB values.

Color Name	Hex Value	RGB Value
Black	#000000	rgb(0,0,0)
Maroon	#800000	rgb(128,0,0)
Green	#008000	rgb(0,128,0)
Olive	#808000	rgb(128,128,0)
Navy	#000080	rgb(0,0,128)
Purple	#800080	rgb(128,0,128)
Teal	#008080	rgb(0,128,128)
Silver	#C0C0C0	rgb(192,192,192)
Gray	#808080	rgb(128,128,128)
Red	#FF0000	rgb(255,0,0)
Lime	#00FF00	rgb(0,255,0)
Yellow	#FFFF00	rgb(255,255,0)
Blue	#0000FF	rgb(0,0,255)
Fuchsia	#FF00FF	rgb(255,0,255)
Aqua	#00FFFF	rgb(0,255,255)
White	#FFFFFF	rgb(255,255,255)

Figure **5.4.1**: Window color names and Hex and RGB values. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Color Styles Properties

Colors can be applied either to the text on a page or to the background of the container holding the text. These two style properties are shown in the following table.

Color Name	Hex Value
color	*#rrggbb* rgb(r,g,b) *color name*
background-color	*#rrggbb* rgb(r,g,b) *color name*

Figure **5.4.2**: Color style properties. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Colors can be applied as broadly as the entire page by associating a color style with the `<body>` tag down to a single text character by associating a color style with a `<span>` tag surrounding the character. Hexadecimal, RGB, and color-name values can be used in any combination for specifying color properties.

Variations of background and foreground colors are shown in the following code. Browser display for this page appears in Figure **5.4.4**.

```
/PE html>

lang="en">


a charset="utf-8">
le>Color Page</title>

>
        {color:white; background-color:silver}
Astyle   {color:rgb(0,0,0)}
ASEstyle {color:#0000FF}
Tstyle   {color:green; background-color:yellow}
e>

>


s page contains three paragraphs. The entire page is styled with a background color of silver and a foreground
ce. This first paragraph uses these page default color settings.</p>

ss="PARAstyle">
econd paragraph sets its own text color to black using an RGB value. Within this paragraph
class="PHRASEstyle">this string of text is changed to blue, while the phrase <span class="TEXTstyle">
en on yellow</b></span> has its own color style setting.</span></p>

s third paragraph reverts to the default color settings for the page.</p>

>
>
```

Figure 5.4.3: Code to apply color style properties to a page. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)



Figure 5.4.4: Application of color style properties. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The entire page has a silver background with white lettering given by the style setting for the body selector. Three style classes are defined for application to the second paragraph in order to override these page settings. The PARAstyle class is applied to the paragraph to set its text to black [rgb(0,0,0)]; the PHRASEstyle class is applied to a portion of the paragraph to change its text color from black to blue [#0000FF]; the TEXTstyle class is applied to a substring of text to display green letters on a yellow background.

Of course, you are not likely to employ so many different colors on your pages since excessive use of colors can detract from the informational content on the page; however, the above example gives you an idea of how flexible it is to apply foreground and background colors to page elements.

Transparency

In addition to colors, elements can be given a transparency value through the CSS property opacity. The opacity property can be assigned a number value between 0 and 1, with 0 being fully transparent (invisible), and 1 being fully opaque (solid). For example, the following text is fully black (#000000), but has an opacity value of 0.5 which makes it halfway transparent. Since it is on a white background, it will appear grey.

```
opacity: 0.5; color: #000000;
```

Halfway transparent text

While this effect may seem somewhat trivial (being that you could have simply assigned the text to be grey), opacity can serve a variety of uses in creative web design. It can apply to a variety of elements such as images, and can be used when objects overlap each other with different CSS techniques.

Internet explorer version 8 and less do not support the opacity property. These older browsers use the property filter. The filter property uses a functional notation to set the opacity of an object, where 0 is completely transparent (invisible) and 100 is completely opaque (solid). For example, setting an element with an opacity value of 50% like in the previous example would look like the following:

```
filter: alpha(opacity=50);
```

It can be helpful to include both the opacity and filter properties in order to provide maximum browser compatibility.

Gradient - TOP

CSS allows for more than just solid background colors. Color gradients may be used as well.

Gradients are smooth transitions from one color to another. They can be added to a web page using CSS and setting color stops, which tell the gradient which colors to transition to and from. A minimum of two color stops are required for any given gradient.

There are two types of gradients, linear and radial, which can transition from any set number of color stops and from any direction: top to bottom, left to right, even diagonal. Color stops can be specified by color names, hexadecimal code, or rgb values for any desired color. To increase the number of colors in a gradient, add more color stops in the order you wish for them to appear.

Gradients have not always been supported by the major browsers. Older browsers may have compatibility issues with gradients. These issues should not occur on browsers later than: Internet Explorer 10, Google Chrome 26.0, Firefox 16.0, Safari 6.1, and Opera 12.1. Earlier versions of Google Chrome, Firefox, Safari, and Opera supported gradients with an appropriate prefix in the CSS. Our examples will depict these prefixes.

Linear Gradients

Linear gradients transition in a straight line from one side of the container to the other. The default direction is from the top to the bottom of the container. If you do not specify a starting direction, it is assumed the gradient begins at the top.

Here is an example of a linear gradient with two color stops: #3399FF and #FF6600. This example shows the prefixes for older browser compatibility.

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Linear Gradient</title>

<style>
        #linearGradient
        {
                background: -webkit-linear-gradient(top, #3399FF, #FF6600);
                background: -o-linear-gradient(top, #3399FF, #FF6600);
                background: -moz-linear-gradient(top, #3399FF, #FF6600);
                background: linear-gradient(top, #3399FF, #FF6600);

                width: 200px;
                height: 200px;
        }
</style>

</head>
<body>

<div id="linearGradient">

</div>

</body>
</html>
```

Figure **5.4.5**: Code to apply a linear gradient. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)



Figure 5.4.6: Example of a linear gradient. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Radial Gradients

Radial gradients transition in all directions from a given point in their container. These work in much of the same way as linear gradients but are declared differently. By default, a gradient starts in the center of the container and the shape is determined by the width and height of the container.

For example, if the container is a perfect square, the gradient will be a perfect circle. If the container is a rectangle, the gradient will appear to be more of an oval shape.

Here is an example of a radial gradient with two color stops: #3399FF, #FF6600, and #33CC33. This example shows the prefixes for older browser compatibility.

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Radial Gradient</title>

<style>
#radialGradient
{
        background: -webkit-radial-gradient(#3399FF, #FF6600, #33CC33);
        background: -o-radial-gradient(#3399FF, #FF6600, #33CC33);
        background: -moz-radial-gradient(#3399FF, #FF6600, #33CC33);
        background: radial-gradient(#3399FF, #FF6600, #33CC33);

        width: 200px;
        height: 200px;
}
</style>

</head>
<body>

<div id="radialGradient">

</div>

</body>
</html>
```

Figure 5.4.7:Code to apply a radial gradient. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

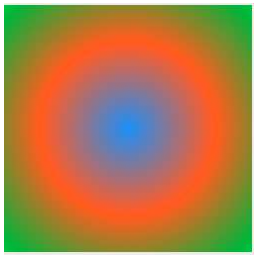


Figure 5.4.8: Example of a radial gradient. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

5.4: Text and Background Colors is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

5.5: Special Characters

The previous font settings can be paired with other style sheet properties to apply additional formatting to strings of text. The following table lists these properties that can bring more variety to text displays. Some of the properties pertain to styling the font itself, while others pertain to the structural arrangement of text within its container.

Display Character	Name	Code	Description
"	"	"	Quote
&	&	&	Ampersand
'		'	Apostrophe
<	<	<	Less than
>	>	>	Greater than
™	™	™	Trademark
			Non-breaking space
¢	¢		Cents
¦	¦	¦	Broken vertical bar
§	§	§	Section
©	©	©	Copyright
«	«	«	Left angle quote
»	»	»	Right angle quote
¬	¬	¬	Not sign
®	®	®	Registered trademark
°	°	°	Degree
±	±	±	Plus/minus
¶	¶	¶	Paragraph
·	·	·	Middle dot
•	•	•	Bullet
¼	¼	¼	Fraction one-quarter
½	½	½	Fraction one-half
¾	¾	¾	Fraction three-quarters
÷	÷	÷	Division
×	×	×	Multiplication
ø	ø	ø	Small o-slash
Ø	Ø	Ø	Large O-slash
–		–	En dash
—		—	Em dash

Figure 5.5.1: Special character names and codes. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)BY-SA 4.0)

As an example of using these special characters, the following code leaves five spaces between words by using non-breaking space ( ) and bullet (•) characters between them.

```
THERE  &bull;  ARE  &bull;  
FIVE  &bull;  SPACES  &bull;  
BETWEEN  &bull;  THESE  &bull;  
WORDS.
```

Figure 5.5.2: Using special characters for Web page display. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

```
THERE  •  ARE  •  FIVE  •  SPACES  •  BETWEEN  •  THESE  •  WORDS.
```

Figure \(\PageIndex{3}\): Browser display of hard spaces and bullets. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Displaying HTML

If you need to display HTML tags as part of the text on a Web page, you cannot type "<" and ">" symbols. These characters are interpreted by the browser as enclosing HTML tags and would be treated as such. Instead, you must use the special HTML characters "<" and ">" to display these symbols. Assume that you wish to display the following HTML code on a Web page.

```
<p>This is a paragraph in which the word <span class="red">RED</span>
is displayed in red by surrounding it with a <span> tag to which a
style class is applied.</p>
```

Figure 5.5.4: Browser display of HTML code. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The above output needs to be entered in the text editor using special characters in place of < and > symbols as shown below:

```
<pre style="font-size:0.9em">
&lt;p&gt;This is a paragraph in which the word <span class="red">&lt;span class="red"&gt;</span>
RED&lt;/span&gt; is displayed in red by surrounding it with a &lt;span&gt;
tag to which a style class is applied.&lt;/p&gt;
</pre>
```

Figure 5.5.5: Code to display HTML tags in the browser. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Styling Character Codes

You should note that the special character codes can have styles applied to them just as other alphanumeric characters. For example, the following equation is displayed in 1.8em bold Courier New font.

```
<style>
  .equation {font-family:courier new; font-size:1.8em; font-weight:bold}
</style>

<p class="equation">&frac14; + &frac12; = &frac34;</p>
```

Figure 5.5.6: Applying styles to character codes. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

¼ + ½ = ¾

Figure 5.5.7: Character codes with styles applied. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

5.5: Special Characters is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

6: Graphic Images

6.1: Graphic Formats

6.2: Displaying Images

6.3: Background Images

6.4: Image Transparency

6: Graphic Images is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

6.1: Graphic Formats

Graphic images can be used to enhance the look of Web pages as well as to provide content that supports the textual information on a page. When used judiciously, images can be attractive and informative; when used to excess, they can be distractive and bothersome. When choosing images to place on a page you need to make sure that they support the purposes of the page and that they do not detract from them. Plus, you need to make sure that the file sizes of images, many of which can become quite large, do not cause unnecessarily long download times.

GIF Image Format

GIF (Graphics Interchange Format) is the most widely supported graphics format. Pictures saved in this format have the **.gif** file extension. GIF format can display images in black and white, grayscale, or color. When used for color pictures the GIF format is limited to displaying up to 256 colors. Normally when a graphics program saves an image in GIF format the software uses (up to) 256 colors that best represent the colors in the picture. Because of the compression technique used for GIF images, this format is best for pictures with spot colors rather than continuous colors. In other words, this is a good format for line drawings, logos, icons, text, and other images with discontinuous colors; however, it is not the best format for photographic images.

One of the concerns about using graphics on a Web page are the file sizes produced by the format. Larger file sizes mean longer download times that cause visitors to wait longer than necessary before being able to view the page. In general, it is not the size of the GIF image that affects the size of the file, it is the number of different colors in the image. Therefore, the most effective way of reducing file sizes and reducing download times is to reduce the number of colors in the image.

Most graphic programs permit choices of the number of colors saved with an image. In Figure 5-1 are shown two save options for GIF files permitted by Adobe PhotoShop. The default option on the left uses the full complement of 256 colors. The color pallet for the image is shown in the bottom-right of the picture. It produces a file size of approximately 10.9 KB. On the right is the same image formatted with 16 colors. There are no noticeable differences between the two pictures. The one on the right, however, produces a file size of only 3.5 KB. If you are creating your own images, you should explore techniques that make your file sizes as small as possible without distorting the image or misrepresenting its colors.



Figure **6.1.1**: Options for saving GIF images with varying color palettes. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Transparent Images

One version of the GIF format -- **GIF89a** -- has the capability of producing transparent images. You can specify one particular color in the image which is then rendered transparent when the image is displayed on the page. Most often this is the single background color in the picture. When set to transparent, the background disappears, and leaves only the foreground image displayed against the page background. The following figure shows the transparent effect.



Figure **6.1.2**: Standard and transparent images against a patterned background. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Both the standard and transparent images are created on a white background and are saved as GIF89a files. The "Transparent" image, however, has the color white selected as the transparent color. When the images are displayed on a textured background, the white background of the transparent image is rendered transparent to permit the page background to show through. Of course, if the background color of the page is the same as the background color of the image, then there is no need to make the picture background transparent.

Interlaced Images

Another feature of GIF89a format is its ability to produce **interlaced** images. Normally, when an image is loaded into the browser it is revealed a few lines at a time beginning at the top of the picture. If the file size is large and the connection speed is slow you see the picture revealed a little at a time until the complete picture is downloaded.

If you choose to save your images as interlaced then the entire picture is revealed at increasingly higher resolutions. That is, it is first revealed as a low-resolution version of the entire picture. As more of the picture is downloaded it becomes sharper and sharper as more details are added. Although the time taken to download an interlaced image is the same as for a regular image, it often appears to download faster since the complete, although not final, image is viewable much quicker. Whether you use standard or interlaced images is more a personal preference than technical need.



Figure **6.1.3**: Interlaced image revealed at increasingly higher resolutions. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Animated Images

Multiple single images can be packaged together to produce **animated** GIF images. These are image files containing two or more images that are revealed in a timed sequence.

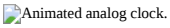


Figure **6.1.4**: An animated GIF Image. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The following Adobe ImageReady screen shows the creation of a few of the 60 images used to produce the above clock whose second hand iterates through 60 seconds. Each hand position is created as a different layer of the image. The software converts the layers to animation frames, or cells that are revealed in sequence when saved as a GIF file.

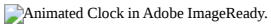


Figure **6.1.5**: Adobe ImageReady screen showing animation cells created from individual layers of a clock image. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

A slightly different animation technique is used in other animation programs. Some GIF animators require the creation of separately saved GIF images. Each image represents a different frame of the animation. These individual GIF files are imported into the software and are converted into sequenced cells of the animation.

An animated GIF file is retrieved by the browser just like any other GIF file. When displayed in the browser, the file produces the animation. Of course, if you are not particularly skilled in working with graphics or you do not have the patience to put together the series of pictures to be animated, you can probably find animated GIF images on the Web that suit your needs.

Image Compression

When images are saved, they are compressed to help reduce redundancy of the image data so that the image can be stored or transmitted in an efficient form. Image compression may be lossless or lossy. With lossless compression nothing in the original image is lost. When viewed in the browser, the image will contain the same number of pixels as the original image. Lossless compression is used when the GIF is saved. In contrast, lossy compression means that some of the pixels in the original image are lost or removed. When viewed in the browser the image will be similar, but not exactly the same as the original image.

PNG Image Format

A newer format that is increasing in popularity is **PNG (Portable Network Graphic)** format, pronounced "ping." This format is used for the same purposes as GIF; however, it produces smaller file sizes and faster loading times without loss of resolution quality. It can reveal interlaced images and produce them faster than GIF format.

The most noticeable difference between PNG and GIF formats occurs with transparent images. PNG format permits up to 254 levels of transparency, and allows images to better blend with the background color or pattern of a page. Figure 5-6 shows the blending differences between PNG and GIF images.

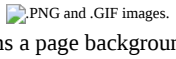


Figure **6.1.6**: Comparison of PNG and GIF formats agains a page background color. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

In the GIF image on the right, transparency of pixels is either "on" or "off," and produces hard edges surrounding what would otherwise be a smooth transition between the drop shadow and full transparency of the image's background. A white "halo"

surrounds the drop shadow at its boundary with full transparency. In contrast, the PNG image on the left blends smoothly with the background color. This results from its multiple levels of pixel transparency whereby pixels surrounding the drop shadow take on increasing levels of transparency the further removed they are from the image.

JPEG Image Format

The **JPEG (Joint Photographic Experts Group)** format is designed for storing photographic images with millions of colors at different compression rates. During compression, graphics programs use special algorithms to sample and render colors close to those in the original picture but without retaining full color information in order to minimize file sizes.

You normally have a choice of compression settings when saving pictures in JPEG format. Smaller file sizes normally mean greater loss of picture details. Still, with moderate compression you can display an image on screen that appears very similar in quality to the original picture. The four pictures below show the original image and three compressions along with resulting file sizes.You can notice a loss of sharpness in those images with higher compressions and smaller file sizes.

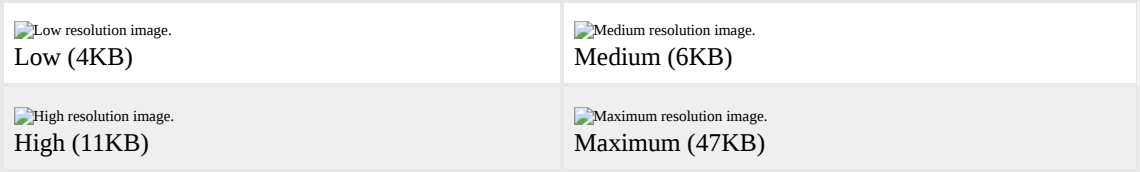


Figure **6.1.7**: JPEG images at various compressions. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

For Web images that are displayed at normal 72 pixels per inch, compression percentages that reduce file sizes to as small as 1/8 to 1/4 of original file sizes still retain satisfactory visual precision.

JPEG images are saved as files with the **.jpg** extension. JPEG format does not support interlacing or transparency; plus, it is not a good format for text or line drawings since the precision needed to produce straight lines or hard edges is not as accurate with compression.

The following table provides a summary of the image file types.

Image Type	Extension	Transparency	Animation	Interlacing	Compression	Colors
GIF	.gif	Yes	Yes	Yes	Lossless	256 Colors
JPEG	.jpg or .jpeg	No	No	Yes	Lossy	Millions of Colors
PNG	.png	Yes	No	Yes	Lossless	Millions of Colors

Sources for Graphics

How do you obtain graphics for your pages? There are many sources for obtaining graphics: create graphics using a graphics application such as Adobe Photoshop, Adobe Illustrator, and Jasc Paint Shop Pro; download graphics from a graphics web site; purchase a graphics collection online; take digital photographs; or scan existing photographs. The following are a few sites that may be helpful when looking for graphics:

- Microsoft Clip Art
- Free Stock Photo Search Engine
- SuperStock
- Free Images
- FreeFoto
- OpenPhoto
- Stock Photography

- Search Engine for Stock Photos

Always use caution when downloading graphics from the Web. Materials on a Web site are copyrighted and are not free to use unless the owner of the site permits it.

6.1: Graphic Formats is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

6.2: Displaying Images

A graphic image is placed on a Web page by making reference to the *location* of a GIF, PNG, or JPEG file. The image file is then downloaded from the Web server along with the page. As the browser displays the page, this reference is used to retrieve the image and place it on the page.

The `<img>` Tag

Graphic images are placed on Web pages by using the `<img>` tag. The general format for this tag is shown in Figure **6.2.1**. Note that the `<img>` tag is not a container tag and does not require a separate closing tag.

```
<img src="url" alt="text">
```

Figure **6.2.1**: General format for `<img>` tag. (Copyright; author via source)

An `<img>` tag appears on the Web page *at the location* at which the image is to be displayed. The **src** (source) attribute gives the location of the image file. This can be a local or remote URL. The **alt** (alternate text) attribute is required under HTML5 standards. It provides a text description of the picture which appears instead of the image if the image cannot be loaded or displayed.

Not only is it a requirement, including the alt attribute provides accommodation for users with visual impairment. Special browser software can verbalize the text string when the picture itself cannot be viewed in detail. Therefore, you should take care to provide alt text that gives an accurate and useful description of the picture. Also, browsers can display the text while waiting for the image to download, thereby giving visitors a text preview of the visual contents of the page. Finally, some people prefer to turn off display of graphic images altogether to speed up page downloads or when printing the page. These pop-up boxes help to identify images that are not otherwise displayed.

Displaying Local Images

An image file can be located in the same directory as the Web page which displays it; or you can create a separate folder for storing images, usually subordinate to the Web page directory. If the file is in the same directory as the Web page, only the file name is needed as the src URL, as shown in the following listing.

```
<p><img src="yoda.jpg" alt="Picture of Yoda"/></p>
```

Figure **6.2.2**: Displaying an image located in the same directory as the Web page.(Copyright; author via source)

Figure **6.2.3** show the display of this image file.

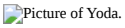


Figure **6.2.3**: Display of image file. (Copyright; author via source)

If the image file is in a different directory from the Web page that displays the picture, then the path to that folder needs to be coded as part of the src URL. In the following example, an image file is located in the Media folder, a subdirectory of the directory containing the Web page that displays the image.

```
<p><img src="Media/yoda.jpg" alt="Picture of Yoda"/></p>
```

Figure **6.2.4**: Displaying an image located in the Media subdirectory of the current page directory. (Copyright; author via source)

Displaying Remote Images

You can connect to and display images that are located anywhere on the World Wide Web without having to download the image for display from a local directory. In this case, a full URL address to the image, beginning with "http://", is required in the src

attribute. Figure **6.2.5** shows a Web page with a link to a remote image file. Note that coding of the `<img>` tag includes reference to the site and to the directory path of the image.

```
<img src="http:  //www.exoticfelineimages.org/20050805/images/2005-08-05%20
    12.03.03.jpg" alt="Picture of an adorable bunny"/>
```

 Figure **6.2.5**: Remote Image linked to and displayed on Web page. (Copyright; author via source)

When linking to remote pictures you should give reference credit. It may be necessary, as well, to get approval from site owners to link to their resources. Since you are relying on others sites to remain accessible, you need to continually check the link to make sure the image is still available.

Image Size

Unless you specify otherwise, an image is displayed at its original width and height. By coding the width and/or height style, you can resize the image for display. If you intend to resize the image proportionately, you need only to code *either* a width *or* height dimension. The browser determines the appropriate size for the other dimension to maintain proper scale. However, you can purposely distort an image by coding disproportionate width and height sizes.

The following image is displayed in two sizes -- its original size and at 1/2 its original size. Only the width is changed to permit the browser to set the picture height to maintain proportionality.

```
<img src="SlowLorisFull.jpg" alt="Picture Of a slow loris"/>
<img src="SlowLoris.jpg" style="width: 60px" alt="Picture of a slow loris"/>
```



Figure **6.2.6**: Images displayed at full size and at 1/2 size. (Copyright; author via source)

When resizing an image, it is best not to attempt to enlarge the picture beyond its original size. Its original size is its best and sharpest resolution. By enlarging the image, "pixelization" may occur which produces loss of sharpness. Most computer monitors display images at 72 dpi (dots per inch). It is usually satisfactory to save and display images at this resolution. Saving images at higher resolutions does not produce noticeable display improvement, and it results in larger file sizes that increase download times.

Image Positioning

As with other page elements, alignment styles can be applied to position a picture on the page. The usual choice is whether to display the picture in-line with surrounding text, on a line by itself, or to float the picture so that accompanying text wraps around it.

Stand-alone Images

If an image appears on a line by itself, then it can be positioned horizontally to the left, to the right, or centered. This positioning occurs by coding the text-align style, not for the image itself, but for a container encompassing the picture, usually a `<p>` or `<div>` tag that separates its content from surrounding text with a blank line or line break.

```
<p style="text-align:center"><img src="Earth.gif" alt="Picture of Earth"/></p>
```

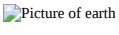


Figure **6.2.7**: Horizontal alignment of an image using a `<p>` tag. (Copyright; author via source)

In the above example, the `<img/>` tag is included inside a paragraph to offset it from surrounding text by blank lines. Paragraph content is centered thereby centering the image contained in the paragraph.

Floated Images

Images can be floated to the left or right of the page -- with word wrap around the picture -- by coding the float:left or float:right style inside the `<img/>` tag. Extra white space can be included around the picture to separate it from surrounding text by increasing its margin settings. The accompanying image is floated to the right and has a left margin setting of 25 pixels to leave space between the image and this paragraph.

```
<p>
<img src="Earth.gif"  alt="Picture of Earth"
style="float Right(); margin-left:25px"/>
... wrapped text ...
</p>
```

Figure **6.2.8**: Code to float an image to the right of the page. (Copyright; author via source)

Remember, when images or other page elements are floated to the left or right of the page with word wrap around them, the `<img/>` tag is coded first followed by the wrap-around text.

In-line Images

An image can appear "in-line," that is, on the same line as its accompanying text. In this case, it needs to be determined how the text is to be aligned with the image. By default, text surrounding an in-line image is aligned with the bottom of the picture. You can, however, apply the vertical-align style property to the `<img/>` tag to reposition surrounding text. Values for the vertical-align property are shown in Figure 5-13.

Property:	Value
vertical-align	text-top middle text-bottom (default)

Figure **6.2.9**: Vertical-align style property. (Copyright; author via source)

Application of the vertical-align style to in-line `<img/>` tags is illustrated in Figure **6.2.10**. Accompanying text is align with the top, middle, and bottom of the pictures.

```
<p>A picture <img src="Stonehenge.jpg" style="vertical-align: Text-top"/>
    with text vertically aligned text-top.</p>

<p>A picture <img src="Stonehenge.jpg" style="vertical-align:middle"/>
    with text vertically aligned middle.</p>

<p>A picture <img src="Stonehenge.jpg" style="vertical-align:text-bottom"/>
with text vertically aligned text-bottom.</p>
```

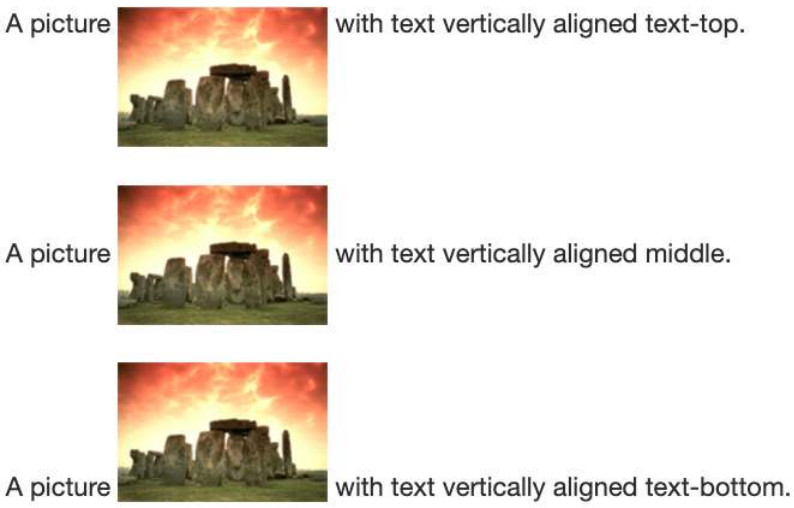


Figure **6.2.10**: In-line images aligned with accompanying text. (Copyright; author via source)

Graphic Text

Recall from an earlier discussion that you are limited to using font faces that are installed on the user's computer system. Irrespective of the fonts you have available to design your pages, they will not display properly unless they are also available to your visitor. Although if you insist on using a particularly font, then you can convert text to a graphic image and place it on the page.

ormally you will create a single letter or word for display in stylish characters. The example letter at the beginning of this paragraph is from the Adobe Rosewood font collection. Most users probably do not have this font installed on their systems. However, it is visible in the browser because it has been converted into a graphic image and placed on the page with an `<img/>` tag that floats to the left of this paragraph.

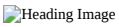


Figure **6.2.11**: Example of a heading image. (Copyright; author via source)

The same technique can be used to create headings. Use your favorite graphics program to style the text and save it as an image, then place it on the page with an `<img/>` tag. Be careful about using too many fancy styles. They can be distractive.

Shortcut Icons (Favicons)

You can use shortcut icons to display your logo or some other small graphic on the Internet Explorer favorite's menu or Firefox bookmark's menu, and address bar. They have no special Web server requirements and are a great way to add brand recognition to your Web site.

A shortcut icon must be square in size, and at least 16 x 16 pixels. Consider creating both a 16 x 16-pixel icon as well as a 32 x 32-pixel icon (and larger, bandwidth permitting). This course of action might be necessary because on high-DPI displays, Internet Explorer may stretch the icon to fit the available space.

The following are popular programs and tools for creating shortcut icons:

Axialis Icon Workshop
Irfan View
FavIcon from Pics

After creating the icon, you must associate it with your Web page. This is done by adding a line of XHTML code to the page's head element. The line of code includes a `<link>` tag that specifies the location and name of the icon file. You can include this `<link>` tag on a per-page basis. First, save the icon with a file name other than favicon.ico, and then add the following code to the head element of your page.

```
<head>
     <link rel="shortcut icon" href="myicon.ico" />
     <title> My Title </title>
</head>
```

Figure **6.2.12**: Code for adding a shortcut icon. (Copyright; author via source)

Deprecated Image Attributes

There are a number of deprecated `<img/>` tag attributes that are being replaced by style settings. You are likely to encounter these attributes on older Web pages, but you are encouraged not to use them on your pages.

```
border="n"
```

A solid black border is drawn around the image with the **border** attribute. The size of the border is given in *n* pixels.

```
align="top|middle|bottom"

align="left|right"
```

One version of the align attribute controls the placement of text surrounding the image. When using align="top", align="middle", or align="bottom", the image appears in-line -- on the same line as surrounding text -- and accompanying text is aligned with the top, middle, or bottom of the picture. These three alignments attributes are identical in effect to the vertical-align style properties text-top, middle, and text-bottom. If alignment is not specified in the `<img/>` tag, the image appears in-line with surrounding text aligned with the bottom edge of the picture.

A second variation of the align attribute controls word-wrap around the image. When align="left" or align="right" is coded, the image appears at the specified left or right margin of the page with accompanying text wrapped around it. The `<img/>` tag must appear immediately before the text that is wrapped around it. These alignments are identical in effect to the float:left and float:right style properties.

```
hspace="n"

vspace="n"
```

You can insert space around an image by using either or both the **hspace** (horizontal spacing) or **vspace** (vertical spacing) attributes and specifying the number of pixels for each. The hspace attribute specifies the number of pixels of blank space to leave on the

left *and* right sides of the image; the vspace attribute specifies the number of pixels to leave at the top *and* bottom of the image. Unlike using margin style settings, you cannot control spacing on individual sides of the image.

```
height="n"

width="n"
```

Image sizes are controlled by **height** and **width** attributes. These work identically to their style property counterparts, using pixel measurements to set image sizes.

6.2: Displaying Images is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

6.3: Background Images

Any HTML container, from the `<body>` tag down through the `<span>` tag, can use graphic images as backgrounds behind their text content. Of course, you need to be judicious in selecting images that do not overpower or detract from the text communication intended by the page.

Image Sources

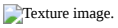


Figure **6.3.1**: Texture image. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

Most background images are **textures** rather than pictures. Pictures tend to be too "busy" and distract attention from the message. These textures are either standard GIF or JPEG images; plus, the images are usually not the full size of the container for which they provide the background. Textures are small, specially designed images that can be repeated across and down a page and fit together seamlessly to form a smooth backdrop.

The accompanying graphic is typical of an image used as a background for page elements. It displays a texture that fits together side by side when the image is repeated horizontally and vertically across and down the background area of the page element. Repeatable textures such as this are sometimes difficult to make, but there are plenty to choose from at most Web clip-art sites.

Background Styles

There are four style properties pertinent to displaying background images behind text containers. These styles, shown in Figure **6.3.2**, select an image, repeat an image, position an image, and affix an image for text over-scrolling. Not all properties are required for all background images.

Property:	Value
background-attachment	Sets how a background image is attached to a document, whether or not the image scrolls with the text. fixed scroll
background-image	Sets the background image of page element specified as url(*url*) none
background-position	Sets the location of the top-left corner of the background image relative to the top-left corner of the container element, using a measurement, a percentage, or a keyword or pair of keywords. *left top* *left% top%* top|right|bottom|left|center top left|top center|top right center left|center center|center right bottom left|bottom center|bottom right
background-repeat	Sets whether background image repeats horizontally, vertically, in both directions, or not at all. repeat-x repeat-y repeat no-repeat

Figure **6.3.2**: Background style properties. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

A background image is specified with the **background-image:url**(*url*) property, where the location of the image file is given by its *url* address. If the image file is stored in the same location as the page, only the file name need be coded. If no other style

settings are made pertaining to the background image, it is repeated horizontally and vertically, and across and down the container to fill the background with the image.

The page shown in Figure **6.3.3** uses the background-image property to fill the page horizontally and vertically with a texture. No other style properties are required.

```
<style>
 body {background-image:url(Background.jpg)}
</style>
```

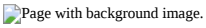


Figure **6.3.3**: Page with texture image used as background. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The Background.jpg image has an actual size of 144 x 144 pixels, or 2" square. With no additional style properties given, the image is automatically repeated across and down the page the fill to available space.

Of course, it is not necessary to use repeatable textures as background images. You can make use of any GIF or JPEG image. You need to be cautious when using regular images as backgrounds since they may have colors that are too intense and make the overlaying text difficult to read. If you have photo editing software available you can reduce the opacity, or transparency, of the image to temper its intensity for use as a "watermark" image. This is done for the image shown in Figure **6.3.4**.

```
<style>
 body {background-image:url(Stonehenge.jpg)}
</style>
```

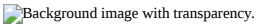


Figure **6.3.4**: Page with repeated background image with reduced opacity. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

As in the case of the previous background texture, the image repeats both horizontally and vertically to fill the page. This repeating pattern is more noticeable when using a regular graphic image.

You can cause one or more background image to appear by coding the **background-repeat** property. The value repeat-x repeats the image horizontally for as many times as needed to fill the container from left to right thereby producing a single row of images; repeat-y repeats the image vertically for as many times as needed to fill the container from top to bottom subsequrntly producing a single column of images; repeat repeats the image in both directions to completely fill the container (the default value); and no-repeat produces a single image in the top-left corner of the container.

If an image is not repeated, you can indicate the location of the single image inside the container by coding the **background-position** property. The following code positions an image in the center of the page with background-position:320px 150px. Browser output is shown in Figure **6.3.5**.

```
<style>
 body {background-image:url(Stonehenge.jpg); background-repeat:no-repeat;
       background-position:320px 150px}
</style>
```

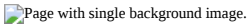


Figure **6.3.5**: Page with single background image positioned at exact pixel location. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The two pixel measurements--320px 150px--give the distance from the left and top of the page to the left and top of the image. In other words, the left edge of the picture is 320 pixels from the left edge of the page, and the top edge of the picture is 150 pixels from the top edge of the page. You may have to try several measurements before you get an image positioned to your satisfaction.

In addition to specifying exact pixel locations for a background image, you can specify certain percentages of page width and page height for placement of the image. Also, various keywords or pairs of keywords can be used to position the image. The keyword top, right, bottom, left, or center positions the image, respectively, at the top-center, right-center, bottom-center, and center of the page. The keyword pairs top left, top center, top right, center left, center center, center right, bottom left, bottom center, or bottom right position the image at these named locations on the page.

It is important to remember that these positions are relative to the full size of the Web page, not to the size of the browser window. Thus, an image positioned at the center or at the bottom of a page may not be visible until the page is scrolled to bring the image into view.

In the above examples, the background image is scrolled when the text on the page is scrolled. For certain types of backgrounds, you may want the image to remain "fixed" so that text scrolls over the fixed background. This effect is accomplished with the **background-attachment** property. Its default value is scroll. Changing it to fixed keeps the background image in the same position while the text foreground scrolls over the top. This effect is shown in Figure **6.3.6**.

```
<style>
 body {background-image:url(Stonehenge.jpg); background-repeat:no-repeat;
       background-position:320px 150px; background-attachment:fixed}
</style>
```



Figure **6.3.6**: Page with fixed background image over which page content scrolls. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The previous examples show background images for an entire Web page; however, any container tag can have a background image. Figure 5-22 shows a paragraph with a background texture.

```
 <style>
p#BKGND {background-image:url(Background.jpg); margin:20px; font-family:
        verdana; font-size:12pt; font-weight:bold; text-align:justify}
 </style>

 <h1> Paragraph Background Texture </h1>

 <p id="BKGND"> This is a paragraph with a background texture. The default
background-image property repeats the background pattern horizontally
and vertically to fill the area of the paragraph. </p>
Paragraph with background image
```



Figure **6.3.7**: Paragraph with background texture. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

6.3: Background Images is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

6.4: Image Transparency

Image Transparency/Opacity

With CSS you can change the transparency/opacity of images to enhance the look of your page. Currently, Internet Explorer 8 and earlier use a different technique for applying image transparency than Firefox, Chrome, and other Internet browsers. Style properties for the browsers are shown in the table below.

Property:	Value	Description
opacity	x x can be a value between 0.0 - 1.0. A lower value makes the element more transparent.	Changes the transparency/opacity of an image. **This property is supported by all major browsers including IE9 and later.**
filter:alpha(opacity=x))	x x can be a value from 0 - 100. A lower value makes the element more transparent	Changes the transparency/opacity of an image. **** Currently this property is used only by IE8 and earlier versions.**

Figure **6.4.1**: Properties and values for image transparency/opacity. ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

The following example demonstrates how to use the transparency properties in both IE and other browsers

```
<body style="background-image:url(stonehenge1.jpg);"
background-repeat:repeat;font:10pt arial">


<div style="width:400px;height: 180px;margin: auto;"
background-color: #ffffff;border: 1px solid black;
padding:10px;filter:alpha(opacity=60);opacity:0.6">

<p>
A background image is specified with the background-image:url(url)
property, where the location of the image file is given by its url
address. If the image file is stored in the same location as the page,
only the file name need be coded. If no other style settings are
made pertaining to the background image, it is repeated horizontally
and vertically, across and down the container, to fill the background
with the image.
</p>
</div>
</body>
```



Figure **6.4.2**: Transparency/Opacity Example ("MGA Web Development Tutorials" by Floyd, Kevin; Kwak, Myungjae; and Stines, Alan, Computer Science and Information Technology Ancillary Materials. 4, GALILEO Open Learning Materials is licensed under CC BY-SA 4.0)

6.4: Image Transparency is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

7: Applying Special Styles

7.1: Styling Lists

7.2: Borders and Padding

7.3: Sizing Elements

7.4: Positioning Elements

7: Applying Special Styles is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

7.1: Styling Lists

Styling Lists

Previously you were introduced to some of the basic style properties used to format page elements. The following sections add to your repertoire of properties by focusing on ways to add special styling to page elements. In this section, various ways of adding special styles to list structures are discussed.

Styling Unordered Lists

```
<ul>
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>List Item 3</li>
</ul>
```

- List Item 1
- List Item 2
- List Item 3

Figure **7.1.1**: An Unordered List

One of the list structures is an unordered list: a series of items preceded by bullet characters and set off from surrounding text by single blank lines. The list is single spaced and indented from the left margin. An example unordered list is coded and displayed below.

Bullet types can be specified by using the deprecated type="disc|circle|square" attribute of the `<ul>` tag. Preferably, style sheet properties should replace this attribute. For unordered lists, there are two style properties that can be used to designate the type of bullet symbol with which to prefix list items. These properties and their associated values are shown in Figure **7.1.2**.

Property	Value
list-style-type	disc circle square none
list-style-image	url(*url*)

Figure **7.1.2**: List-style bullet types.

Specifying Bullet Characters

The unordered list shown in Figure **7.1.3** uses the list-style-type property to display a circle as the bullet type. Styling is given in an embedded style sheet entry for the ul selector.

```
<style type="text/css">
    ul {list-style-type:circle}
</style>

<ul>
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>List Item 3</li>
</ul>
```

Figure **7.1.3**: Code for list-style bullet type.

Of course, by using the simple selector ul in the stylesheet, *all* unordered lists on the page will have circle bullets. It might be preferable to use ID selectors so that different lists can be assigned different bullet characters. The following style sheet identifies three such lists, each with its own

prefix character.

```
<style type="text/css">
    ul#List1 {list-style-type:disc}
    ul#List2 {list-style-type:circle}
    ul#List3 {list-style-type:square}
</style>
```

Figure **7.1.4**: Embedded style sheet for list-style bullet types.

A particular list structure takes on a bullet type by assigning the associated id to its `<ul>` tag. That is, a list identified with the tag `<ul id="List3">` will have square characters as bullets.

Setting Margins for Unordered Lists

Normally, unordered lists are indented a fixed number of pixels from the left margin. In certain cases, you may not wish the list of items to be indented. You can include a margin-left style setting to move items to the left margin (the margin setting for the list equates to the margin setting for its container). In the following code, this margin setting is added to the ul selector to align bullets at a page margin of 20 pixels.

```
<style type="text/css">
    ul {list-style-type:circle; margin-left:20px}
</style>
```

Figure **7.1.5**: Aligning an unordered list at the page margin.

Suppressing Display of Bullet Characters

You may wish to use other than the supplied bullet characters to prefix your list. In the following example, the list-style-type property is set to none to remove display of bullets. Then each list item is prefixed with an arrowhead character from the Webdings font (the numeric character "4" displays the right-arrowhead symbol in this font family).

```
<style type="text/css">
    ul {list-style-type:none; margin-left:20px}
</style>

<ul>
    <li><span style="font-family: webdings">4</span> List Item 1</li>
    <li><span style="font-family: webdings">4</span> List Item 2</li>
    <li><span style="font-family: webdings">4</span> List Item 3</li>
</ul>
```

Figure **7.1.6**: Aligning an unordered list at the page margin.

- List Item 1
- List Item 2
- List Item 3

Figure **7.1.7**: Unordered list with arrowhead characters as bullets.

In this example, the default bullet character is *not replaced* by the arrowhead. Instead, visibility of bullets is suppressed and the arrowhead character is added to the beginning of each list item. Even though bullet display is suppressed, room for the hidden character still occupies space on the line. Therefore, a margin-left setting is needed to move the list items back to their normal positions.

Consider the following list in which the right-arrowhead character has been added in front of the items, but the display of default bullets has intentionally not been suppressed.

```
<style type="text/css">
    ul {margin-left:20px}
</style>

<ul>
    <li><span style="font-family: webdings">4</span> List Item 1</li>
    <li><span style="font-family: webdings">4</span> List Item 2</li>
    <li><span style="font-family: webdings">4</span> List Item 3</li>
</ul>
```

Figure **7.1.8**: Code for unordered list without suppression of bullet character.

- ▸ List Item 1
- ▸ List Item 2
- ▸ List Item 3

Figure **7.1.9**: Unordered list with visible bullets and arrowhead characters.

When list-style-type:none is added to the style sheet, the default disc characters are not displayed; however, the space occupied by these characters remains. In order to move the list and the arrowheads to the left to occupy that space and retain normal indention of list items, the margin-left property is added to the style sheet. A left margin of approximately 20 pixels aligns the right-arrowhead characters at the position previously occupied by the bullets.

Using Graphic Images for Unordered Lists

Rather than using text characters as bullets, you can use graphic images to prefix list items by coding the list-style-image:url(*url*) property. If the image is in the same directory as the page containing the list, then the *url* is simply the name of the graphic file. In the following example an image named Bullet.gif is used to prefix list items. Browser output is shown in Figure **7.1.11**.

```
<style type="text/css">
    ul {list-style-image:url(Bullet.png); vertical-align:middle}
</style>

<ul>
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>List Item 3</li>
</ul>
```

Figure **7.1.10**: Code for unordered list with graphic image as bullets.

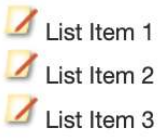


Figure **7.1.11**: Unordered list with graphic image as bullets.

Depending on the size of the image you may have to adjust vertical alignment of the list items to align properly with the image. Normally, text aligns at the bottom of an in-line image. In this example the vertical-align:middle property is applied to the image to align its accompanying text with the middle of the image. Also, text-top and text-bottom values can be used to vertically align text with the top or bottom of an image.

Styling Ordered Lists

An ordered list is a series of items preceded by sequence numbers and set off from surrounding text by single blank lines. The list is single spaced and indented from the left margin in the same way as an unordered list. A default ordered list is coded and displayed as shown below.

```
<ol>
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>List Item 3</li>
</ol>
```

Figure 7.1.12: Code for an ordered list.

1. List Item 1
2. List Item 2
3. List Item 3

Figure 7.1.13: An ordered list.

Style sheet settings can be used to designate the type of numbering symbol to prefix list items. The list-style-type property can be applied using the values shown in Figure 6-8. The decimal value is default.

Property	Value
list-style-type	decimal upper-alpha lower-alpha upper-roman lower-roman none

Figure 7.1.14: List style types of numbering symbols.

Specifying Number Characters

The following ordered list is numbered with upper-roman numerals. Browser output is shown in Figure 7.1.16.

```
<style type="text/css">
    ol {list-style-type:upper-roman}
</style>

<ol>
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>List Item 3</li>
<ol>
```

Figure 7.1.15: Code for an ordered list with upper-roman characters.

I. List Item 1
II. List Item 2
III. List Item 3

Figure 7.1.16: An ordered list with Roman numeral numbering.

Nested Ordered Lists

Ordered lists can be nested inside each other with each list having its own numbering scheme. In the following example, an outer list is numbered with upper-case Roman numerals and an inner list is numbered with lower-case Roman numerals. Numbering characters are assigned to particular lists by using ID selectors.

```
<style type="text/css">
    ol#List1 {list-style-type:upper-roman}
    ol#List2 {list-style-type:lower-roman}
</style>

<ol id="List1">
    <li>List Item 1</li>
    <li>List Item 2
        <ol id="List2">
            <li>List Item 2a</li>
            <li>List Item 2b</li>
        </ol>
    </li>
    <li>List Item 3</li>
<ol>
```

Figure **7.1.17**: Code for nested ordered lists.

I. List Item 1
II. List Item 2
    i. List Item 2a
    ii. List Item 2b
III. List Item 3

Figure **7.1.18**: Nested ordered lists with Roman numeral numbering.

Styling Definition Lists

A definition list is a series of terms and definitions offset from surrounding text by a single blank line. The terms in the list are blocked at the left margin; definitions are indented and words wrapped on the following lines.

Recall that a definition list is enclosed inside `<dl>` tags and contains one or more `<dt>` tags listing the terms to be defined. Each term has an associated `<dd>` tag surrounding its definition. An example definition list is coded below and displayed in Figure **7.1.20**.

```
l>
  <dt>Term 1</dt>
      <dd>This is the Term 1 definition. The definition term appears on a line by itself and is follow
              by a definition text block. The definition is indented and word wrapped.</dd>
  <dt>Term 2</dt>
      <dd>This is the Term 2 definition. The definition term appears on a line by itself and is follow
              by a definition text block. The definition is indented and word wrapped.</dd>
dl>
```

Figure **7.1.19**: Code for a definition list.

Term 1

This is the Term 1 definition. The definition term appears on a line by itself and is followed by a definition text block. The definition is indented and word wrapped.

Term 2

This is the Term 2 definition. The definition term appears on a line by itself and is followed by a definition text block. The definition is indented and word wrapped.

Figure **7.1.20**: A definition list.

There are no style properties specifically designed for definition lists. Fortunately, you can apply other formatting styles to a list to give it a different look and alignment. In the following example, the above list is given additional spacing between items by applying margin settings to the <dd> tags.

```
tyle type="text/css">
  dd {margin-top:10px; margin-bottom:10px}
style>

l>
  <dt>Term 1</dt>
      <dd>This is the Term 1 definition. The definition term appears on a line by itself and is follow
              by a definition text block. The definition is indented and word wrapped.</dd>
  <dt>Term 2</dt>
      <dd>This is the Term 2 definition. The definition term appears on a line by itself and is follow
              by a definition text block. The definition is indented and word wrapped.</dd>
dl>
```

Figure 7.1.21: Code for a definition list with margin settings.

Term 1

This is the Term 1 definition. The definition term appears on a line by itself and is followed by a definition text block. The definition is indented and word wrapped.

Term 2

This is the Term 2 definition. The definition term appears on a line by itself and is followed by a definition text block. The definition is indented and word wrapped.

Figure 7.1.22: Copy and Paste Caption here. (Copyright; author via source)

Contextual Selectors

List structures present a good opportunity to discuss an additional method -- an addendum to ID selectors -- to apply selective styling to XHTML elements. Consider this scenario for two lists requiring different stylings: the list items in an unordered list are displayed in a different color from those in an ordered list.

The two lists *could* apply different color settings through their tag selectors. That is, the ul selector could take on a different color from the ol selector with the following code.

```
<style type="text/css">
    ul {color:red}
    ol {color:blue}
</style>

<ul>
    <li>Red Item 1</li>
    <li>Red Item 2</li>
    <li>Red Item 3</li>
</ul>

<ol>
    <li>Blue Item 1</li>
    <li>Blue Item 2</li>
    <li>Blue Item 3</li>
</ol>
```

Figure **7.1.23**: Code to display unordered and ordered lists in different colors.

- Red Item 1
- Red Item 2
- Red Item 3

1. Blue Item 1
2. Blue Item 2
3. Blue Item 3

Figure **7.1.24**:
Unordered and ordered lists displayed in different colors.

The `<ul>` list is displayed in red and the `<ol>` list is displayed in blue. An alternative is to focus on differences in the `<li>` tags between the two lists. The `<li>` tag appearing in the `<ul>` list needs to be assigned a different color from the `<li>` tag appearing in the `<ol>` list. The need is to differentiate between the `<li>` tags appearing in the two lists. This differentiation is made with the following style sheet format.

```
<style type="text/css">
    ul li {color:red}
    ol li {color:blue}
</style>
```

Figure **7.1.25**: Using contextual selectors.

The *pairs* of selectors -- ul li and ol li -- describe the *combination* of tags that must occur for the associated style to be applied. These **contextual selectors** -- two or more selectors separated by blank spaces -- provide the tag relationships that must occur before the style is applied. The preceding code sets the color to red where an `<li>` tag follows a `<ul>` tag, and the color is set to blue when an `<li>` tag follows an `<ol>` tag. That is, the `<li>` tag takes on a style depending on its context, whether it is associated with a `<ul>` tag or with an `<ol>` tag.

Contextual selectors can also include ID selectors to further differentiate styles. The following code uses id values to distinguish two unordered lists for the purpose of applying different bullet styles. At the same time, the two sets of list items are assigned different colors depending on which unordered list they appear in.

```
<style type="text/css">
    ul#ListA    {list-style-type:disc}
    ul#ListB    {list-style-type:circle}
    ul#ListA li {color:red}
    ul#ListB li {color:blue}
</style>

<ul id="ListA">
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>

<ul id="ListB">
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>
```

Figure **7.1.26**: Using contextual selectors to differentiate unordered lists.

The two lists are differentiated in the style sheet by id values assigned to their `<ul>` tags. The first two style settings apply different bullet styles to the identified ul selectors. The last two style settings apply different colors to the li selector depending on *which* ul list it is contained in. The li color style depends on the ul ID context. Browser output of this code is shown in Figure **7.1.27**.

- Item 1
- Item 2
- Item 3

- Item 1
- Item 2
- Item 3

Figure **7.1.27**: Unorder lists displayed with different bullets and colors.

Use of contextual selectors is not limited to styling list structures. There will be numerous occasions throughout these tutorials where they are used to differentiate all manner of tag combinations.

Using Lists and CSS for Navigation Layout

Semantically speaking, since a navigation menu is a list of hyperlinks, it is much better to use unordered lists to configure a navigation menu. The following code block demonstrates how to use a list to create a navigation menu:

```
<div id="menu">

<ul>

<li><a href="home.htm">Home</a>/li>
<li><a href="about.htm">About Us</a></li>
<li><a href="products.htm">Products</a></l
<li><a href="contact.htm">Contact Us</a></li>

</ul>

</div>
```

- Home
- About Us
- Products
- Contact Us

Figure 7.1.28: **list-style-type** CSS property

The **list-style-type** CSS property can be set to **none** to prevent the browser from displaying the bullets. In addition, the CSS **display** property can be used to configure the list structure as an inline element. By default, a list structure is a block level element that displays each list item vertically down the page. Setting each list item to display inline will create a horizontal list of navigation items. The code to accomplish this is shown below:

```
.menu li {display: inline; list-style-type:none}
```

Home About Us Products Contact Us

Figure 7.1.29: Inlined Lists

Additional spacing between each menu item can be added using the **margin** property.

7.1: Styling Lists is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

7.2: Borders and Padding

Nearly all HTML elements can have borders, and can include padding (white space) around their contents. This is the case whether or not there are normally borders around the element or whether padding space is a normal characteristic of the tag.

Border Styles

Border styles include properties pertaining to the type of border, its width, and its color. The following table lists these properties.

Property	Value
border-style border-top-style border-right-style border-bottom-style border-left-style	dashed dotted double groove inset none outset ridge solid
border-width border-top-width border-right-width border-bottom-width border-left-width	thin medium thick *n*px
border-color border-top-color border-right-color border-bottom-color border-left-color	#000000 - #FFFFFF *color name* rgb(*r*,*g*,*b*)
border	border:*style size color*

Figure **7.2.1**: Border styles and properties.

Border style, width, and color properties can be applied to all four sides of an XHTML element or they can be selectively applied to individual sides. For example, the five types of border style properties are

Style	*Applies to*
border-style	*- applies to all four sides*
border-top-style	*- applies only to the top edge*
border-right-style	*- applies only to the right edge*
border-bottom-style	*- applies only to the bottom edge*
border-left-style	*- applies only to the left edge*

When a border specification applies to all four sides of an element, the shortcut border property combines and separates with spaces the following values within a single property declaration: style, width, and color, *in that order*. In other words, instead of coding the three separate specifications,

> *border-style:solid*
> *border-width:1px*
> *border-color:black*

these settings can be combined within a single border property:

> *border:solid 1 black*

All three of these values do not have to be given, but the remaining ones must be in the correct order: border:solid 1px (unspecified color).

There are eight border styles from which to choose. These styles are shown in Figure 6-16 with their widths set to 3 pixels. Smaller border widths do not display some of the styles.



Figure **7.2.2**: Border styles.

Borders are normally applied to tags such as `<div>`, `<p>`, and `<span>` tags, those that are containers for text. You can, however, experiment with other tags to see their border effects.

To illustrate styling variations, the following division is displayed with a border that has different styles on all four sides. The division encloses a paragraph that has its own border settings and includes a text string with its own border.

```
<style type="text/css">
div#A  {border-width:7px;
        border-color:red;
        border-top-style:solid;
        border-right-style:dashed
        border-bottom-style:ridge;
        border-left-style:double}

p#B    {border:dashed 3px blue}

span#C {border:solid 4px green}
</style>

<div id="A">
  <p id="B">This is a <span id="C">text string with borders</span>
  inside a paragraph with borders inside a division with four different
  borders.</p>
</div>
```

Figure **7.2.3**: Code for various border styles.

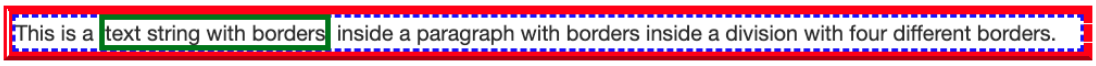


Figure **7.2.4**: Borders surrounding page elements.

Since the enclosing division displays different borders on each of its four sides, individual specifications are given for each side. Since identical borders appear around all sides of the paragraph and spanned text, the shorthand border property is declared for these containers. Although you probably will not go to this extreme in adding borders to page elements, this example shows the different border settings that can be made.

Border Radius

An alternative to regular borders would be the use of border-radius. This property adds a beveled appearance to the corners of the contents of a container.

Unlike regular borders, border-radius allows more customization. For instance, with this property, you can customize every corner at once, each one individually, or you can use a combination of beveled and regular corners to achieve a varied look. Additionally, containers styled using border-radius can still have a regular border around it. Figure 6-19 shows examples of these applications.

The style properties for adding a border-radius to a container is shown in Figure **7.2.5**.

Property	Value
border-radius border-top-left-radius border-top-right-radius border-bottom-left-radius border-bottom-right-radius	*n*px *n*em

Figure **7.2.5**: Borders surrounding page elements.

It is worth noting that while the border-radius property works well on container elements, such as div or span, it does not always display properly when applied to images. Therefore, to achieve an rounded corner appearance on an image, it is sometimes necessary to first wrap the image in a containing element, such as a div or span, and apply the border-radius property to that.

```
<style type="text/css">
div#A
        {
        width: 5em;
        height: 5em;
        background-color: red;
        border: solid 2px blue;
        border-radius: 1em;
        }

div#B
        {
        border-top-right-radius: 2em;
        border-bottom-left-radius: 2em;
        }

</style>

<div id="A">
</div>


<id="B" img src="DinosaursWithLasers.jpg">
```

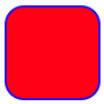


Figure **7.2.6**: Code for adding border radius to various objects.



Figure **7.2.7**: Border-radius application examples.

Padding Styles

In the above styling example, borders are collapsed around the text they enclose. In most cases, for visual attractiveness and readability, you will wish to leave space between the text and its border. This is accomplished by introducing **padding** inside the text container. Padding is the amount of space between the borders of a container and its enclosed content.

Padding is added to a container with the style properties shown in Figure **7.2.8**. The padding property introduces white space around all four sides of the container; padding-top, padding-right, padding-bottom, and padding-left selectively apply padding to each of the four sides.

Property	Value
padding padding-top padding-right padding-bottom padding-left	*n*px *n*em

Figure **7.2.6**: Padding style properties.

The following code is a repeat of the previous division with padding added to the `<div>`, `<p>`, and `<span>` tags to introduce additional space between the text and its enclosing borders. Different effects can be achieved by specifying different padding amounts on each of the four sides.

```
<style type="text/css">
div#A  {padding:7px;
        border-width:7px;
        border-color:red;
        border-top-style:solid;
        border-right-style:dashed;
        border-bottom-style:ridge;
        border-left-style:double}

p#B    {padding:7px; border:dashed 3px blue}

span#C {padding:2px; border:solid 4px green}
</div>

<div id="A">
  <p id="B">This is a <span id="C">text string with borders</span>
  inside a paragraph with borders inside a division with four different
  borders.</p>
</div>
```

Figure **7.2.9**: Code for adding border radius to various objects.

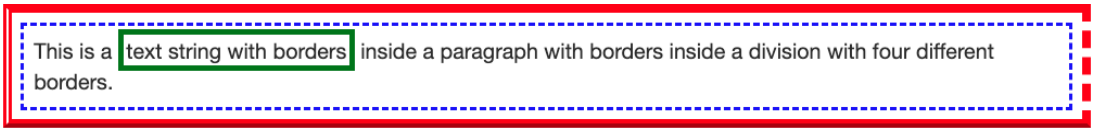


Figure **7.2.10**: Padding surrounding text elements.

Image Borders

Borders can be displayed around a picture by coding a border style. The following code produces a ridge border 7 pixels wide as shown on the left in Figure **7.2.11**.

```
<img src="Stonehenge.jpg" style="border:ridge 7px red">
```



Figure **7.2.11**: Pictures with Borders

You cannot separate the border from the picture by introducing padding around the picture itself. You can enclose an image inside another container, say a `<span>` tag, and add padding to this container. This technique is coded below and shown on the right in Figure 6-23.

```
span style="border:ridge 7px red; padding:10px; display:inline-block"><img src="Stonehenge.jpg"></spa
```

Figure **7.2.12**: Code for padding surrounding an image.

Margins, Borders, and Padding

The illustration in Figure 7.2.13 gives you a visual sense of the margin, border, and padding components of page elements. Each of these component parts can be sized the same around all four sides of a container, or individual sides can take on different measurements. In combination with container size and placement styles discussed, next you should be able to arrange and style containers for very precise placement and for enhanced readability of page content.

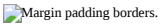


Figure 7.2.13: Margins, padding, and borders surrounding page elements.

7.2: Borders and Padding is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

7.3: Sizing Elements

Unless they are styled otherwise, text containers such as `<div>`, `<p>`, and `<span>` tags are sized to fit the contents they contain. It is often the case that you wish to have more control over the sizes of container tags, and you can with the style settings shown in Figure 6-24. You were introduced to the height and width properties in a previous tutorial. Here they are combined with the overflow property to control the size and visual appearance of text containers.

Property	Value
height	*n* px *n* % auto
width	*n* px *n* % auto
overflow	visible hidden scroll auto

Figure \(\PageIndex{1}\): Element size style properties.

Container Widths and Heights

In previous tutorials, height and width settings are applied to images and horizontal rules to give them specific sizes. The same can be done with virtually any XHTML tag, including text containers. For instance, the following paragraph is sized at 50% of the width of the browser window rather than expanding by default to the entire page width. A border is added to visualize the effect, and padding is used to keep the border from closing in around the text. This paragraph is displayed in the browser as shown in Figure **7.3.3**.

```
<style type="text/css">
  p#Sized {width:50%; border:solid 1px; padding:10px}
</style>

<p id="Sized">This is a paragraph with its width set to 50% of the width of the browser window.
Still, the text wraps within these boundaries. Since its width is set as a percentage, the paragraph
resizes to remain at 50% of the page width when resizing the browser window.</p>
```

Figure **7.3.2**: Code to size and pad a paragraph.

This is a paragraph with its width set to 50% of the width of the browser window. Still, the text wraps within these boundaries. Since its width is set as a percentage, the paragraph resizes to remain at 50% of the page width when resizing the, browser window.

Figure **7.3.3**: Paragraph with its width set to 50% of the page width.

The width of the paragraph always remains at 50%. Since no height styling is applied, the height of the paragraph expands to encompass the amount of text contained within it.

Container Overflows

By default, the height of a container always expands to display its contained text, irrespective of its specified height setting. In order to set the exact height of a container, you must also indicate how to handle text "overflow," where the container is not sized large enough to display all of its content. You must supply an overflow property to deal with potential "hidden" text beyond the boundaries of the container.

There are four possible values for the overflow property, the results of which are shown in the following examples of paragraph styling.

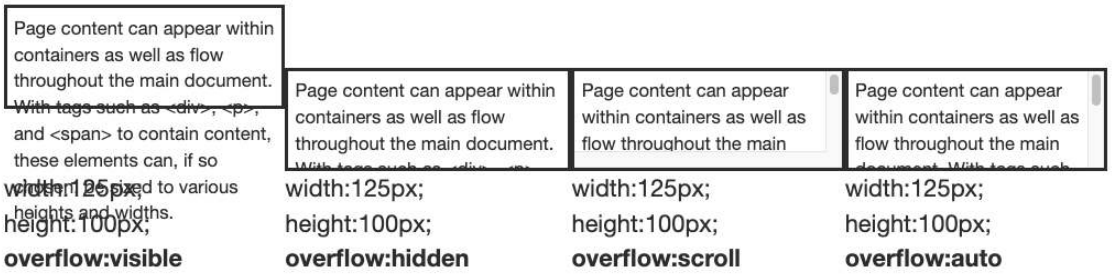


Figure **7.3.4**: Text container overflow settings.

A setting of overflow:visible displays all content irrespective of the specified height of the container (the default setting); overflow:hidden applies the specified height setting even if part of the text remains hidden; overflow:scroll displays horizontal and vertical scroll bars whether needed or not to view hidden text; overflow:auto displays a vertical scroll bar if needed to view hidden text.

The overflow:auto style is the most generally useful and visibly pleasing for displaying potential scrolling text within a container of a specified width and height. Coding for the above `<p>` tag with overflow:auto is shown below.

```
<style type="text/css">
  p#OFLOW {  overflow:auto;   width:125px; height:100px; padding:5px;
           border:solid 1px}
</style>

<p id="OFLOW">Page content can appear within elements as well as flow throughout the main document.
With tags such as &lt;div&gt;, &lt;p&gt;, and &lt;span&gt; to contain content, these elements can,
if so chosen, be sized to various heights and widths.</p>
```

Figure **7.3.5**: Code to handle overflow text within a paragraph.

This particular paragraph has id="OFLOW" to differentiate it from the other three paragraphs which have their own id values. The paragraph has width and height settings that do not permit full display of the entire paragraph. Therefore, it is also given an overflow:auto style so that a vertical scroll bar appears for accessing the hidden text.

Floating Containers

This is a division with its width set to 250 pixels and its height set to 200 pixels. It has a ridge border 5 pixels in width, padding of 10 pixels, and a background color of gray. The division floats to the right of the page. Its left margin is set at 20 pixels for separation from the text on the page.

Since all of the text cannot display within the specified height and width, the division is given a style setting of overflow:auto to create a vertical scroll bar.

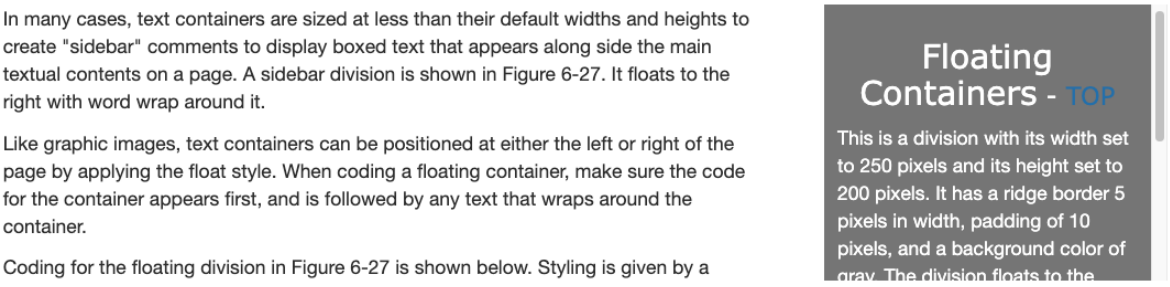


Figure **7.3.6**: A sidebar division.

In many cases, text containers are sized at less than their default widths and heights to create "sidebar" comments to display boxed text that appears along side the main textual contents on a page. A sidebar division is shown in Figure **7.3.6**. It floats to the right with word wrap around it.

Like graphic images, text containers can be positioned at either the left or right of the page by applying the float style. When coding a floating container, make sure the code for the container appears first, and is followed by any text that wraps around the container.

Coding for the floating division in Figure **7.3.6** is shown below. Styling is given by a class selector within an embedded style sheet. Division code appears immediately before the text that wraps around the container.

```
<style type="text/css">
  .FLOAT {float:right; width:250px; height:200px; overflow:auto; color:white;
          background-color:gray; margin-left:20px; padding:10px; border:
          ridge 5}
  h3     {text-align:center}
</style>

<div class="FLOAT">

<h3>Floating Containers</h3>

<p>This is a division with its width set to 250 pixels and its height set to
200 pixels. It has a ridge border 5 pixels in width, padding of 10 pixels, and
a background color of gray. The division floats to the right of the page. Its
left margin is set at 20 pixels for separation from the text on the page.</p>

<p>Since all of the text cannot display within the specified height and width,
the division is given a style setting of overflow:auto to create a vertical
scroll bar.</p>

</div>

... wrapped text ...
```

Figure **7.3.7**: Code for a floating division.

Notice that only about half of the content is visible within the division's width and height settings. Therefore, the overflow:auto property is added to the division's style sheet to display a vertical scroll bar for accessing all text. A border is added for visibility.

The above code is a good illustration of using the block-level `<div>` tag to enclose and style other text blocks as a single unit of page content. In this case, the `<div>` tag encapsulates a heading and two paragraphs. Its purpose is to identify this collection of content to receive styling in the form of a floated, bordered container inside of which the content can scroll.

7.3: Sizing Elements is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

7.4: Positioning Elements

Content on a Web page normally appears in the physical order in which it is coded in HTML. In addition, elements can be floated to the left or right of the page with word wrap around them. You may, however, wish to have additional control over placement of page elements. You can, indeed, have precise pixel control over element positioning with the style properties listed in Figure **7.4.1**.

Property	Value	Description
position	static relative absolute fixed	Places an element in a static (default), relative, absolute, or fixed position
left	*n* px *n* %	Sets how far the left edge of an element is to the right/left of the left edge of the parent element
right	*n* px *n* %	Sets how far the right edge of an element is to the left/right of the right edge of the parent element
top	*n* px *n* %	Sets how far the top edge of an element is above/below the top edge of the parent element
bottom	*n* px *n* %	Sets how far the bottom edge of an element is above/below the bottom edge of the parent element
z-index	*n*	Sets the stack order of an element
display	*none* *block* *inline*	Controls how and if the element will display
visibility	*visible* *hidden*	Controls whether an element will display and take up space on a Web page
float	*left* *right*	Sets the horizontal placement (left or right) of an element within its parent element
clear	*left* *right* *both*	Specifies the display of an element in relation to floating elements. Cancels the effects of a float.

Figure **7.4.1**: Positioning style properties.

In order to be positioned precisely on the page, an element must be assigned a `position` property. Thereafter, the element can be placed at a precise pixel location using the accompanying `left`, `right`, `top`, `bottom`, and `z-index` properties.

Static Positioning

By default, all elements are positioned statically position:static style. Static positioning means that all page elements appear in a normal flow: elements appear one after another from the top of the document and flow in sequence to the bottom of the document. Static elements have no special positioning applied and cannot be affected by any of the offset properties - left, right, top, bottom, z-index.

Relative Positioning

Elements on a Web page normally appear static at a location that is *relative* to surrounding elements on the page. That is, they are physically displayed in the order in which they are coded. This is the case with the following `<h1>` heading line. It appears next in order following the preceding paragraph because its coding appears next in order in the XHTML document.

```
<p>Preceding paragraph...</p>

<h1>Words in a Heading</h1>

<p>Following paragraph...</p>
```

Figure 7.4.2: Code to position a heading relative to surrounding content.

Preceding paragraph...

Words in a Heading

Following paragraph...

Figure 7.4.3:A heading positioned relative to surrounding content.

Normally, in-line text cannot be moved from its default position on the page. The built-in formatting given by the tag and its surrounding tags dictate where content is located. An `<h1>` tag always appears a double-space below a preceding paragraph and a double-space above a following paragraph. By assigning a position property to a tag, it can be repositioned with pixel precision up, down, or across the page.

In order to reposition the `<h1>` tag in the above example, it can be assigned a position:relative style. Then, by applying the left and/or top property, it can be moved by a certain number of pixels from its original location. The following code repositions this heading by styling the tag with positioning properties.

```
<style type="text/css">
  h1 {  position:relative; left:50px; top:-10px  }
</style>

<p>Preceding paragraph...</p>

<h1>Words in a Heading</h1>

<p>Following paragraph...</p>
```

Figure 7.4.4: A heading repositioned relative to its original location.

With a position of relative the tag can be repositioned *relative to its original location*. The left property gives the pixel distance by which the element is offset from its normal horizontal position; the top property gives the pixel distance by which the element is offset from its normal vertical position. In the above example, the heading is positioned +50 pixels from the left of its original location, and it is positioned -30 pixels from the top of its original location.

Notice that pixel directions can be positive or negative. While a positive value for the left property moves the element to the right, a negative value moves it to the left. A positive value for the top property moves the element down the page, and a negative value moves it up the page.

In the following example, each word in a sentence is packaged in a separate `<span>` container in order to style it separately. Each word then has its top position offset relative to its normal vertical position across the line. All words are contained by a `<div>` tag to apply font sizing to the group of words.

```
<div style="font-size:24pt">
  <span style="position:relative; top:-15px">Words</span>
  <span style="position: relative; top:+10px">in</span>
  <span style="position: relative; top:-5px">a</span>
  <span style="position:relative; top:+5px">sentence.</span>
</div>
```

Figure 7.4.5: Code to reposition words relative to their original locations.

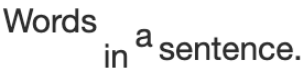


Figure 7.4.6: Words positioned relative to their normal vertical alignment.

Each `<span>` tag is positioned relative so that its top property can be applied. It is not necessary to position the words horizontally with a left property since `<span>` tags are, by default, positioned side by side across the line. Only the tops of the words are repositioned from their normal vertical alignment.

As you are repositioning page elements, you may need to use trial-and-error methods to get the exact positioning you want. There is no easy way to know at a glance exactly how many positive or negative pixel offsets are needed for horizontal and vertical placements.

Layering Elements

Page elements are layered on top of one another as they are added to a Web page. That is, each subsequently coded element is contained in a layer that sits on top of previous elements. Normally this layering is not evident and not important to know since page elements typically do not overlap. When elements are explicitly positioned on the page, they may overlap thereby making this layering obvious. It also may be necessary to change this default layering so that elements are in preferred overlapping order.

You can explicitly change default layering of page elements by coding their z-index style properties. The z-index value is a relative measurement. Elements with *larger* numeric values appear *on top of* elements with lower values. Thus, an element with z-index:2 appears on top of an element with z-index:1, and an element with z-index:20 appears on top of an element with z-index:10. The absolute value of z-index does not matter. All that matters are the *relative* z-index magnitudes assigned to a set of layered elements.

The colored squares shown in Figure 7.4.8 demonstrate various positions and layers. In this case, layering is given solely by the default order in which the squares are coded. Those coded last appear on top of those coded previously.

```
<style type="text/css">
  .RED    {display: inline-block;position:relative; width:100px; height:100px; left:0px; top:0px;
           background-color:red; border:solid 1px white; color:white;
           text-align:right}
  .GREEN  {display: inline-block;position:relative; width:100px; height:100px; left:-50px; top:25px;
           background-color:green; border:solid 1px white; color:white;
           text-align:right}
  .BLUE   {display: inline-block;position:relative; width:100px; height:100px; left:-100px; top:50px
           background-color:blue; border:solid 1px white; color:white;
           text-align:right}
</style>

<div>
<span class="RED">Red</span>
<span class="GREEN">Green</span>
<span class="BLUE">Blue</span>
</div>
```

Figure 7.4.7: Code to overlap page elements.

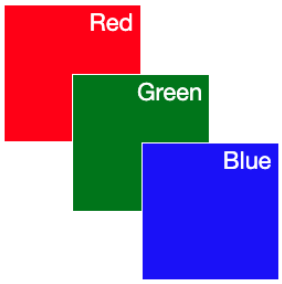


Figure 7.4.8: Normal layering of page elements.

The red square is coded first so it appears below the green square which is coded second, which appears below the blue square which is coded last. Notice that these squares are produced with `<span>` tags that are given widths, heights, colors, background colors, and borders. No z-index settings are necessary to create this layering; however, the squares are given left and top stylings to offset them horizontally and vertically from their natural side-by-side positions to overlap them and make their layering visually evident.

The above squares can have their layers reversed simply by assigning them z-index values. In the following code, the red square is assigned the largest value (bringing it to the top) and the blue square is assigned the smallest value (sending it to the bottom). The order of coding the `<span>` tags remains unchanged. Display of these squares is shown in Figure 6-33.

```
<style type="text/css">
  .RED    {display: inline-block;position:relative; width:100px; height:100px; left:0px; top:0px;
           background-color:red; border:solid 1px white; color:white;
           text-align:right;   z-index:3  }
  .GREEN  {display: inline-block;position:relative; width:100px; height:100px; left:-50px; top:25px;
           background-color:green; border:solid 1px white; color:white;
           text-align:right;   z-index:2  }
  .BLUE   {display: inline-block;position:relative; width:100px; height:100px; left:-100px; top:50px
           background-color:blue; border:solid 1px white; color:white;
           text-align:right;   z-index:1  }
</style>
```

Figure 7.4.9: Code to relayer page elements.

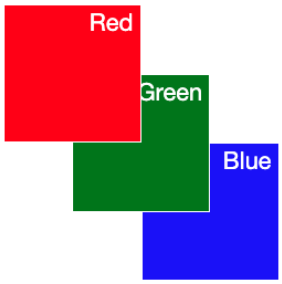


Figure 7.4.10: Reversed layering of page elements.

Recall that z-index values do not matter so long as the differences in magnitude are in proper relationship. The 3, 2, and 1 values used above could have been coded as 30, 20, and 10; or 300, 200, and 100; or 300, 20, and 1. The largest value is on top and the smallest value is on the bottom, irrespective of their absolute magnitudes.

Repositioning Blank Space

When using relative positioning there is not a lot of latitude in moving elements from their normal vertical positions on the page. The reason is that space is still reserved for the repositioned element at *its original location* on the page irrespective of the fact that it has been moved from that location. This can cause excessive blank space to appear on the page as shown by the following example of a repositioned paragraph.

```
<p>Preceding paragraph....</p>

<p style="position:relative; top:25px; font-size:24pt">
  Repositioned Paragraph.
</p>

<p>Following paragraph....</p>
```

Figure **7.4.11**: Code to reposition a paragraph.



Figure **7.4.12**: Repositioned paragraph leaving excessive white space at original position.

From its normal position the repositioned `<p>` tag is moved 25 pixels down the page. The original size and location of this tag is maintained in the flow of page elements, occupied now by blank space where the tag would normally appear. Not only that, any follow-on text maintains its position relative to the *original* location of the repositioned `<p>` tag, thereby being nearly overwritten by the moved text.

As long as page elements are repositioned vertically within a reasonable number of pixels from their original locations, the above spacing issues should not cause concern. As vertical distances increase, blank space appears in the flow of page elements representing abandoned space previously occupied by the repositioned element. You will probably need to try various positionings to get page elements to appear in satisfactory relationships to one another.

Absolute Positioning

Whereas position:relative positions a page element relative to surrounding elements, position:absolute positions an element *relative to its container element that is also positioned*. By default, the container element is the Web page itself, the `<body>` tag. Therefore, elements are absolutely positioned in relation to the top-left or bottom-right corner of the Web page and, importantly, are taken *out* of the normal flow of page elements.

Absolute positioning is shown in the example in Figure **7.4.14** where the word "DRAFT" is enclosed in a `<div>` tag that is positioned absolute. With this positioning the tag is located relative to the top-left corner of the page. It is offset 50px from the top and 280px from the left of the page, placing it beneath the accompanying paragraph.

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>Positioning</title>
</head>
<body>

<div style="  position:absolute; top:50px; left:280px; z-index:-1;
font-family:impact; font-size:68pt; color:#D6D6D6">DRAFT</div>

<p>Positioned beneath this paragraph is the word "DRAFT" defined by
the &lt;div&gt; tag shown above. This tag appears in the HTML code
immediately before this paragraph. It is positioned absolute; therefore,
it is taken out of the normal flow of page elements. With this word's
removal from the flow of page elements, this paragraph moves up to occupy
the abandoned page space, thereby overlaying the word. Thus, the word "DRAFT"
occupies an absolute position on the page unaffected by whatever else surrounds
it. It is also given a z-index value of -1 to layer it beneath the text layer
of the page so that it does not overlay the text.</p>

</body>
</html>
```

Figure **7.4.13**: Code to absolutely position content beneath the text layer of a page.



Figure **7.4.14**: Absolute positioning of content beneath the text layer of a page.

The text layer of a page always has a z-index value of 0 (zero). Therefore, the `<div>` tag is given a z-index value of -1 value in order to layer the word beneath the text layer so that it does not block out the text.

When a page element is positioned absolute and is given left and top property settings, it is taken out of the normal flow of page elements. In the above example, the `<div>` tag is removed from its normal physical location preceding the paragraph. The paragraph, then, moves up to occupy the abandoned position of the `<div>` tag. This means that it does not really matter where an element that is positioned absolute is coded on the page. Irrespective of where it is physically coded it is still positioned in relation to the top-left corner of the page. The only case where its coded position matters is when the element is positioned only by its left (but not its top) property. If it is not repositioned vertically, it stays in its coded position.

In some cases, the container for an absolute positioned element may be something other than the Web page itself or `<body>` tag. In the example below, a `<div>` serves as a container for a smaller red `<div>` that is positioned absolute.

```
<p>Paragraph above container...</p>

<div id="container" style="position:relative;height:200px;width:300px;border:solid 3px black;">

<div id="box" style="position:absolute;height:80px;background-color:red;width:80px">

</div>

</div>

<p>Paragraph below container...</p>
```

Figure 7.4.15: Code to surround container

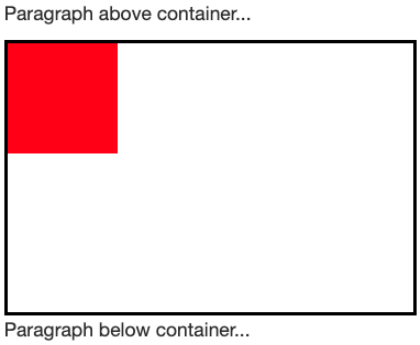


Figure 7.4.16: Container and surrounding paragraphs

The container `<div>` is positioned relative and therefore maintains the normal page flow between two

elements. Since the box `<div>` is coded inside of the container, it will be positioned absolute relative to the parent container `<div>` . As additional elements are added above the container, it will continue to move down the page and maintain normal page flow; however, the box will always remain positioned absolute with the container element.

Fixed Positioning

Fixed positioning is very similar to absolute positioning, with two major differences:

- Elements are always positioned absolutely relative to the browser window, no matter what type of positioning there parent element may have applied. In contrast, absolute positioned elements are positioned relative to their parent element.
- Elements that have a fixed positioning remain fixed in place as the document is scrolled by the user

Fixed positioning is useful when creating pages that include a consistent navigation system. The navigation menu can be styled to remain in place as the page scrolls, providing the user with easy to access navigational options. Fixed positioning provides a means by which web developers can use CSS to emulate a frames page.

Determining Positional Locations of Page Elements

Since it can be difficult to determine left and top page positions relative to a lengthy, scrolling Web page, elements that are positioned in relation to one another are often placed inside another container element. This container element is positioned relative to maintain its position within the flow of page elements, and the contained elements are positioned absolute inside the container. Thus, absolute position measurements are made relative to the *container* rather than to the page. The container becomes the easier-to-manage coordinate system within which contained elements are precisely positioned.

A good general solution to positioning page elements in relation to one another is

1. Define and size a tag to encompass the positioned elements. Position the division relative, thus locating it within the flow of page content. As content is added to or removed from the page, this division still maintains its relative position between other page elements. It moves up or down the page as page content changes.

2. Place elements to be positioned inside the division, and position them absolute<. Thus, the left and top distances for the positioned elements are always measured from the top-left corner of the division. This "local" coordinate system stays the same even if the division changes positions within the flow of page elements.

The division shown in Figure 6-36 is positioned relative to maintain its position among other XHTML elements on a page. It is displayed with a dotted border to make it visible. The enclosed squares are positioned absolute, with left and top positions measured from the top-left corner of this division.

```
<style>
  .DIV    { position:relative;   width:300px; height:160px; border:dotted 1}

  .RED    { position:absolute;   width:100px; height:100px; left:0px; top:0px;
            background-color:red; border:solid 1px white; color:white;
            z-index:1; text-align:right}
  .GREEN  { position:absolute;   width:100px; height:100px; left:50px; top:25px;
            background-color:green; border:solid 1px white; color:white;
            z-index:2; text-align:right}
  .BLUE   { position:absolute;   width:100px; height:100px; left:100px; top:50px;
            background-color:blue; border:solid 1px white; color:white;
            z-index:3; text-align:right}
</style>

<div class="DIV">

  <span class="RED">Red </span>
  <span class="GREEN">Green </span>
  <span class="BLUE">Blue </span>

</div>
```

Figure 7.4.17: Code to absolutely position elements inside a relatively positioned container.

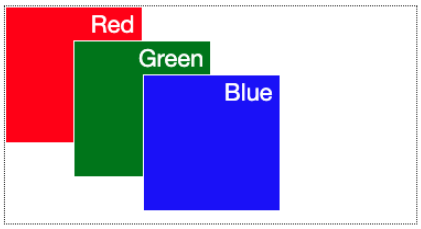


Figure 7.4.18: Absolute positioning inside a container positioned relative.

The advantage of using this strategy is that the container division can be moved to any location on the page and its contained elements still maintain their absolute positions within the division. It is not necessary to recalculate their positions since they are always relative to the top-left corner of the container division.

The following code uses this same strategy to produce a drop-shadow effect. Browser output is shown in Figure 7.4.20.

```
<style type="text/css">
  div#CONTAIN { position:relative;  height:45px; width:180px; border:dotted 1px}

  div#BLACK   { position:absolute;  left:0px; top:0px; z-index:2; font-family:
                impact; font-size:24pt; color:black}
  div#SILVER  { position:absolute;  left:+5px; top:+5px; z-index:1; font-family:
                impact; font-size:24pt; color:silver}
</style>

<div id="CONTAIN">
  <div id="SILVER">Drop Shadow</div>
  <div id="BLACK">Drop Shadow</div>
</div>
```

Figure **7.4.19**: Code to create a drop-shadow effect.



Figure **7.4.20**: Absolute positioning of elements inside a container positioned relative.

An encompassing division (positioned relative within the flow of page content) contains the two layered elements to be positioned. The container division is given a dashed border to show its size and position. The words and shadow positions can be easily determined by their distances from the top-left corner of this division. The words are positioned at the very top-left corner of the division (left:0px; top:0px); the shadow is offset 5 pixels from the words (left:+5px; top:+5px). The words are given a z-index value larger than the shadow to layer them on top.

When enclosing positioned content inside a division you normally need to set the width and height properties of the division. A positioned division has a default width that spans the width of its container -- the width of the Web page if its container element is the `<body>` tag. You may wish to set the division width only large enough to enclose its contained elements as is done for the above drop shadow effect. A positioned division also has a default height of 0 pixels irrespective of its content. Therefore, you will need to set its height large enough to reserve vertical page space for displaying its enclosed content.

When elements are positioned, they take a *shrink-to-fit* behavior. In other words, the `<div>` element only expand enough to accommodate the content contained within. In addition to the CSS height and width properties, the top, bottom, left, and right offset properties can affect dimensions of the element. If both top and bottom are specified on an absolutely positioned element, height is implied. Likewise, if both left and right offset properties are applied, width is implied.

CSS Page Layout

In addition to formatting text, page colors, and element positioning, CSS can also be used to configure the layout of a Web page. CSS layout is becoming more popular despite early problems with lack of browser support and continued use of traditional layout methods such as Frames and Tables. A CSS layout has many advantages over traditional page layout techniques. When CSS is used to configure the page layout in addition to formatting text and colors, the following advantages of using CSS are enhanced:

- Style is Separate from Structure
- Smaller HTML files
- Easier Site Maintenance
- Enhanced Accessibility
- Increased Page Layout Control
- Support of the Semantic Web

A crucial building block of CSS positioning is the CSS Box Model. The CSS Box Model describes how the browser measures properties such as width, padding, borders, and margins. Under the CSS box model, each page element is considered to a rectangular box. This box consists of a content area surrounded by padding, a border, and margins. The content area consists of a combination of text, graphics, or other Web page elements. The visible width of the element on the page is the total of the content width, the padding width, and the border width. The padding area is between the content and the border. The default padding is zero. The border area is between the padding and the margin. The default

border has a value of zero and does not display. The margin determines the empty space between the element and any adjacent elements. Figure 6-38 illustrates a typical CSS box model.

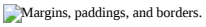


Figure **7.4.21**: Margins, padding, and borders surrounding page elements.

Creating a Simple Layout Using CSS

Figure 6-39 shows a simply page using CSS to control the page layout. Notice that the page consists of four containers or boxes: wrapper, heading, content, and footer. The wrapper or container is used to contain the page. All page elements are coded within the wrapper. The heading is used to code a heading for the page. Here a level 1 heading is used to format the name of the Web page. The heading container might also contain a company logo. The content box contains the main contents of the page. This can include any XHTML element (`<p>` , `<img/>` , `<div>` , `<h1>`). Finally, the footer is used to contain footer information. Here copyright information is included.



Figure **7.4.22**: Simple CSS page Layout.

The code for the simple layout is show below:

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>Positioning</title>
</head>
<body>

<div id="wrapper" style="border:solid 2px black;width:auto">


<div id="heading" style="border:solid 3px black;
width:800px;padding:10px;margin:10px">

<h1> World Wide Web Products</h1>

</div>

<div id="content" style="border:solid 3px black;
width:800px;padding:10px;margin:10px">

<p>"Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa
quae ab illo inventore veritatis et quasi architecto beatae vitae
dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit
aspernatur aut odit aut fugit, sed quia consequuntur magni dolores
eos qui ratione voluptatem sequi nesciunt.Neque porro quisquam est,
qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit,
sed quia non numquam eius modi tempora incidunt ut labore et dolore
magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum
exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea
commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea
voluptate velit esse quam nihil molestiae consequatur, vel illum qui
dolorem eum fugiat quo voluptas nulla pariatur?"</p>


<p>"Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae
ab illo inventore veritatis et quasi architecto beatae vitae dicta
sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur
aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione
voluptatem sequi nesciunt.Neque porro quisquam est, qui dolorem ipsum
quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam
eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat
voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam
corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur?
Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam
nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas
nulla pariatur?"</p>

</div>
```

```
<div id="footer" style="border:solid 3px black;
width:800px; padding:10px;margin:10px">
<p>© 2010 WWW Products</p>
</div>

</div>

</body>
</html>
```

Figure 7.4.23: Code for simple layout

The CSS for the simple layout is described below:

Wrapper Container - #wrapper {border:solid 2px black;width:auto}

The wrapper or container division is configured with a solid, 2px, and black border. It can be helpful to initially display borders around the divisions to help with layout. The border can be removed on the final display. The width of the wrapper is set to auto. By default, width and height properties have an auto value. The meaning of the auto keyword changes depending on the type of element that it is applied to. When used with the element, the element spans all the horizontal space available to it and expands vertically to accomodate any content inside of it, including text, images, and other divisions. In this example, the width of the wrapper horizontally spans the width of the page.

Heading Container - #heading {border:solid 3px black; width:800px;padding:10px;margin:10px}

The heading is the first container coded in the wrapper. It is given a 3px, solid, and black border. The heading is assigned a width of 800px. 10 pixels of padding is applied to the container and it is assigned a 10px margin.

Content Container - #content {border:solid 3px black; width:800px;padding:10px;margin:10px}

The content container appears below the heading container. Like the heading container, it is given a width of 800px, padding of 10px, and a margin of 10px.

Footer Container - #footer {border:solid 3px black; width:800px; padding:10px;margin:10px}

The footer container appears below the content container. Like the heading and content containers, it is given a width of 800px, padding of 10px, and a margin of 10px.

Since all containers are positioned static, they follow the normal page flow. In other words, they appear in the order in which they are coded - heading, content, and footer.

Creating a Two-Column Layout Using CSS

Figure 6-40 shows a two-column layout using CSS to control the page layout. Notice that the page consists of three containers or boxes: wrapper, menu, and content. The wrapper or container is used to contain the page. All page elements are coded within the wrapper. The menu is used to code a menu for the page. Here, a level 2 heading is used to format the menu heading. A list structure is used to create the three menu items. The menu container is set to float left. The content box contains the main contents of the page. This can include any XHTML element (`<p>` , `<img/>` , `<div>` , `<h1>`). The content container wraps around the right side of the floating menu container to create a two-column effect.

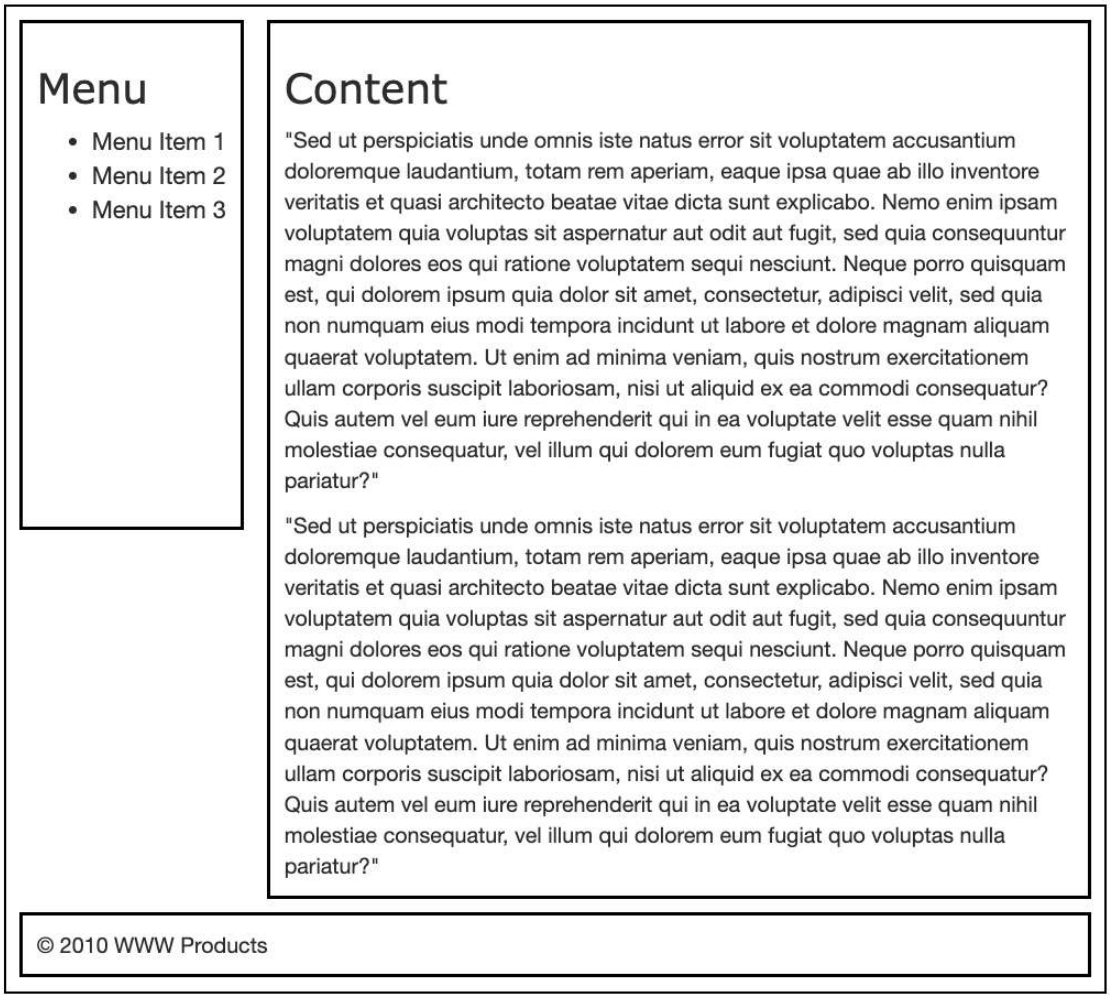


Figure **7.4.24**: Two-Column CSS page Layout.

The code for the two-column layout is show below:

```
<div id="wrapper" style="border:solid 2px black;width:auto">

<div id="menu" style="border:solid 3px black;
  width:150px;height:370px;padding:10px;margin:10px;float:left">

<h2>Menu</h2>

<ul>

<li>Menu Item 1</li>
<li>Menu Item 2</li>
<li>Menu Item 3</li>

</ul>

</div>

<div id="content" style="border:solid 3px black;
  padding:10px;margin:10px;width:650px;margin-left:190px">

<h2>Content</h2>

<p>"Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa
quae ab illo inventore veritatis et quasi architecto beatae vitae
dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit
aspernatur aut odit aut fugit, sed quia consequuntur magni dolores
eos qui ratione voluptatem sequi nesciunt.Neque porro quisquam est,
qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit,
sed quia non numquam eius modi tempora incidunt ut labore et dolore
magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum
exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea
commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea
voluptate velit esse quam nihil molestiae consequatur, vel illum qui
dolorem eum fugiat quo voluptas nulla pariatur?"</p>


<p>"Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae
ab illo inventore veritatis et quasi architecto beatae vitae dicta
sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur
aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione
voluptatem sequi nesciunt.Neque porro quisquam est, qui dolorem ipsum
quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam
eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat
voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam
corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur?
Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam
nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas
nulla pariatur?"</p>
</div>
```

```
<div id="footer" style="border:solid 3px black;padding:10px;margin:10px;width:auto;clear:left">

<p>© 2010 WWW Products</p>

</div>


</div>
```

Figure **7.4.1**: Copy and Paste Caption here. (Copyright; author via source)

The CSS for the two-column layout is described below:

Wrapper Container - #wrapper {border:solid 2px black;width:auto}

The wrapper or container division is configured with a solid, 2px, and black border. It can be helpful to initially display borders around the divisions to help with layout. The border can be removed on the final display. The width of the wrapper is set to auto. By default, width and height properties have an auto value. The meaning of the auto keyword changes depending on the type of element that it is applied to. When used with the element, the element spans all the horizontal space available to it and expands vertically to accommodate any content inside of it, including text, images, and other divisions. In this example, the width of the wrapper horizontally spans the width of the page.

Menu Container - #menu {border:solid 3px black;width:150px;height:370px;padding:10px;margin:10px;float:left}

Like the wrapper container, the menu is initially given a border set to solid, 3px, and black in color. This border can also be removed on the final display. The key to the two-column layout design is the float property. The menu container is set to float to the left. The width of the container is set to a value of 150 pixels. It is given 10 pixels of padding (white space between the menu border and any content coded within the container), and a 10 pixel margin surrounding the menu container. The height of the menu is set to 370 pixels so that it is the same height as the content container.

Content Container - #content {border:solid 3px black;padding:10px;margin:10px;width:650px;margin-left:190px}

The content container wraps around the right side of the floating menu. It has a solid, 3px, and black border. The width of the content is set to 650px. Since the menu container is 150 px wide with 10px (left and right margins) and 10px pixels of padding (left and right padding), the content container is given a left margin of 190px (150px + 20px + 20px). This margin value should be greater than or equal to the width (including any margin or padding values) to provide the look of a two-column layout.

Footer Container - #footer {border:solid 3px black;padding:10px;margin:10px;width:auto;clear:left}

The footer container appears below the menu and content containers. It has a solid, 3px, and black border. The width of the footer is set to auto so that it spans the width of the wrapper. The clear property is also applied to cancel the effects of the floating menu container. The clear property is set to left so that the footer aligns with the left margin and does not attempt to wrap around the menu. When the float property is applied to an element, all subsequent elements will attempt to wrap around the floating element. The clear property cancels out this effect.

CSS Layout Design Issues

Using CSS for page layout can be challenging. It requires lots of practice and patience. One of the major problems with CSS layout is that even modern browsers implement CSS in different ways. A CSS design may look perfect in Firefox, but the display could be drastically different in Internet Explorer. Testing CSS layout using multiple browsers is crucial. Expect that pages will display slightly different in various browsers. Your goal should be to design a page that looks best on the most commonly used browsers (currently Internet Explorer and Firefox) and displays acceptably well on other less common browsers.

It is also helpful to be able to debug CSS documents. Some common debugging techniques include:

- Manually check your CSS for Syntax Errors
- Use the CSS validation Tool at http://jigsaw.w3.org/css-validator/ to check CSS syntax
- Configure temporary background colors and borders for page elements. This will help you easily identify page elements and locate possible problems associated with them
- Use CSS comments to block sections of CSS until the coding problem is located

7.4: Positioning Elements is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

8: Linking Pages

8.1: Text Links
8.2: Graphic Links
8.3: Image Maps
8.4: Redirecting Pages
8.5: Traversing Folders

8: Linking Pages is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

8.1: Text Links

The feature that best characterizes the World Wide Web is its ability to link directly from one page to any other page anywhere on the Web. Normally, this hyperlinking is triggered by a mouse click on a letter, word, phrase, or graphic image on the linking page, and the linked page is retrieved and loaded immediately into the browser. Web links can be made to local pages on the same Web server as the linking page or to a page at any other site on the Web. This is a very powerful yet easy to use feature that allows you to navigate to pages located around the world with the mere click of the mouse.

The <a> Tag

The most common type of link is a clickable word or phrase that transfers directly to the target page. A text link is created by surrounding the text string with an <a> anchor tag that specifies the location of the page to which to link. The basic format of the tag is shown in Figure 7-1.

```
< a href= "url">linking text> </ a >
```

Listing 7-1. *General format for <a> tag.*

By default, the linking text is underlined and displayed in blue as a visual clue that the text string is a clickable link. The location of the linked page is given by the **href** (hyperlink reference) attribute.

You can link to your own page or to a page at a remote Web site. If the local linked page resides in the same directory as the linking page, then only the page name is needed as the URL. If the linked page is on the same Web server as the linking page but in a different directory, then the directory path to that target page is used as the URL (a relative link). If the linked page is at a different Web site, the link must include the protocol and domain reference "http*://domain name*" (an absolute link). You can link to the site name to retrieve the default home page or, if it is known, to a particular page at that site.

```
<a href="xhtml07-01.htm"> Reload This Page </a>
<a href="http://www.weather.com"> The Weather Channel
```

Reload This Page

The Weather Channel

Listing 7-2. *Linking to local and remote Web pages.*

Recall that an <a> tag is an in-line tag that must be enclosed inside a block-level tag such as a <p> or tag to meet HTML 5 standards.

accesskey Attribute

The accesskey attribute is used with the <a> tag to identify a keyboard key that can be used in combination with other keys to activate the control. The key combination depends on the browser. In Firefox, the combination is (ALT + SHIFT + accesskey) in Windows or (CTRL + accesskey) on Mac. In Internet Explorer, Chrome, and Safari the combination is (ATL + accesskey). An accesskey value "w" is set for the following link:

The Weather Channel

```
<a href="http://www.weather.com"  accesskey="w" > Th
```

Listing 7-3. *Opening a web page with an access key combination.*

title Attribute

The title attribute is used with the `<a>` tag to add descriptive text to links, especially if the link text itself does not clearly describe the link's destination. The title text appears when the user moves the mouse over the link. A title is set for the following link:

The Weather Channel

```
<a href="http://www.weather.com"  title="The Weather
```

Listing 7-4. *Example of a link with a title attribute.*

Targeting a New Browser Window

Unless instructed otherwise, the linked page opens in the same browser window that displays the linking page. The original page is replaced by the linked page. Especially when linking to remote sites on the Web, it is often convenient to open that page in a different browser window. When visitors leave your site to browse remote sites, they may not be able to back-click their way to your original page. By opening remote sites in a new window, your visitors do not lose contact with your pages. Your site is always available in the original window. In HTML, you can specify how a linked page is to be opened by coding a **target**="*_window*" attribute in the `<a>` tag.

The Weather Channel

```
<a href="http://www.weather.com"  target="_blank" >
```

Listing 7-5. *Opening a page in a new browser window.*

The above code includes a **target** set to the value **_blank** which opens the given URL in a new or blank browser window.

There are other **target** values, but they apply to using links with inline frames or iframes. These will be discussed later in the inline frames section of the tutorial.

Link Styles

Text links are displayed in three different colors to identify three conditions of a link. An **unvisited** link is displayed in blue, a **visited** link is displayed in purple, and an **active** link (the mouse button is clicked while pointing at the link text) is displayed in red. Also, text links are underlined.

You can override these default colors and underlines as well as include other visual indicators of link status with the style sheet selectors for the `<a>` tag shown in Figure 7-1.

Property:	Value
a:link a:visited a:hover a:active	Any text properties.

Figure 7-1. Link selectors and stylings.

The **a:link** selector identifies an unvisited link, the **a:hover** selector identifies a link with the mouse positioned over it, the **a:active** selector identifies a link being clicked, and the a:visited selector identifies a link that has been visited. Any combination of text style properties and values can be applied to these linking states. The following style sheet illustrates possible settings for the four link states.

```
<style type="text/css">
 a:link    {color:blue; text-decoration:none; font-si
 a:visited {color:gray; text-decoration:none; font-si
 a:hover   {color:green; font-weight:bold; text-decor
            font-size:1.2em}
 a:active  {color:red; text-decoration:underline; fon
</style>
<p>
    <a href="SomePage.htm"> Text Link </a>
</p>
```

Listing 7-6. Setting styles for text links.

Heading

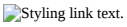


Figure 7-2. Styling link text to respond to user actions.

A normal unvisited link is colored blue but does not display the underline. When the mouse cursor is moved on top of the link it changes to green, is underlined, and is displayed in 1.2em size. When the link is clicked its color changes to red. A visited link is displayed at 1em gray with no underline.

Fragment Identifiers

Links are normally made between different Web documents so that visitors can navigate between pages. Links can, however, be made to different locations in the same document by coding a hyperlink to a **fragment identifier** which is an HTML element such as a `<div>` that is assigned an **id**.

This idea is illustrated in Figure 7-3 where text links at the top of the page transfer to content sections lower on the page. In addition, links at the end of each content section transfer back to the top of the page.

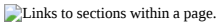


Figure 7-3. On-page links.

In order to create on-page links you need to code the `<a>` tags shown in Figure 7-4.

```
 <   a href=  "#name"> link text </a>
...
 <   div id=  "name">" fragment identifier or target te
```

***Figure 7-4.** General formats for `<a>` tags to create on-page links.*

The location *to which* a link is made is enclosed in an `< div id ="name">` tag that supplies a text value to identify the on-page link destination. Typically, the location of a link leads to a side heading on the page, but any other fragment identifier can be used. The text *from which* the link is made is enclosed in an `< a href ="#name">` tag in which this destination name is coded, preceded by "#".

The following code shows three links to three different locations on a page. The target locations have the fragment identifiers ITEM1, ITEM2, and ITEM3. When a link is clicked at the top of the page, the browser scrolls to one of these identified destination links.

```
<div> <a href="#ITEM1"> Go to Item 1 </a> </div>
<div> <a href="#ITEM2"> Go to Item 2 </a> </div>
<div> <a href="#ITEM3"> Go to Item 3 </a> </div>



<div id="ITEM1"> <b> Here is Item 1 </b> </div>
...
<div id="ITEM2"> <b> Here is Item 2 </b> </div>
...
<div id="ITEM3"> <b> Here is Item 3 </b> </div>
...
```

Listing 7-7. *Code to create on-page links.*

When linking from the top of a Web page to locations farther down the same page, it is a good idea to provide return links to the top of the page. In this example, each linked section (ITEM1, ITEM2, and ITEM3) is followed by a link back to the Top of the page. Each of these on-page links is coded as follows.

```
<div><a href="#"> Top </a></div>
```

Listing 7-8. *Code to link to top of page.*

Note that there is *no* on-page destination name given in the link URL: href="#". When no named location is given for an on-page link, the browser returns to the top of the page by default.

Fortunately, you can supply a named text value as the return destination for linking back to the top of the page or to any other location on the page. For instance, you could add a heading at the top of the page as the return link target.

```
<h1> <div id="TOP"> Top of Page </div> </h1>
.
.
.
<div> <a href="#TOP"> Top </a> </div>
```

Listing 7-9. *Code identifying a named target at top of page.*

Now when a Top link is clicked, the page is scrolled to the named location of the heading "Top of Page."

It is possible to link from one page to a specified location on a different page by combining an external link with an on-page link. Assume, for instance, that you need to link to a section of NextPage.htm that is identified with the tag

`<div id="SECTION3">` . Use the format for an on-page link -- `<a href="#name">` -- and simply prefix the target #name with the name of the page.

```
<div>  <a href="NextPage.htm#SECTION3"> Go to Section
```

Listing 7-10. *Code to link to a named target on a different page.*

This link transfers to a page named NextPage.htm and scrolls to the tag identified with `<div id="SECTION3">` . This type of linking can normally be done only with pages you create. It is unlikely that you would know the named target sections of remote pages even if they are available for linking.

Links to Other Documents Types

You can create links to documents other than Web pages. You can use the `<a>` tag to link to text documents, word processing documents, spreadsheets, graphic files, and other types of documents. The manner in which the browser treats the linked document depends on the document's file extension and how the browser is set up to handle non-HTML extensions.

Text Files

Standard text files with the .txt extension are opened in the browser just like an XHTML document. The file is displayed in the original font face and style used to create the document. Furthermore, the browser maintains original line breaks and spaces coded in the document. The link below retrieves a text document located in the same directory as the Web page containing the link and opens it in the browser as shown in Figure 7-5.

```
<p><a href="  TextDocument.txt  "> Display Text Docume
```

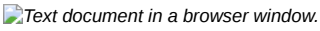


Figure 7-5. *Text document opened in browser window.*

In this example, the text document is opened inside the browser window, and replaces the linking page. However, when a document with a .txt file extension is opened in a separate window, it is opened by the program associated with that file type. In the case of a .txt file, the default text editor program, normally Notepad, is used to open the document, which then appears inside the external Notepad window. If the user has chosen a different editor program as the system default, then the document is opened inside that program.

Word Processing Documents

Word processing documents can be displayed in the browser if compatible software is available on the the user's computer. When linking to a Microsoft Word document -- and Microsoft Word (or the special Word reader software) is installed on the user's computer system -- then the Word document opens inside the browser window. The following link opens a Word document by replacing the linking page as shown in Figure 7-6.

```
<p><a href= " WordDocument.doc " > Display Word Docume
```

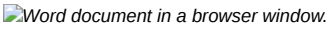


Figure 7-6. *Word document opened in browser window.*

Notice that the Microsoft Word program opens inside the browser window and that a subset of Word menus is added to the browser's Menu Bar. Users can perform basic editing of the Word document and can save it to their local computers. Of course, they can only edit and re-save the copy of the document displayed in the browser window. No alterations may be made to the original document. The browser's Back button is used to return to the linking page. If the linked document is opened in a separate browser window it also includes the subset of Word editing menus.

Spreadsheet Documents

As with word processing documents, spreadsheets are displayed in the browser if compatible software is available on the user's computer. When linking to a Microsoft Excel document -- and Microsoft Excel (or the special Excel reader software) is installed on the user's system -- then the spreadsheet opens inside the browser window. The following link opens an Excel spreadsheet in the same browser window as the linking page as shown in Figure 7-7.

```
<p><a href= " ExcelDocument.xls " > Display Excel Docu
```

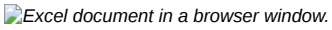


Figure 7-7. *Excel document opened in browser window.*

A subset of Excel menus is added to the browser's Menu Bar for minor editing and saving of the document to the local computer. The Back button is used to return to the linking page. A similar display opens when the link is made to a separate browser window.

Presentation Documents

Slide presentations are displayed in the browser if the same software is available on the user's computer. When linking to a Microsoft PowerPoint document -- and PowerPoint (or the special PowerPoint reader software) is installed on the user's system -- the presentation opens inside the browser window. The browser's Back button is used to return to the linking page. The following link opens a PowerPoint document for display in the browser window as shown in Figure 7-11.

```
<strong>"<p><a href= " PowerPointDocument.ppt " > Disp
```

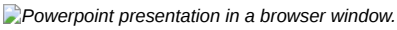


Figure 7-8. *PowerPoint document opened in browser window.*

Graphic Files

GIF, PNG, and JPEG images can be displayed in the browser through direct links to these graphic files. In other words, it is not necessary to embed a graphic on a Web page in order to display it. Simply by linking to files with .gif, .png, and .jpg file extensions the images are opened directly inside the browser.

Since the images are not formatted on a Web page, they are positioned in the top-left corner of the browser window at their original sizes. The following link opens a .gif image file as shown in Figure 7-9. When the image opens in the same browser window, the Back button is clicked to return to the linking page.

```
<p><a href= " GraphicImage.gif " > Display GIF Image
```

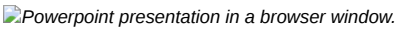


Figure 7-9. *GIF image opened in browser window.*

Email Links

There is a variation of the href attribute that permits you to set up an email link. A click on the link opens the visitor's default email program for correspondence with the address provided in the link. The general format for this link is shown in Figure 7-10.

```
<p><  <a href="mailto:" :email@address" > link text </
```

Figure 7-10. *General format for mailto: link.*

The **href="mailto:"** attribute is followed by an email address which should be the address at which you wish to receive correspondence from visitors. For example, the following link opens the visitor's email program (the one that is set as their browser's default email program), and inserts the specified email address into the address line.

```
<p><a href="  mailto: anybody@mail.mga.edu" > Contact
```

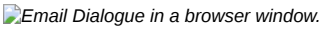


Figure 7-10. *Opened email program from href="mailto:" link.*

Download Attribute

When linking to documents, you can also specify whether the target file will be downloaded by using the download attribute. This is a new HTML 5 attribute and currently has only limited browser support. The value of the attribute specifies the name that will be assigned to the file once it is downloaded. If a value is omitted, the original file name will be used.

```
<p><a href= " GraphicImage.gif "  download="flower"> D
```

Download GIF Image

In this case, when the hyperlink is clicked, the file will also be downloaded with the name "Flower.gif"

Using CSS and Pseudo-Classes to Build Navigation Buttons

The navigation buttons below are coded using a combination of CSS, pseudo-classes, and the anchor tag. These buttons can be used instead of graphics buttons. They require no special graphics application to develop and save on bandwidth used by graphics files. Notice how the appearance changes when the mouse hovers over the menu buttons.

Home About Us Contact Us

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

      #buttons {margin-top:25px}

      #buttons a {border:inset 2px #CCCCCC; padding:
      background-color:#000066; font: bold arial; te

      #buttons a:link {color:#FFFFFF}
      #buttons a:visited {color:#CCCCCC}
      #buttons a:hover {color:#66CCFF; border: outse
      #buttons a:active {color:#DDA0DD; border:outse
   /style

<div id="buttons">
<a  href="#"> Home </a>
<a  href="#"> About Us </a>
<a  href="#"> Contact Us </a>
```

```
</div>
```

Deprecated <body> Attributes

Link colors for a page can be changed by coding attributes in the <body> tag. The three attributes are as follows:

link=*"color"* - an unvisited link vlink=*"color"* - a visited link alink=*"color"* - an active link

Use color names or hexadecimal values to set the color of an unvisited link, a visited link, and an active link, respectively.

```
<body link="green" vlink="gray" alink="purple">
```

These attributes are deprecated under XHTML and HTML 5, and preference should be given to using comparable link selectors and style sheet properties.

Deprecated Named Anchors

Prior to XHTML 1.1, named anchors, using the <a name=""> , were used to create on-page links. While they continue to be support by most modern browsers, the use of named anchors is no longer supported under XHTML 1.1 or HTML 5. Named anchors used the now deprecated **name** attribute to identify or name the fragment. The code below shows the deprecated format for creating on-page links:

```
<a name="top"> Top of Page </a>


<a href="#top" >Top </a>
```

It is sometimes necessary to link to a named fragment on another Web page. To accomplish this, the "#" followed by the fragment identifier id is placed after the URL in the <a> tag. The following code shows how to link from a Web page named HomePage.htm to link to a fragment identifier in another Web page named AboutUs.htm:

```
<-- HomePage.htm -->

<a href="AboutUs.htm#Section1>" Link to fragment iden


<-- AboutUs.htm -->

<div name="Section1"> Fragment identifier "Section1"
```

8.1: Text Links is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

8.2: Graphic Links

Links can be triggered by graphic images by enclosing an `<img>` tag inside an `<a>` tag. The general format for assigning links to graphic images is shown below.

```
< a href= "url"><  img  src  ="url" alt="text"></  a  >
```

Listing 7-11. *General format for graphic link.*

Images are particularly effective for links since, with informative graphics, you can tell at a glance the site destinations. The links coded in Listing 7-12 and shown in Figure 7-11 are made from images taken from their associated sites.

```
<style>
 img  {border:0; vertical-align:middle}
</style>

<p>
    <a href="http://www.ebay.com"><img src="ebay.jpg"
    <a href="http://www.ebay.com"> Ebay </a>
</p>

<p>
    <a href="http://www.nascar.com"><img src="nascar.
    <a href="http://www.nascar.com"> NASCAR </a>
</p>

<p>
    <a href="http://www.weather.com"><img src="weathe
    <a href="http://www.weather.com"> The Weather Cha
</p>

<p>
    <a href="http://www.microsoft.com"><img src="micr
    <a href="http://www.microsoft.com"> Microsoft </a
</p>
```

Listing 7-12. *Code to create graphic and text links.*

eBay Link*Ebay*

NASCAR Link*NASCAR*

Weather Link*The Weather Channel*



Microsoft

Figure 7-11. *Graphic and text links.*

By default, the link graphic is surrounded by a border to indicate that it is a link, and it serves the same purpose as the underline style of text links. In this example, the border is suppressed by including a style sheet to set image borders to 0px in width.

When using graphic images as links, it is also a good idea to provide text links as well. This is done in the current example by including separate text links which are aligned at the middle of their associated images through the style sheet vertical-align property.

Recall that a remote Web site should be opened in a separate browser window so that the user remains in contact with your site. The above links are modified in the following code to open them in a new browser window using the target attribute described previously.

```
<style>
 img  {border:0; vertical-align:middle}
</style>

<p>
    <a href="http://www.ebay.com" target="_blank"><im
    <a href="http://www.ebay.com" target="_blank"> Eb
</p>

<p>
    <a href="http://www.nascar.com" target="_blank"><
    <a href="http://www.nascar.com" target="_blank">
</p>

<p>
    <a href="http://www.weather.com" target="_blank">
    <a href="http://www.weather.com" target="_blank">
</p>

<p>
    <a href="http://www.microsoft.com" target="_blank
    <a href="http://www.microsoft.com" target="_blank
</p>
```

Listing 7-13. *Code to open graphic and text links in a separate browser window.*

8.2: Graphic Links is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

8.3: Image Maps

An **image map** is a graphic image with clickable areas that link individually to different pages. Often, images maps are used to display a large image on the opening page of a site with portions of the image representing links to the separate site areas to which the visitor can navigate. The clickable areas of the graphic can take the form of circles, rectangles, and/or multi-sided polygons.

The image map and accompanying text links shown in Figure 7-12 are on-page links to headings appearing farther down the page. This image map uses graphics of the basic shapes that can be defined for clickable areas. These shapes, however, can be superimposed on any type of graphic, including photographs and complex line drawings.



Figure 7-12. Image map and accompanying text links.

Determining Mapped Areas - TOP

To create an image map you start with the picture on which you wish to map clickable areas. Then, using a graphic program to display the image, you determine the shapes and sizes of the different areas to which you wish to assign links. Usually, graphic programs display the horizontal and vertical coordinates of the mouse pointer as you move it around the image. By viewing this display, you can determine the exact pixel coordinates that define the shapes of those clickable areas. Once this information is determined, you can code the HTML to convert the picture to an image map.

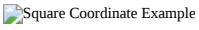
Figure 7-13 shows the picture from which the above image map is made. The image is displayed in the Windows Paint program which can track the movement of the mouse around the image.



Figure 7-13. Determining horizontal and vertical coordinates on graphic image.

Notice in the status bar of the window that the horizontal and vertical coordinates of the "pencil" pointer are displayed. The pointer is at one of the corners of the polygon shape and the tracking position shows 175,200. Im other words, the pointer is 175 pixels from the left edge of the image (horizontal position) and 200 pixels from the top edge of the image (vertical position). Coordinates are always measured from the top-left corner of the full image.

You need different positioning information depending on the shape of the mapped area.



For a **rectangle** you need to know the horizontal and vertical coordinates of the top-left and bottom-right corners. These two h,v coordinates can be determined by placing the pencil pointer at each of these corners and reading the coordinates from the status bar. In the above graphic, these two pixel coordinates are top-left = 8,13 and bottom-right = 130,123.



For a **circle**, you need to know the coordinates of the center point and the pixel width of the radius. The center coordinates of the above circle are determined by positioning the pencil pointer at the (approximate) center and reading the status bar = 228,123. The radius is the distance from the center to the edge of the circle. You can determine the radius by reading the coordinates of the right edge of the circle (at the same vertical position as its center point) and finding the difference between the horizontal center and horizontal edge: radius = (290 - 228) = 62.



For a **polygon** you need to know the coordinates of each of its corner points tracing either clockwise or counter-clockwise around its boundaries, starting at any corner point on the polygon. Tracing clockwise from the top of the above polygon the coordinates are 100,144; 175,200; 155,255; 50,250; and 45,172.

With these shape coordinates in hand, you are ready to code the image map and to assign URL links to the different shapes.

Coding an Image Map - TOP

The image to become an image map is placed on the page using an `<img>` tag. Along with its standard attributes this tag contains a **usemap**="*mapname*" attribute that points to a like-named `<map>` tag giving descriptions of the areas of the image that are clickable for linking. The general formats for the `<img>` and `<map>` tags are shown in Listing 7-14.

```
<img src="url" alt="text" usemap="#mapname">

<map name="mapname"  id="mapname">
  <area
    shape="rect|circle|poly"
    coords="coordinates"
    href="url"
    alt="text"
  >
  ...
</map>
```

Listing 7-14. *Associating a displayed image with an image map.*

In the `<img>` tag a *#mapname* is assigned in order to associate the image with a `<map>` tag identified by *mapname*. Inside the `<map>` tag are `<area>` tags, one for each mapped area in the image. These tags specify the shape, pixel coordinates, and link page for all clickable areas.

The following code defines the picture shown above in Figure 7-13 as an image map with links to different on-page headings on a Web page.

```
<img src="MyMap.gif" alt="Image Map" usemap="#MyImage

<map name="MyImageMap">
  <area shape="rect" coords="8,13,130,123"
    href="#LINK1" alt="Determining Mapped Areas"/>
  <area shape="circle" coords="228,123,62"
    href="#LINK2" alt="Coding an Image Map"/>
  <area shape="poly" coords="100,144,175,200,155,255,
    href="#LINK3" alt="Overlapped Mapped Areas"/>
</map>
```

Listing 7-15. *Coding an image map.*

Note the relationship between usemap="#MyImageMap" in the `<img>` tag and name="MyImageMap" in the `<map>` tag. The id "MyImageMap" associates the image map with the graphic image. Also, when coding the final page, both the `<img>` and the `<map>` tags must appear inside a block-level tag such as a `<p>` or `<div>` .

The `<area>` tags give the coordinates (coords) to define the individual shapes and to associate a URL with a mouse click on that shape. In this example, the URLs are on-page links, and you can link to other pages at your site or to external Web sites.

- When **shape="rect"** (rectangle), the coords are specified by four numbers separated by commas. This notation represents two pairs of *h*,*v* coordinates, the first pair giving the coordinates of the top-left corner of the rectangle, and the second pair giving the coordinates of the bottom-right corner (8,13,130,123).
- When **shape="circle"**, the coords are specified by three numbers separated by commas. The first two numbers represent the *h*,*v* coordinates of the center point of the circle and the last number is the pixel width of the radius (228,123,62).
- When **shape="poly"** (polygon), the coords are specified by as many pairs of *h*,*v* coordinates as there are points on the polygon. Number pairs can be listed clockwise or counter-clockwise around the polygon; they are separated by blank spaces (100,144,175,200,155,255,50,250 45,175).

The `<map>` tag can appear anywhere on the page. Wherever it might be located in the body of the document it is associated with the proper `<img>` tag through the map name.

Even if you produce image maps with the alt attribute coded for each clickable area along with alt text for the `<img>` tag itself, it is still a good idea to provide a set of comparable text links for visitors with text reader software or for those who have graphics turned off in their browsers.

Overlapped Map Areas - TOP

If necessary, you can have overlapping mapped areas. When the mouse is clicked in the overlap area, the link that takes precedence is the one associated with the shape that is coded first in the `<map>` tag.

In the example in Figure 7-14, a click in the overlap area links to the URL specified for shape="rect". Its definition appears prior to shape="circle" in the image map even though the graphic of the circle overlays the graphic of the rectangle.

```
<map name="ImageMap">
  <area shape="rect" ...>
  <area shape="circle" ...>
</map>
```

Figure 7-14. *Precedence of overlapped image map areas.*

You might also wish to assign a link to the area of an image that is not specifically mapped to any of the shapes. A link associated with this "background" shape should appear last in the list of `<area/>` tags so that it does not take precedence over other shapes as an overlapping image. The coordinates for the background would encompass the entire rectangular area of the picture.

Coding for an Image Map Page - TOP

The following listing shows image map and link coding for the page described in the current example. The text paragraphs are not fully coded. Only the on-page links to these sections of the page are coded fully.

```
<!DOCTYPE html>

<html lang="en">
><head>
  <title> Using Image Maps </title>

<style>
 body    {margin:20px}
 .center {text-align:center}
</style>
```

```
</head>
<body>

<h2 class="center"> Using Image Maps </h2>

<p class="center">
<img src="MyMap.gif" usemap="#MyImageMap" alt="Image

<map name="MyImageMap">
  <area shape="rect" coords="8,13,130,123"
    href="#LINK1" alt="Determining Mapped Areas>"
  <area shape="circle" coords="228,123,62"
    href="#LINK2" alt="Coding an Image Map" >
  <area shape="poly" coords="100,144 175,200 155,255
    href="#LINK3" alt="Overlapped Mapped Areas">
</map>
</p>

<p class="center">
<a href="#LINK1"> Determining Mapped Areas </a>  |
<a href="#LINK2"> Coding an Image Map </a>  |
<a href="#LINK3"> Overlapped Mapped Areas </a>
</p>

<h3>  <a name="LINK1"> Determining Mapped Areas </a>
 ...text...
<p>  <a href="#"> Top </a></p>


<h3>  <a name="LINK2"> Coding an Image Map </a></h3>
 ...text...
<p><a href="#"> Top </a></p>


<h3><a name="LINK3"> Overlapped Mapped Areas </a></h3
 ...text...
<p><a href="#"> Top </a></p>
```

```
    </body>
    </html>
```

Listing 7-16. *Coding for image map page.*

This example uses on-page links for the image map. The href attribute of an `<area>` tag can likewise open an external Web site in the same or in a new browser window.

8.3: Image Maps is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

8.4: Redirecting Pages

You probably have visited Web sites at which you are automatically redirected from one page to another. This technique is often used when a home page is moved from one URL to another. For a certain period of time, the original URL redirects to the new URL until users have a chance to change their bookmarks or link references.

The `<meta>` Tag

Redirection is provided with a `<meta>` tag which appears in the `<head>` section of the document, and in general, is used to provide information about the HTML document. A general format for this redirection type tag is shown below.

```
<meta
  http-equiv="refresh"
  content="seconds; url=url"
>
```

Listing 7-17. *General format for* `<meta>` *tag.*

The **http-equiv** attribute identifies this use of the `<meta>` tag as a refreshing, or reloading, of the current page. The **content** attribute specifies the number of seconds to wait before reloading takes place, along with the URL of the page to which redirection is to take place when the current page is reloaded.

For example, the following link goes to a page named Redirect.htm located in the current directory. The Redirect.htm page is coded with a meta tag that redirects back to this page after five seconds.

Redirect Page

In the `<head>` section of the Redirect.htm page is the following `<meta>` tag to link back to this page.

```
<head>
  <title> Redirect Page </title>
  <meta http-equiv="refresh" content="5; url=redirect
</head>
```

Listing 7-18. *Coding to automatically redirect to a different page.*

A Slide Show - TOP

A `<meta>` tag with http-equiv and content attributes can also be coded on a series of pages to provide a self-running slide show that automatically loads one page after another. Each `<meta>` tag specifies the viewing time for the page along with a link to the next page in sequence.

The following graphic link loads the first (Slide1.htm) of five pages of a slide show that are all located in the current directory. This initial link is coded below.

```
<a   href="Slide1.htm"  ><img src="Pixar.gif" alt="Vie
```

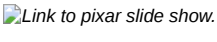


Figure 7-15. *Link to slide show.*

Each of the slide show pages contains a `<meta>` tag giving the URL of the next page in sequence and setting the refresh timer for three seconds. The following abbreviated code appears on the slide pages.

```
<head>
  <title> Slide 1 </title>
  <meta http-equiv="refresh" content="3; url=Slide2.h
</head>
<body>
  
  <h1> Toy Story </h1>
  <p><img src="ToyStory.jpg" alt="Toy Store Image"></
</body>

<head>
  <title> Slide 2 </title>
  <meta http-equiv="refresh" content="3; url=Slide3.h
</head>
<body>
  
  <h1> A Bug's Life </h1>
  <p><img src="BugsLife.jpg" alt="A Bugs's Life Image
</body>

<head>
  <title> Slide 3 </title>
  <meta http-equiv="refresh" content="3; url=Slide4.h
</head>
<body>
  
  <h1> Finding Nemo </h1>
  <p><img src="FindingNemo.jpg" alt="Finding Nemo Ima
</body>

<head>
  <title> Slide 4 </title>
  <meta http-equiv="refresh" content="3; url=Slide5.h
</head>
<body>
  
```

```
    <h1> Monsters, Inc. </h1>
    <p><img src="MonstersInc.jpg" alt="Monsters, Inc. I
  </body>

  <head>
    <title> Slide 5 </title>
    <meta http-equiv="refresh" content="3; url=xhtml07-
  </head>
  <body>
    <h1> Toy Story 2 </h1>
    <p><img src="ToyStory2.jpg" alt="Toy Store 2 Image"
  </body>
```

Listing 7-19. *Partial coding for slide show pages.*

The final page of the slide show redirects back to this tutorial page containing the original link to the first slide.

If you want to run the show in a separate browser window, then the original link opens a new window.

```
  <a href="Slide1.htm" target="_blank">
    <img src="Pixar.gif" alt="View Slide Show">
  </a>
```

Listing 7-20. *Opening a slide show in a separate browser window.*

Note that since a `<meta>` tag cannot close a browser window, the final slide page that is opened in this separate window should *not* contain a `<meta>` tag to redirect back to the original page, thereby resulting in the original page being opened in two separate windows: in the original window and in the separate slide-show window. Rather, the final page of the slide show should inform visitors simply to close the slide show window to reveal the original window containing the original link page.

8.4: Redirecting Pages is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

8.5: Traversing Folders

As you accumulate a large number of Web documents, you will likely begin organizing different Web applications within separate directories subordinate to your root directory -- subordinate to your main Web directory. Figure 7-16 illustrates this idea with three subdirectories within root directory Website, each containing Web pages. In addition, application3 folder contains a Media subdirectory for storing all graphic files pertaining to that application.



Figure 7-16. Example Web directory structure.

When coding the src attribute in an `<img>` tag or the href attribute in an `<a>` tag, you need to trace the directory hierarchy from the Web page containing the tag to the location of the actual image or page specified in the link. This means that you may need to go *down* one or more levels of directories to find the picture or linked page; or that you need to go *up* the hierarchy to find the picture or page; or that you need to go *up* the hierarchy into a higher-level directory and then *down* into the folder containing the image or link.

The reason for this up and down traversing of directories is because linked pages or images are referenced relative to the page containing the link. By way of illustration, suppose that PageA.htm in directory Application1 contains an `<img>` tag pointing to Picture1.gif in the Media subdirectory of directory Application 3. In order to make the src reference from PageA.htm to Picture1.gif, the following path is followed:

- Go up one level from the Application1 folder to the WebSite folder.
- Go down one level from the WebSite folder to the Application3 folder.
- Go down one more level from the Application3 folder to the Media folder.

Two types of path specifications are used to traverse a directory hierarchy to locate Web pages and graphic images:

- Code two dots followed by a forward slash ("../") for each level traversed up the folder hierarchy.
- Code a folder name followed by a forward slash ("/") for each level traversed down the folder hierarchy;

For example, the URL for an `<img>` link from PageA.htm to Picture1.gif would be

```
<img src="../Application3/Media/Picture1.gif">
```

Listing 7-21. *Linking to a graphic file by traversing directories.*

From the current directory containing PageA.htm, back up one level (../) from the Application1 directory to the WebSite directory. From the WebSite directory, go down through the Application3/ and Media/ directories to find Picture1.gif.

Normally, path specifications are not as complex as this. It is common for a folder containing Web pages to compile all images used on those pages into a single Media subfolder inside the page folder. Then, a src link to an image would only need to point down one level of directories:

```
    <img src="Media/Picture.gif">
```

Listing 7-22. *Linking to a graphic file in a subdirectory.*

The above examples connect from a Web page to an image. The same process is followed when linking from one page to another. In the above directory structure, a link appearing on PageA.htm to Page2.htm would be

```
    <a href="../Application3/Page2.htm"> To to Page2.htm
```

> *Listing 7-15. Linking to a Web page in a different folder.*

It is fairly simple to trace the path from a Web page to a graphic image or other Web pages if you focus on the folder structure by starting with the folder containing the page and ending at the folder containing the file. Remember to pay attention to the directory tree. You must follow the hierarchy up and down the branches of the tree rather than make lateral references where no "lines' connect the folders.

8.5: Traversing Folders is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

9: Using Tables

9.1: Creating Tables
9.2: Table Borders
9.3: Table Captioning
9.4: Table Captioning
9.5: Table Size and Alignment
9.6: Aligning Text and Cells
9.7: Column and Row Spanning
9.8: Table Groupings

9: Using Tables is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

9.1: Creating Tables

Tabular arrangements of data into rows and columns can help the viewer sort through masses of data to understand their underlying structure and content. Therefore, tables are useful devices for presentation of information in a meaningful form. At the same time, tables have historically been used to structure the layout of a Web page; however, this approach is becoming deprecated in favor of CSS-based layouts. In short, tables are important presentation devices and should be used to present tabular data. They should not be used to define the layout of a Web page.

The `<pre>` Tag

Previously you were introduced to the `<pre>` tag as one method for arranging data into rows and columns for tabular presentation. A block of text surrounded by the `<pre>` tag is displayed in a monospace font in precisely the way it is typed in the text editor. The tag is often used to create simple tables where alignment of columns is produced with embedded blank spaces. For example, the following editor code shows a table produced by using the keyboard Enter key to create separate rows, and the Space Bar key to align columns of data. This table layout appears on a Web page in exactly the format it is typed inside the `<pre>` tags.

```
<pre>
                  Table 1
              Sales by Region
-----------------------------------------------
Region/Year   2000     2001     2002     2003
-----------------------------------------------
East           35.2     37.4     39.8     40.0
West           28.0     25.6     27.4     29.8
South         102.3     86.1     98.6    100.2
North          10.5      8.6      9.8     10.4
-----------------------------------------------
</pre>
```

Figure **9.1.1**: Code to create a table using the `<pre>` tag.

This method provides limited capabilities for arranging and styling a table. It is useful only for producing fairly simple tables without the need for styling it in conformance with the overall page design. A better method is to construct tables using special HTML tags designed for that purpose.

A Simple Table

Like any other tabular presentation, a table produced with HTML table tags contains rows and columns. The intersections of these rows and columns form the **cells** of the table. Information can then be placed into these cells. A simple 3 x 3 table is shown in Figure 8-1.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.1.2**: A 3 x 3 table.

The rows, columns, and cell intersections of a table are defined with three basic tags. A `<table>` tag surrounds the entire table description; `<tr>` (table row) tags defined the rows of the table; and `<td>` (table data) tags define the cells, or columns, that appear along each row. The `<td>` tag encloses the content that appears in a cell. These tags are shown in their general structural formats in Figure 8.2.

```
<table>
  <tr>
    <td>cell content</td>
    <td>cell content</td>
    ...
  </tr>
  ...
</table>
```

Figure **9.1.3**: General formats for basic table tags.

There are as many `<tr>` tags as there are rows in the table, and there are as many `<td>` tags as there are cells (columns) in each row. When building tables, you need to be cautious about properly pairing opening and closing tags. One minor oversight, and the table arrangement can be totally disrupted. Also, take care to organize your coding so that the table structure is clearly evident. For example, the following code is used to create the table shown above in Figure 8.1.

```
<table>
<tr>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
</tr>
</table>
```

Figure **9.1.4**: Code to create a 3 x 3 table.

Note that `<table>` , `<tr>` , and `<td>` tags are container tags that all require both an opening and closing tag. The `<table>` tag encloses all coding for the table; `<tr>` tags enclose each row of the table; `<td>` tags enclose each cell of a table row.

The code in Listing 8-2 is arranged in a line-by-line hierarchy of tags suggestive of the structure of the table. This manner of coding is helpful in visualizing the table, although you can code table tags in free-format style, like any other HTML coding. For instance, the above table displays correctly coded in the following fashion.

```
<table>
<tr><td>Cell 1.1</td><td>Cell 1.2</td><td>Cell 1.3</td></tr>
<tr><td>Cell 2.1</td><td>Cell 2.2</td><td>Cell 2.3</td></tr>
<tr><td>Cell 3.1</td><td>Cell 3.2</td><td>Cell 3.3</td></tr>
</table>
```

Figure **9.1.5**: Alternate code to create a 3 x 3 table.

However, since table cells (`<td>` ... `</td>`) can enclose information of varying length and complexity, it is best to code them on separate lines as in Listing 8-2 for more accurate visualization and typing of cell contents.

The table below presents a summary of the HTML tags used for creating tables by discussing what was covered in this tutorial.

Tag	Description
table	Defines a table. All other table elements are coded within this element.
tr	Defines a table row.
td	Defines a table data cell within a row.
th	Defines a table header cell within a row.
caption	Contains a description of the table. The element must be coded immediately after the opening table tag.
thead	Groups one or more rows or tr elements into a table header.
tfooter	Groups one or more rows or tr elements into a table footer.
tbody	Groups one or more rows or tr elements between the header and footer into a table body.

Table and Cell Borders

By default, table tags produce no borders around the table or around its cells. The code in Listing 8-2 and 8-3 actually create the table layout shown below in Figure 8-3. Data are aligned within rows and columns; however, no borders are displayed. In certain cases, you may not wish to display table borders, but in most cases you will. Therefore, style sheets must be applied to the table and to its cells to display borders around both.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.1.6**: A standard 3 x 3 table.

A style sheet is applied to the table *and* to its cells to display borders around both. A border surrounds the entire table by styling the `<table>` tag. Borders surround individual cells by styling the `<td>` tag. The code below produces the bordered table shown in Figure 8.1. The entire table is surrounded by a 1-pixel outset border and each of the cells is surrounded by a 1-pixel inset border. This styling is typical for a standard HTML table.

```
<style type="text/css">
  table {border:outset 1px;}
  td    {border:inset 1px;}
</style>

<table>
<tr>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
</tr>
</table>
```

Figure **9.1.7**: Code to create a bordered 3 x 3 table.

An embedded style sheet is the most convenient method of styling table elements. In-line style sheets can be used; however, individual style sheets would have to appear in *each* of the many `<td>` tags. It is much easier declaring embedded styling for table and td selectors. Tables can have any style of outer borders, and cells can have there own border styles. Border styling is covered in a following tutorial.

Table Size

By default, the size of a table depends on the size of the data appearing in its cells. A column is as wide as the widest entry in the column. All rows are as wide as the combined width of data within the widest cells. In effect, table cells collapse around the data they contain. Although it is not apparent in the above examples, data are aligned horizontally left and vertically centered within the cells. Style settings to manage table and cell sizes and data alignment are described later in this tutorial.

Nested Tables

Tables can be nested; that is, a table can appear inside a cell of another table. This arrangement does not involve any particular coding complication, Table specifications are simply inserted as contents of a cell. The cell expands in size to permit full display of the nested table as shown in the following table and HTML code.

Cell 1	Cell 2
Cell 3	Cell A Cell B Cell C Cell D

Figure **9.1.8**: Nested tables.

```
<style type="text/css">
  table {border:outset 1px;}
  td    {border:inset 1px;}
</style>

<table>
<tr>
  <td>Cell 1</td>
  <td>Cell 2</td>
</tr>
<tr>
  <td>Cell 3</td>
  <td>

    <table>
          <tr>
            <td>Cell A</td>
            <td>Cell B></td>
          </tr>
          <tr>
            <td>Cell C</td>
            <td>Cell D</td>
          </tr>
      </table>

  </td>
</tr>
</table>
```

Figure **9.1.9**: Code for nested tables.

In this example, you can also see the default alignment of data within cells. Notice in "Cell 2" that the entry is horizontally aligned at the left of the cell; in "Cell 3" the entry is vertically aligned in the middle of the cell.

Styling Multiple Tables

Since tables provide one of the most effective methods for arranging content on a Web page, it is often the case that a single page contains two or more tables that are either nested tables or multiple tables containing different data content. It is also often the case that these tables need to be styled differently. Therefore, the embedded style sheet must differentiate among the tables so that each can have its own special styling.

Recall from earlier discussions that page elements can be assigned id values to uniquely identify them. These ids are then used in the embedded style sheet to indicate which styles pertain to which unique elements. This technique can be used to apply different styles to different tables on the same page.

For example, the two tables in Figure 8-5 have different border styles and sizes. The first table is styled with standard borders; the second table uses the outset style for both table and cell borders and uses different border widths.

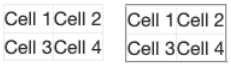


Figure 9.1.10: Styling multiple tables on the same page.

The style sheet and table definitions for these two tables are shown in Listing 8-6.

```
<style type="text/css">
  table#Table1    {border:outset 1px;}
  table#Table1 td {border:inset 1px;}

  table#Table2    {border:outset 3px;}
  table#Table2 td {border:outset 2px;}
</style>

<table   id="Table1"  >
<tr>
  <td>Cell 1</td>
  <td>Cell 2</td>
</tr>
<tr>
  <td>Cell 3</td>
  <td>Cell 4</td>
</tr>
</table>

<table   id="Table2"  >
<tr>
  <td>Cell 1</td>
  <td>Cell 2</td>
</tr>
<tr>
  <td>Cell 3</td>
  <td>Cell 4</td>
</tr>
</table>
```

Figure **9.1.11**: Code to style multiple tables on the same page.

ID selectors are used to separately style the `<table>` tags for the two tables. Contextual selectors are used to style their `<td>` tags. In this way, any number of tables can take on different styles without styling conflicts. (See tutorial "Applying Special Styles" for a review of ID selectors; see "Contextual Selectors" to review these style selectors.)

This introduction to tables has given you a general overview of tables and style sheet approaches to styling them. The following tutorials cover detailed aspects of table design and formatting.

9.1: Creating Tables is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

9.2: Table Borders

When a table is defined by a `<table>` tag with no styles specified, a borderless table as wide and as tall as its enclosed data is displayed. From this simple beginning, you can structure and style a table to take on virtually any display characteristics you can imagine.

Table Border Styles

Any of the border styles that were introduced earlier can be applied to the outer border of a table. Figure **9.2.2** shows example border styles coded in the general fashion giving in Listing 8-7.

```
<style type="text/css">
  table {border:style 5px;}
  td    {border:inset 1px;}
</style>
```

Figure **9.2.1**: General style sheet code for table borders.

The border *style* is identified in Figure **9.2.2** for each of the tables. The style sheet uses the general format {border:*style size;*} for all border stylings.

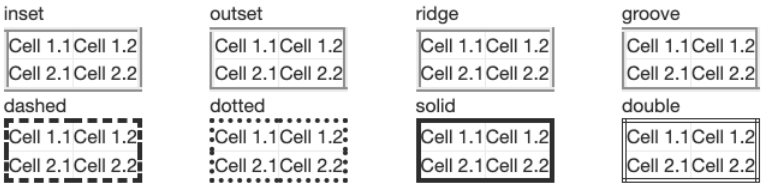


Figure **9.2.2**: Varieties of table borders

All tables above have a outer border of 5 pixels to make then visually obvious. Different effects can be achieved with different border sizes.

Individual Border Styles

In the above examples, borders appear around all four sides of the table; however, you can specify individual styling for the separate sides. Use the properties border-top, border-bottom, border-left, and border-right to selectively apply borders around the table. For example, the table in Figure **9.2.4** displays only top and bottom borders with the style sheet given in Listing 8-8.

```
<style type="text/css">
  table {  border-top:outset 5px; border-bottom:outset 5px;  }
  td    {border:inset 1px;}
</style>
```

Figure **9.2.3**: Style sheet for top and bottom table borders.

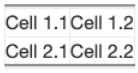


Figure **9.2.4**: Table with top and bottom borders.

Collapsed Borders

Borders surrounding the cells in a table normally are displayed independently from each other. That is, each cell is surrounded by its own border thereby giving a "double border" look to side-by-side cells. These double borders can be collapsed into single

borders with the border-collapse style property applied to the table.

Property	Values
border-collapse	separate collapse

Figure 8-8. Table border-collapse style property.

Figure **9.2.5**: Table border-collapse style property.

The default border setting for a table is border-collapse:separate. By specifying border-collapse:collapse, cells share "single borders" between them. This effect is illustrated in the tables in Figure **9.2.7** styled with collapsed borders.

```
<style type="text/css">
  table {border:style 5;   border-collapse:collapse;  }
  td    {border:solid 1px;}
</style>
```

Figure **9.2.6**: Style sheet to collapse table borders.

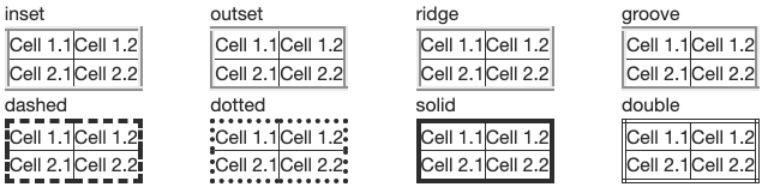


Figure **9.2.7**: Tables with collapsed cell borders.

Note that collapsed borders are specified for the `<table>` tag even though the setting is applied to the table's cells. Also, you cannot selectively apply collapsed border style to individual cells. When table borders are collapsed, all cell borders are styled as border:solid 1px; other styles and widths are not recognized.

Cell Border Styles

If cell borders are not collapsed, the same set of styles that are applied to the table's outer border can be applied to its individual cells. In the examples in Figure **9.2.8**, the outer borders of the tables are 1px in width and cell borders are 5px in width to emphasize their styling.

```
<style type="text/css">
  table {border:solid 1px;}
  td    {border:style 3px;}
</style>
```

Figure **9.2.8**: General style sheet code for cell borders.

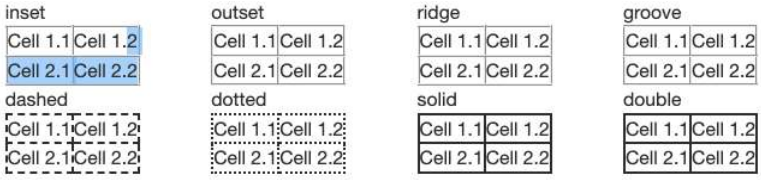


Figure **9.2.9**: Varieties of cell borders.

Selective Border Styles

Borders surrounding cells can be selectively styled with border-top, border-bottom, border-left, and border-right styles applied to the `<td>` tag. These styles are normally used to display top-and-bottom or left-and-right borders to emphasize the rows or columns of a table. The two tables shown in Figure **9.2.11** are displayed with the style sheets given in Listing 8-11.

```
<style type="text/css">
  table {border:outset 5px; border-collapse:collapse;}
  td    {  border-top:solid 2px; border-bottom:solid 2px;  }
</style>

<style type="text/css">
  table {border:outset 5px; border-collapse:collapse;}
  td    {  border-left:solid 2px; border-right:solid 2px;  }
</style>
```

Figure **9.2.10**: Style sheets to apply row and column borders.

top-and-bottom

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

left-and-right

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.2.11**: Application of row and column borders.

In order to extend row and column borders across the entire table, borders must be collapsed. Otherwise, breaks in the lines appear between the separate cells. As mentioned above, collapsed borders also means that the various cell border styles do not display. Therefore, a solid border style is the only one that need be used for the cells.

Individual Border Styles

Cell borders can be styled independently. If needed, each cell can have a different border style. Usually, only selected cells are given emphatic borders to highlight their contents. Of course, in order to view cell borders, the table border cannot be collapsed.

The following table displays different cell borders along its bottom row.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.2.12**: Emphasizing a table row with cell borders.

The above effect can be achieved by applying in-line style sheets to the selected cells. However, this is the type of situation especially made for the use of ID selectors and contextual selectors. Complete table and style sheet coding is shown below.

```
<style type="text/css">
  table#TAB1            {border:outset 3px;}
  table#TAB1 td         {border:inset 1px;}
  table#TAB1 tr#ROW3 td {border:outset 3px;}
</style>

<table   id="TAB1"  >
<tr>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
<tr   id="ROW3"  >
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
</tr>
</table>
```

Figure **9.2.13**: Code to emphasizing a table row with cell borders.

The table itself is assigned an id in order to uniquely identify it among other tables on the page. Moreover, the final row of the table is given an id to uniquely identify it among the other rows of the table. This is the row in which special border styling is applied to its cells.

Within the style sheet, the table is given an outer border style through the ID selector table#TAB1, selecting the `<table>` tag with id="TAB1" for application of the border style. The contextual selector table#TAB1 td applies a 1-pixel inset border to all cells of this table -- i.e., where a `<td>` tag is contained within a `<table>` tag for a table with id="TAB1".

A contextual selector is also used for styling the last row of the table. A 3-pixel outset border is applied with the selector table#TAB1 tr#ROW3 td -- i.e., where a `<td>` tag is contained within a `<tr>` tag with id="ROW3", which is contained within a `<table>` tag with id="TAB1".

If only a single cell has its borders styled, then that cell can be given an id, such as, id="CELL9". Under such conditions, the selector for styling that one cell would be

table#TAB1 td#CELL9 {border:outset 3px;}

where a `<td>` tag with id="CELL9" is contained within a `<table>` tag with id="TAB1". In this case, it would not be necessary to identify the table row.

Deprecated Table Attributes

Borders are displayed around the table and between table cells with the border="n" attribute of the `<table>` tag, where n is the width of the outer border in pixels. The following table has 1-pixel borders:

<table border="1">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Changing the width of the outer border does not effect the size of cell borders. The default style of the outer border is "outset"; the default style of inner borders is "inset":

<table border="5">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

9.2: Table Borders is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

9.3: Table Captioning

The <caption> Tag

You can caption a table by coding a <caption> tag immediately following the <table> tag. Enter a table title between the opening and closing <caption> tags and the title displays centered over the table. The caption is displayed in the page default font style and size, and formatting is applied with style sheets as is done in the following code example for the table appearing in Figure 8.13.

This is My Table

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure 9.3.1: A table with a caption.

```
<style type="text/css">
  table         {border:outset 3px;}
  table td      {border:inset 1px;}
  table caption {font:bold 10pt arial;}
</style>

<table>
  <caption>This is My Table</caption>
  <tr>
    <td>Cell 1.1</td>
    <td>Cell 1.2</td>
    <td>Cell 1.3</td>
  </tr>
  <tr>
    <td>Cell 2.1</td>
    <td>Cell 2.2</td>
    <td>Cell 2.3</td>
  </tr>
  <tr>
    <td>Cell 3.1</td>
    <td>Cell 3.2</td>
    <td>Cell 3.3</td>
  </tr>
</table>
```

Figure 9.3.2: Code for a table with a caption.

You can apply styling to the caption through a caption selector. In this example, font styling is applied with the shorthand font notation,

font:*style variant weight size family*

The caption is centered by default. The text-align property can be applied to the caption to align it with the left or right edge of the table.

Captions appear only at the top of the table. Style sheet standards propose the caption-side:top|bottom|left|right style declaration for positioning captions; however, this property has not been implemented in current browsers.

The `<th>` Tag

Column headings can be supplied with `<th>` tags in place of `<td>` tags enclosing cell contents. The `<th>` tag automatically centers and bolds the enclosed text. Additional styles can be applied to the tags for additional formatting. Notice in the following example that the th selector is added to the style sheet for borders surrounding these cells.

This is My Table

Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.3.3**: Using the `<th>` tag for column headings.

```
<style type="text/css">
  table         {border:outset 3px; border-collapse:collapse;}
  table td,   th    {border:inset 1px;}
  table caption {font:bold 10pt arial;}
</style>

<table>
  <caption>This is My Table</caption>
  <tr>
    <th>  Column 1  </th>
    <th>  Column 2  </th>
    <th>  Column 3  </th>
  </tr>
  <tr>
    <td>Cell 1.1</td>
    <td>Cell 1.2</td>
    <td>Cell 1.3</td>
  </tr>
  <tr>
    <td>Cell 2.1</td>
    <td>Cell 2.2</td>
    <td>Cell 2.3</td>
  </tr>
  <tr>
    <td>Cell 3.1</td>
    <td>Cell 3.2</td>
    <td>Cell 3.3</td>
  </tr>
</table>
```

Figure **9.3.4**: Code to produce column headings for a table.

The `<th>` tag is unique only in that it provides default bold and centered styling for its contents. With style sheets, however, this tag becomes redundant since this same styling can be applied without using this special tag. The following table duplicates the previous table without `<th>` tags simply by identifying and styling the first row of the table.

This is My Table

Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.3.5**: Producing column headings by styling a row.

```
<style type="text/css">
  table         {border:outset 3px; border-collapse:collapse;}
  table tr#HEAD {font:bold; text-align:center;}
  table td      {border:inset 1px;}
  table caption {font:bold 10pt arial;}
</style>

<table>
  <caption>This is My Table</caption>
    <tr id="HEAD">
    <td>Column 1</td>
    <td>Column 2</td>
    <td>Column 2</td>
  </tr>
  <tr>
    <td>Cell 1.1</td>
    <td>Cell 1.2</td>
    <td>Cell 1.3</td>
  </tr>
  <tr>
    <td>Cell 2.1</td>
    <td>Cell 2.2</td>
    <td>Cell 2.3</td>
  </tr>
  <tr>
    <td>Cell 3.1</td>
    <td>Cell 3.2</td>
    <td>Cell 3.3</td>
  </tr>
</table>
```

Figure **9.3.6**: Code to produce column headings by styling a row.

The contextual selector tr#HEAD td indicates styling for td tags that appear inside a tr tag with id="HEAD". It is your choice whether to use the th tag to take advantage of its inherent styling or to manage all styling through style sheets. As a general rule, the fewer tag to deal with the better.

Deprecated Caption align Attribute

Captions can appear at the top or bottom of a table using the align="top" (default) or align="bottom" attribute of the `<caption>` tag.

Top Table Caption

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

e.g., <caption align="top">Top Table Caption</caption>

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Bottom Table Caption

e.g., <caption align="bottom">Bottom Table Caption</caption>

As noted above, style sheet standards propose the caption-side property to replace the align attribute; however, this property has not been implemented in current browsers.

9.3: Table Captioning is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

9.4: Table Captioning

Tables can have outer border colors as well as cell border colors. They also can have background colors, patterns, and graphic images as backdrops for the full table or for selected rows and cells.

Table Border Colors

The appearance of border colors is different depending on the border style. Below are example border styles displayed in red.

```
<style type="text/css">
  .inset { border:inset 5px red; }
  .outset { border:outset 5px red;}
  .ridge { border:ridge 5px red; }
  .groove { border:groove 5px red;}
  .dashed { border:dashed 5px red;}
  .dotted { border:dotted 5px red;}
  .solid { border:solid 5px red;}
  .double {border:double 5px red;}

  .inset td, .outset td, .ridge td, .groove td, .dashed td,
  .dotted td, .solid td, .double td {border:inset 1px #000;}
</style>
```

Figure **9.4.1**: General style sheet for table border colors.

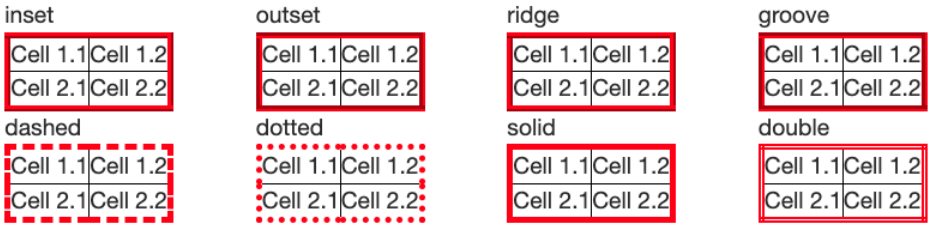


Figure **9.4.2**: Varieties of table border styles and colors.

You can create your own 3-D shading effects by styling each of the borders separately with border-top, border-right, border-bottom, and border-left colors. The following tables creates their own 3-D effects with solid borders tinted to resemble inset and outset styles. The style sheets used for these two tables are shown in Figure **9.4.4**.

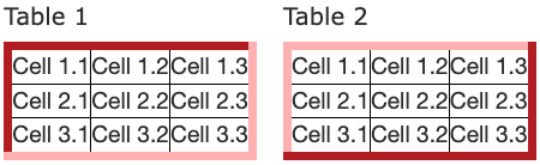


Figure **9.4.3**: Table border 3-D effects using border colors.

```
<style type="text/css">
  /* INSET STYLE */
    #TAB1 { border:solid 7px;
      border-color:#B22222 #FEB9B9 #FEB9B9 #B22222; }

    /* OUTSET STYLE */
    #TAB2 { border:solid 7px;
      border-top-color:#FEB9B9;
      border-right-color:#B22222;
      border-bottom-color:#B22222;
      border-left-color:#FEB9B9; }

    #TAB1 td, #TAB2 td { border:inset 1px #000; }
</style>
```

Figure 9.4.4: Style sheet code for tables with individual border colors.

Cell Border Colors

Borders around table cells are styled and colored independently from the outer border. In the following examples, the outer border is 1 pixel in width for deemphasis, and cell borders are red and 3 pixels in width.

```
<style type="text/css">
  table { border:solid 1px #000; }
    td    { border:style 3px red; }
</style>
```

Figure 9.4.5: General style sheet for table cell border colors.

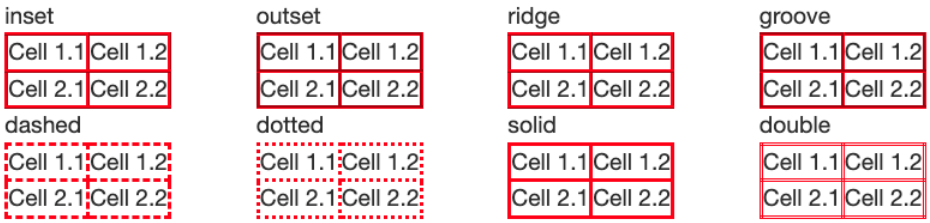


Figure 9.4.6: Varieties of table cell border styles and colors.

As in the case with table borders, cells borders can be styled and colored independently. In Figure 8.19, all cells in the two tables are styled with solid borders 5 pixels in width. Different shades of red are applied to their sides to create 3-D effects resembling inset and outset borders. The two applicable style sheets are shown in Listing 8-19.



Figure 9.4.7: Tables with individual cell border colors.

```
<style type="text/css">
  /* INSET STYLE */
  table    {border:solid 1px #000;}
  table td {border:solid 5px red;
            border-top-color:#FEB9B9;
            border-right-color:#B22222;
            border-bottom-color:#B22222;
            border-left-color:#FEB9B9;}
</style>

<style type="text/css">
  /* OUTSET STYLE */
  table    {border:solid 1px #000;}
  table td {border:solid 5px red;
            border-top-color:#B22222;
            border-right-color:#FEB9B9;
            border-bottom-color:#FEB9B9;
            border-left-color:#B22222;}
</style>
```

Figure **9.4.8**: Style sheet code for tables with individual cell border colors.

Table Background Colors

You can specify a background color for the whole table or select specific rows or cells to apply background colors. In the following table, a light gray background color is applied to the entire table by setting the background-color property of the `<table>` tag.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.4.9**: Table with a background color.

```
<style type="text/css">
  table    { border:ridge 5px red; background-color:#F6F6F6; color:#FFF; }
    table td { border:inset 1px #000; }
</style>
```

Figure **9.4.10**: Code for table with a background color.

Cell Background and Text Colors

The following table contains different background colors for each of its rows by specifying a background-color property for each identified `<tr>` tag. The text color of selected rows is set with the color property. In addition, one particular cell is assigned a background color that overrides the color of the row it occupies, and it is given a different border style.

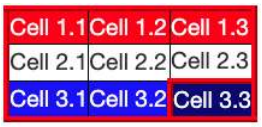


Figure **9.4.11**: Table with a row color.

```
<style type="text/css">
  table          {border:ridge 5px red;}
  table td       {border:inset 1px #000;}
  table tr#ROW1  {background-color:red; color:white;}
  table tr#ROW2  {background-color:white;}
  table tr#ROW3  {background-color:blue; color:white;}
  table td#CELL9 {border:inset 4px red; background-color:navy;}
</style>

<table>
  <tr   id="ROW1"  >
        <td>Cell 1.1</td>
        <td>Cell 1.2</td>
        <td>Cell 1.3</td>
  </tr>
  <tr   id="ROW2"  >
        <td>Cell 2.1</td>
        <td>Cell 2.2</td>
        <td>Cell 2.3</td>
  </tr>
  <tr   id="ROW3"  >
        <td>Cell 3.1</td>
        <td>Cell 3.2</td
        <td   id="CELL9"  >Cell 3.3</td>
  </tr>
</table>
```

Figure **9.4.12**: Code for table with row colors.

Table Backgrounds

In the same fashion used for background colors, you can specify a background image for a table or any of its rows or cells. In the following example, a textured background is applied to the entire table with the background-image property applied to the `<table>` tag. By default, the pattern is repeated across and down to fill the table area.

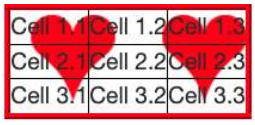


Figure **9.4.13**: Table with background image.

```
<style type="text/css">
  table    {border:ridge 5px red;   background-image:url(heart.png);  }
  table td {border:inset 1px #000;}
</style>
```

Figure 9.4.14: Code for table with background image.

If needed, other background properties -- background-repeat and background-position -- can be applied.

Deprecated Table Attributes

Border colors can be specified with the **bordercolor**="*color*" attribute of the `<table>` tag, where *color* is a color name or hexadecimal value.

<table border="5" bordercolor="red">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

A colored shadow effect can be applied to a table's outer borders with the **bordercolordark**="*color*" and **bordercolorlight**="*color*" attributes.

<table border="5" bordercolordark="red" bordercolorlight="pink">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

The background color of the table is given by the **bgcolor**="*color*" attribute. When applied to the `<table>` tag, the color fills the background.

<table border="5" bgcolor="lightblue">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Cell background colors are set by applying the bgcolor attribute to a `<tr>` tag (to color the row) or to a `<td>` tag (to color the cell). Cell colors override row colors which, in turn, override table background colors. The following table specifies a row color with one of its cells taking on a different background color.

```
<table border="5" bgcolor="gainsboro">
<tr bgcolor="lightblue">
  <td bgcolor="pink">
```

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

A background image is given by the **background**="*url*" attribute. Background images can be applied to individual table cells by coding the attribute in a `<td>` tag.

<table border="5" background="bkgnd.jpg">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

These table, row, and cell attributes are deprecated and should be substituted with equivalent style sheet properties.

9.4: Table Captioning is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

9.5: Table Size and Alignment

The sizes of cells in a table are governed by the size of the data within the cells. The table is as wide as the total of the widths of the widest cells. The table is as tall as the combined heights of the rows. In some cases, however, you may want to specify a width and/or height for a table irrespective of the data it contains.

Table Widths and Heights

The overall table width is supplied by the width property, and its height is given by the height property. Sizes can be measured in pixels -- to set exact sizes; or they can be expressed as percentages of the width and height of the browser window -- to vary the table size to correspond with the window size.

The table in Figure 8-23 is shown at its default width and height. It is sized no larger than is needed to display the data contained in its cells.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure 9.5.1: Table with default width and height.

This same table is redisplayed in Figure 8-24 with the specified width and height shown in the accompanying style sheet.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure 9.5.2: Table with specified width and height.

```
<style type="text/css">
  table    {border:outset 1;   width:50%; height:100px;  }
  table td {border:inset 1;}
</style>
```

Figure 9.5.3: Code for table with given width and height.

The table width is set to 50% of the page width. If you resize your browser window, you can see that the table width remains half as wide as the page. The height is specified as 100 pixels and remains at that height irrespective of the window size. Although it is common to specify table widths for visual design purposes, table heights are normally permitted to expand in size for as many rows as it contains.

Variable and Fixed Table Widths

A table's width can be set using percentages or pixels. A percentage width such as 50%, creates a variable-width table that the browser stretches to take up 50% of the browser window regardless of the screen resolution or window size. A table width of 100% can be used when you want the table to fill the entire browser window. Pixels can be used to set the width for a fixed-width table. This provides a more consistent look across different browsers. A table with a width of 500px will *only* take up 500 pixels of the browser window regardless of the screen resolution or window size. In many cases, fixed-width tables are centered on the page.

Column Widths

You can control column widths by specifying a size for any *one* cell in each column. This single cell width becomes the width to which all other cells in the column conform. The following table is sized to 50% of the width of the window. Subsequently, cells in the first row of the table (containing column headings in this case) are assigned width percentages. These percentages are of the *width of the table*, not of the page. Therefore, the sum of column widths should add up to 100%: the full width of the table.

Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure 9.5.4: Table with specified column widths.

```
<style type="text/css">
  table          {border:outset 1;   width:50%;  }
  table td       {border:inset 1;}
  table tr#HEAD  {font-weight:bold;
                  text-align:center;
                  background-color:#F0F0F0;}
  table td#CELL1 {  width:25%;  }
  table td#CELL2 {  width:50%;  }
  table td#CELL3 {  width:25%;  }
</style>

<table>
  <tr id="HEAD">
    <td   id="CELL1"  >Column 1</td>
    <td   id="CELL2"  >Column 2</td>
    <td   id="CELL3"  >Column 3</td>
  </tr>
  <tr>
    <td>Cell 1.1</td>
    <td>Cell 1.2</td>
    <td>Cell 1.3</td>
  </tr>
  <tr>
    <td>Cell 2.1</td>
    <td>Cell 2.2</td>
    <td>Cell 2.3</td>
  </tr>
  <tr>
    <td>Cell 3.1</td>
    <td>Cell 3.2</td>
    <td>Cell 3.3</td>
  </tr>
</table>
```

Figure **9.5.5**: Code for table with specified column widths.

It is not necessary to provide width percentages for all cells across a row. Any remaining cells on a row are sized equally by the browser so that the widths of all cells along the row total 100% of the width of the table. If cell widths are not sized sufficiently to display their data, the browser increases the widths to the minimum needed.

Rather than using percentages, you can specify table widths in pixels. However, you should typically use percentages to give the browser sufficient latitude in sizing the table to fit within the available window width.

You can set rows heights in the same way you set column widths: specify the height for *one* cell along the row and the other cells expand to conform to that height. Unless you have a reason for expanding the height of a row, you should let the browser determine the necessary height to display the data in the table.

A Table

Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure 9.5.6: A right-floated table.

Tables can be aligned to the left or right of the page by applying the float property to the table selector. When the table is floated, word wrapping takes place around the table in a fashion similar to the way word wrapping occurs around floated images.

Code for a floated table appears immediately before the text to be wrapped around the table. Margins can be added to sides of the table to leave white space between the table and the wrapped text. The following partial style sheet applies to the floated table shown in Figure 8-26.

```
<style type="text/css">
  table {border:outset 1;   float:right; margin-left:20px; margin-bottom:10px;  }
  ...
</style>
```

Figure 9.5.7: Partial style sheet for a floated table.

A table can be aligned to the left, right, or centered on a line by itself by using margin-left and margin-right properties. The table in Figure 8-27 shows a table centered by using margin-left:auto; margin-right:auto;.

A Centered Table

Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure 9.5.8: A centered table.

```
<style type="text/css">
  table         {border:outset 1;   margin-left:auto; margin-right:auto;  }
  table td      {border:inset 1; text-align:center;}
  table caption {font-weight:bold;}
  table tr#HEAD {font-weight:bold;
                 text-align:center;
                 background-color:#F0F0F0;}
</style>

<table>
  <caption>A Centered Table</caption>
    ...
</table>
```

Figure **9.5.9**: Style sheet for a centered table.

Deprecated Table Attributes

The width of a table is given by the **width**="*n*" attribute of the `<table>` tag, where *n* is a pixel width value or a percentage of the width of the browser window.

<table border="1" width="35%">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Cell widths can be set to a pixel value or percentage with the width attribute coded in a `<td>` or `<th>` tag. Cell heights are given with the **height**="*n*" attribute.

A table can be aligned at the left or right of the page with the **align**="left|right" attribute of the `<table>` tag. Text is wrapped around the positioned table. White space surrounding the table must be added with style sheet margin settings. A table with **align**="center" appears centered on a line by itself.

<table border="1" align="center">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

<table border="1" align="right">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

9.5: Table Size and Alignment is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

9.6: Aligning Text and Cells

Table cells are normally only as large as the data they contain. Their borders close in around the data. Often, this effect makes a table seem crowded and not particularly easy to read. Fortunately, you can introduce white space around data within table cells, and you can expand the amount of space between cells.

Cell Padding

Cell padding refers to the amount of blank space surrounding the text within a cell. Its value is set with padding applied to the cell selector in a style sheet. In the following table, padding of 5 pixels is placed around the data in the table cells.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.6.1**: A table with cell padding.

```
<style type="text/css">
  table    {border:outset 1px #000;}
  table td {border:inset 1px #000;   padding:5px  }
</style>
```

Figure **9.6.2**: Style sheet to apply padding to table cells.

Text Alignment

By default, text appears in a table aligned horizontally to the *left* of the cell and vertically within the *middle* of the cell. This alignment becomes noticeable when cell sizes are larger than needed to display their data.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.6.3**: A table sized to show text alignment within cells.

You can override these default horizontal and vertical alignments to position content anywhere within the cell.

Vertical Alignment

By using the vertical-align property, content can be aligned at the top, middle, or bottom of a cell. The table in Figure 8-31 applies these alignments separately to its three rows. Notice that the table is given width and height settings to make cells larger than needed so that vertical alignment of cell data is visually apparent.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure **9.6.4**: A table sized to show vertical alignment within cells.

```
<style type="text/css">
  table         {border:outset 1px #000; width:250px; height:150px}
  table td      {border:inset 1px #000}
  table tr#ROW1 {  vertical-align:top  }
  table tr#ROW2 {  vertical-align:middle  }
  table tr#ROW3 {  vertical-align:bottom  }
</style>

<table>
<tr   id="ROW1"  >
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</style>
<tr   id="ROW2"  >
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</style>
<tr   id="ROW3"  >
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
</tr>
</table>
```

Figure 9.6.5: Style sheet to apply vertical alignment of text within cells.

Horizontal Alignment

By using the text-align property, content can be aligned at the left, center, or right of a cell. The following table applies these alignments separately to its three rows that have vertical alignments like the previous table.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Figure 9.6.6: A table sized to show horizontal alignment within cells.

```
<style type="text/css">
  table         {border:outset 1px #000; width:250px; height:150px}
  table td      {border:inset 1px #000}
  table tr#ROW1 {vertical-align:top;   text-align:left  }
  table tr#ROW2 {vertical-align:middle;   text-align:center  }
  table tr#ROW3 {vertical-align:bottom;   text-align:right  }
</style>
```

Figure **9.6.7**: Style sheet to apply horizontal alignment of text within cells.

Empty Cells

Borders surround cells only when there is data inside the cells. If this does not occur, no borders are displayed. This effect can be seen in the first of the two tables in Figure 8-33 where data are missing from three of the cells. This may be an acceptable display on occasion, but usually borders should display even around empty cells. This situation can be handled by coding a non-break space character ( ) in the cells thereby producing the second of the table displays where all borders are visible.

	Cell 1.2	Cell 1.3
Cell 2.1		Cell 2.3
Cell 3.1	Cell 3.2	

	Cell 1.2	Cell 1.3
Cell 2.1		Cell 2.3
Cell 3.1	Cell 3.2	

Figure **9.6.8**: Tables with missing cell data.

```
<table>
<tr>
  <td>     </td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>     </td>
  <td>Cell 2.3</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>     </td>
</tr>
</table>
```

Figure **9.6.9**: Using non-break space characters to display borders around empty cells.

Non-wrapped Table Cells

By default, text within cells is word wrapped to fit within the overall size of the table. Normally, this is acceptable, to let the browser arrange cell content for best fit with the cells. For certain cells, you may not wish the browser to wrap its content. Such an example might be a row of column headings or a column of row labels. The following table shows default sizing and alignment of cell contents with non-wrapped display of column and row headings.

	Table Column 1	Table Column 2	Table Column 3
Table Row 1	Cell 1.1	Cell 1.2	Cell 1.3
Table Row 2	Cell 2.1	Cell 2.2	Cell 2.3

Figure **9.6.10**: Table with properly spaced row and column headings.

The problem arises when the table is resized. Such a problem may occur when the browser window is sized smaller than the width of the table needed for non-wrapped presentation of data. When this happens, the headings may wrap inside their cells which leads to the table display shown below.

	Table Column 1	Table Column 2	Table Column 3
Table Row 1	Cell 1.1	Cell 1.2	Cell 1.3
Table Row 2	Cell 2.1	Cell 2.2	Cell 2.3

Figure **9.6.11**: Table with improperly spaced row and column headings.

There is nothing wrong with the content of the table, but it may look quite different from what you intended. Text wrapping can be prevented in table cells by coding the **white-space** style property for any cell in which you do not wish text to be wrapped.

Property:	Value
white-space	normal nowrap pre

Figure **9.6.12**: The white-space style property.

This style property determines how the browser should handle white spaces within a text string. The normal setting collapses all contiguous spaces to a single space, and breaks the line of text at a blank space where the remaining text on the line does not fit within the specified width of its container. In contrast, the nowrap setting treats spaces as non-break space characters ( ) and does not break the line. The pre setting works like a `<pre>` tag, and retains as many blanks spaces as are coded by again treating them as non-break spaces.

If a table cell is not wide enough to display its text on a single line, then the text normally is wrapped at a blank space within the text string. This occurs with the column and row headings in the table in Figure 8-35. By setting the white-space:nowrap property for these table cells, wrapping of headings can be avoided. These specifications are made in recoding the table as follows.

	Table Column 1	Table Column 2	Table Column 3
Table Row 1	Cell 1.1	Cell 1.2	Cell 1.3
Table Row 2	Cell 2.1	Cell 2.2	Cell 2.3

Figure **9.6.13**: Table with non-wrapped row and column headings.

The following listing shows the complete code for the above table with non-wrapped cells. In this case, style class .NOWRAP is defined with the property setting white-space:nowrap. All cells that should not permitting text wrapping are assigned to this class.

```
<style type="text/css">
  table            {border:outset 1px #000; width:230px}
  table td         {border:inset 1px #000; padding:3px}
  table tr#C-HEAD  {font-weight:bold; background-color:#F0F0F0; text-align:center}
  table td#R-HEAD1 {font-weight:bold; background-color:#F0F0F0; text-align:left}
  table td#R-HEAD2 {font-weight:bold; background-color:#F0F0F0; text-align:left}
  table td#BLANK   {background-color:#FFFFFF}
  .NOWRAP          {white-space:nowrap}
</style>

<table>
<tr id="C-HEAD">
  <td id="BLANK">   class="NOWRAP"  </td>
  <td   class="NOWRAP"  >Table Column 1</td>
  <td   class="NOWRAP"  >Table Column 2</td>
  <td   class="NOWRAP"  >Table Column 3</td>
</tr>
<tr>
  <td id="R-HEAD1"   class="NOWRAP"  >Table Row 1</td>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
<table>
<table>
  <td id="R-HEAD2"   class="NOWRAP"  >Table Row 2</td>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
</table>
```

Figure **9.6.14**: Using non-wrapping of table cells.

Deprecated Table Attributes

Text is horizontally aligned within cells with the **align**="left|center|right" attribute applied to the `<tr>` , `<td>` , or `<th>` tags. Text is vertically aligned within cells by applying the **valign**="top|middle|bottom" attribute to the cell.

Cell padding is specified with the **cellpadding**="*n*" (pixels) attribute coded in the `<table>` tag. Spacing between cells is given by the **cellspacing**="*n*" (pixels) attribute. The cellspacing attribute must be used until browsers adopt the proposed border-spacing style property. More information on this depreciated attribute is below.

<table border="1" cellpadding="5">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

<table border="1" cellpadding="5" cellspacing="5">

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Wrapping of text within cells can be controlled by coding **nowrap** or **nowrap**="true" for any table cell: `<td nowrap>` or `<td nowrap="true">` .

Cell Spacing

Cell spacing refers to the amount of space (in pixels) between the cells of the table, effectively, the width of the inner borders between cells. Although CSS standards propose the border-spacing property to adjust this spacing, current browsers do not recognize it. Therefore, cell spacing must be applied with the deprecated **cellspacing**="*n*" attribute of the `<table>` tag, specifying the number of pixels between cell borders. Although this is a deprecated attribute, it remains valid under HTML5 Strict standards. In the following example, cellspacing is set to 10px.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

```
<style type="text/css">
  table    {border:outset 1px #000}
  table td {border:inset 3px #000; padding:5px}
</style>

<table   cellspacing="10"  >
  ...
</table>
```

9.6: Aligning Text and Cells is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

9.7: Column and Row Spanning

Table cells can be combined to make one larger cell from two or more contiguous cells. A cell can span two or more columns or two or more rows. One use of spanning is to display a heading across several columns as shown by the heading cells in the following table.

Heading 1		Heading 2	
Cell 2.1	Cell 2.2	Cell 2.3	Cell 2.4
Cell 3.1	Cell 3.2	Cell 3.3	Cell 3.4

Figure **9.7.1**: A table with cell padding.

The colspan Attribute

In the above example, the first row of the table is used for column headings. The cell specifications use the **colspan**="*n*" attribute to extend each of the `<td>` heading cells across 2 cells.

```
<style type="text/css">
  table         {border:outset 1px}
  table td      {border:inset 1px; padding:3px}
  table tr#HEAD {font-weight:bold;
                 text-align:center;
                 background-color:#F0F0F0}
</style>

<table>
<tr id="HEAD">
  <td   colspan="2"  >Heading 1</td>
  <td   colspan="2"  >Heading 2</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
  <td>Cell 2.4</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
  <td>Cell 3.4</td>
</tr>
</table>
```

Figure **9.7.2**: Code to span table cells.

Although there are four `<td>` cell tags for each data row, note that there is a need for only two `<td>` heading tags in the first row. Since each `<td>` heading tag spans two cells (colspan="2"), a total of four cells, the full width of the table, is accounted for.

The rowspan Attribute

In the same way that you span columns, you can span rows across two or more contiguous cells by using the **rowspan**="*n*" attribute as shown in Figure 8-39.

Heading	Cell 1.2	Cell 1.3	Cell 1.4
	Cell 2.2	Cell 2.3	Cell 2.4

Figure **9.7.3**: Table with spanned rows.

```
<style type="text/css">
  table         {border:outset 1px}
  table td      {border:inset 1px; padding:3px}
                 text-align:center;
                 background-color:#F0F0F0}
</style>

<table>
<tr>
  <td id="HEAD"   rowspan="2"  >Heading</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
  <td>Cell 1.4</td>
</tr>
<tr>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
  <td>Cell 2.4</td>
</tr>
</table>
```

Figure **9.7.4**: Code to span table rows.

Note that the first cell in the first row spans two rows. As a result, there is one less cell to specify in the second row. The first row has *four* `<td>` tags. The second row needs only three `<td>` tags since the first cell in that row is encompassed by the previously spanned row.

Spanning Rows and Columns - TOP

You can span cells across rows and columns to produce interesting and useful table structures. The coding is not that complex if you follow closely across each row, and determine whether you need to code a `<td>` tag for each cell position or whether you

can leave out the `<td>` tag because the cell has already been spanned. In the following code, the `<td>` tags that do not have to be coded are displayed in brackets [ ]. These cells are accounted for by other cells that have spanned across their table positions.

<table>
<tr><td colspan="2">Cell 1.1</td><td>Cell 1.3</td><td rowspan="2">Cell 1.4</td></tr>
<tr><td>Cell 2.1</td><td rowspan="2">Cell 2.2</td><td>Cell 2.3</td></tr>
<tr><td>Cell 3.1</td><td colspan="2">Cell 3.3</td></tr>
</table>

Figure **9.7.5**: Table with spanned rows and columns.

```
<style type="text/css">
  table    {border:outset 1px}
  table td {border:inset 1px; padding:3px}
</style>

<table>
<tr>
  <td colspan="2">Cell 1.1</td>
  [<td>Cell 1.2</td>]
  <td>Cell 1.3</td>
  <td rowspan="2">Cell 1.4</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td rowspan="2">Cell 2.2</td>
  <td>Cell 2.3</td>
  [<td>Cell 2.4</td>]
</tr>
<tr>
  <td>Cell 3.1</td>
  [<td>Cell 3.2</td>]
  <td colspan="2">Cell 3.3</td>
  [<td>Cell 3.4</td>]
</tr>
</table>
```

Figure **9.7.6**: Code to span table rows and columns.

Listing 8-35.

A Spanning Example

The output below is an example of using row and column spanning to produce a complex table structure.

Quantity and Value of Installed Software

Software	Department A		Department B	
	Copies	$ Value	Copies	$ Value
Word Processors	10	700.00	15	1,050.00
Spreadsheets	12	780.00	18	1,170.00
Databases	7	595.00	9	765.00
Total Value	$ 2,075.00		$ 2,985.00	

Figure **9.7.8**: Complex table with spanned rows and columns.

Determining Cell Spanning

Before you begin coding, it is a good idea to visualize the row and column spanning that needs to take place. Figure **9.7.9** shows the table with vertical arrows indicating row spanning, and horizontal arrows indicating column spanning. Once this structure is visualized, it becomes relatively easy to produce the code to describe the table.

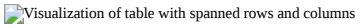


Figure **9.7.9**: Visualization of table with spanned rows and columns.

Coding the Table Structure

The following code begins the table description with a caption followed by the coding for the first row of the table. At this point, setting up the style sheet is not a consideration. Focus needs to be on the structure of the table and required column and row spanning.

```
<table>
<caption>Quantity and Value of Installed Software</caption>
<tr>
  <td rowspan="2">Software</td>
  <td colspan="2">Department A</td>
  <td colspan="2">Department B</td>
</tr>
```

Figure **9.7.10**: Code for column header spanning.

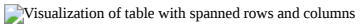
Beginning at the top-left of the table in Figure (\PageIndex{9}\), notice that the first cell spans two rows. Thus, this top-left cell is coded with rowspan="2" to combine it with the cell beneath to produce a single, combined cell. Furthermore, note that this cell contains no content, only the pair of `<td></td>` tags. By leaving the cell empty, cell borders are not displayed.

Moving from left to right across this first row, notice that the next cell, containing Department A, is not a single cell. It spans two horizontal cells. Therefore, the `<td>` tag contains colspan="2" to combine this cell with the one in the next column.

Moving to the right, you come to the cell containing the heading Department B. Again, this is not a single cell, but a spanned cell that crosses two columns. This `<td>` tag, like the previous one, contains the colspan="2" attribute.

You have now reached the end of the first row of the table. It is important to realize that, although you have coded only three cell descriptions, you have accounted for a total of five cells: the total number of columns in the full table. Always keep in mind this fact that you must describe as many rows and as many columns as there are in the full dimensions of the table.

Continuing with the coding for the second row of the table, this row contains the Copies and $ Value subheadings for their respective columns.

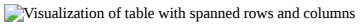



```
<tr>
  <td>Copies</td>
  <td>$ Value</td>
  <td>Copies</td>
  <td>$ Value</td>
</tr>
```

Figure 9.7.11: Code for second row of table.

Again, begin at the left of the table and move to the right across the row. The first thing to notice is that the first cell in this row does not need to be coded. It has already been accounted for by the row spanning that took place for the cell above. Now move to the right to the next cell along the row. The next four cells are, in fact, regular non-spanned cells that are each described by a `<td>` tag. You come to the end of the second row having coded four cells, but still accounting for a total of five columns.

Coding for the third through fifth row of the table is added below.




```
<tr>
  <td>Word Processors</td>
  <td>10</td>
  <td>700.00</td>
  <td>15</td>
  <td>1,050.00</td>
</tr>
<tr>
  <td>Word Processors</td>
  <td>12</td>
  <td>780.00</td>
  <td>18</td>
  <td>1,170.00</td>
</tr>
<tr>
  <td>Word Processors</td>
  <td>7</td>
  <td>595.00</td>
  <td>9</td>
  <td>765.00</td>
</tr>
```

Figure 9.7.12: Code for third through fifth row of table.

Here, there are five separate cells containing a row heading and the main information content of the table. There are no spanned cells which means that there are five `<td>` tags across the row. This same coding format can be followed for the next two rows

of the table.

The final row of the table is a summary line presenting totals of the dollar amounts in the body of the table. This row is coded below.

Visualization of table with spanned rows and columns

```
<tr>
  <td>Total Value</td>
  <td colspan="2">$ 2,075.00</td>
  <td colspan="2">$ 2,985.00</td>
</tr>
</table>
```

Figure **9.7.13**: Code for final row of table.

Listing 8-39. Code for final row of table.

Again, trace across the row beginning with the first cell. This is a standard cell containing the text Total Value. Next, you come to a cell that is combined with the following cell and requiring the colspan="2" attribute. Finally, you encounter another cell that is combined with its following cell. You have coded three cell specification, but again have accounted for a total of five cells across the row.

Viewing the Table Structure

When the above coding is completed, the overall structure of the table and the cell contents are finished. This unstyled table is shown in Figure 8-43. Borders have been added to help visualize the cells. The issue at this point is whether you have correctly produced the table structure, ignoring for the moment any styling of the cells.

Quantity and Value of Installed Software

Software	Department A		Department B	
	Copies	$ Value	Copies	$ Value
Word Processors	10	700.00	15	1,050.00
Spreadsheets	12	780.00	18	1,170.00
Databases	7	595.00	9	765.00
Total Value	$ 2,075.00		$ 2,985.00	

Figure **9.7.14**: Basic coding of table with spanned rows and columns.

Styling the Table

Now, it is a matter of adding a style sheet to decorate the table. In this example, shown in Listing 8-40, style classes are used to assign header, row, and footer styling. Class .HEAD is applied to the column headings, setting bold and centered text with a background color. Class .ROW formats each data row with right-aligned text, and class .LABEL overrides this row styling for the first cell in the row by aligning text labels to the left. Class .TOTAL formats the footer row with bold and right-aligned text with a background color.

```
<style type="text/css">
  table         {border:outset 1px}
  table td      {border:inset 1px; padding:5px}
  table caption {font:bold 12pt}
  .HEAD         {font-weight:bold; text-align:center; vertical-align:middle;
                 background-color:#F0F0F0}
  .ROW          {text-align:right}
  .LABEL        {text-align:left}
  .TOTAL        {text-align:right; font-weight:bold;
                 background-color:#F0F0F0}
</style>

<table>
<caption>Quantity and Value of Installed Software</caption>
<tr   class="HEAD"  >
  <td rowspan="2">Software</td>
  <td colspan="2">Department A</td>
  <td colspan="2">Department B</td>
</tr>
<tr   class="HEAD"  >
  <td>Copies</td>
  <td>$ Value</td>
  <td>Copies</td>
  <td>$ Value</td>
</tr>
<tr   class="ROW"  >
  <td   class="LABEL"  >Word Processors</td>
  <td>10</td>
  <td>700.00</td>
  <td>15</td>
  <td>1,050.00</td>
</tr>
<tr   class="ROW"  >
  <td   class="LABEL"  >Databases</td>
  <td>7</td>
  <td>595.00</td>
  <td>9</td>
  <td>765.00</td>
</tr>
<tr   class="TOTAL"  >
  <td>Total Value</td>
  <td colspan="2">$ 2,075.00</td>
  <td colspan="2">$ 2,985.00</td>
</tr>
```

Figure **9.7.15**: Final code for table with style sheet attached.

What appeared to be a complex coding exercise turns out to be fairly easily managed by separating structure from styling. First, work systematically across the rows of the table, code all stand-alone cells and code cells that require spanning. Ignore cells that are encompassed by a previous span. Take a look at the row table, and make sure that its structure is correct. Finally, apply a style sheet to format those rows or cells that require styling.

9.7: Column and Row Spanning is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

9.8: Table Groupings

When working with large tables with varying cell sizes, alignments, and colors, you can bring a more orderly structure to the table by *grouping* rows. In this fashion, you can set structural and styling specifications for the groups, and have those features propagate automatically across the individual cells contained within the groups.

The table in Figure **9.8.1** makes use of the special table tags `<thead>`, `<tbody>`, and `<tfoot>` to classify and organize the rows of the table.

Sales Order

Your Name Your Address City, ST 55555				
No.	Description	Quantity	Price	Amount
11111	Microsoft Windows XP Professional	2	169.99	339.98
22222	Microsoft Office XP Professional	2	449.99	999.98
33333	Adobe Photoshop 7.0	1	579.95	579.95
44444	HP PhotoSmart 7550 Printer	1	299.99	299.99
55555	USB Device Cable	1	5.49	5.49
			Shipping	50.00
			Sales Tax	155.77
			Total	$ 2,431.16

Figure **9.8.1**: Table with row groupings.

It is instructive to view the general layout of this table and its component parts in outline form. This view is produced below, and identifies the various grouped parts of the table as well as the major tags used to structure it.

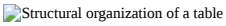


Figure **9.8.2**: Structural organization of a table.

There are three main structural parts to a table: the "table head," "table body," and "table foot," which are set apart within associated `<thead>`, `<tbody>`, and `<tfoot>` tags. The outline XHTML coding for these parts is shown below.

```
<table>
<caption>Sales Order</caption>

<thead>
  XHTML for table head
</thead>

<tfoot>
  XHTML for table foot
</tfoot>

<tbody>
  XHTML for table body
</tbody>

</table>
```

Figure 9.8.3: Code outline of head, body, and foot of a table.

When arranging the code for these sections, both the `<thead>` and `<tfoot>` tags must appear *before* the `<tbody>` tag.

The `<thead>` Tag

The `<thead>` section contains information that appears one time at the beginning of the table. It is coded very much like a normal set of headings. In this example, there are two rows comprising the heading. The first row contains name and address information that is placed in a single cell that spans the width of the table. The second row is a set of column headings.

```
<thead>
<tr>
  <td colspan="5">
    Your Name<br/>
    Your Address<br/>
    City, ST 55555<br/>
  </td>
</tr>
<tr>
  <td>No.</td>
  <td>Description</td>
  <td>Quantity</td>
  <td>Price</td>
  <td>Amount</td>
</tr>
</thead>
```

Figure 9.8.4: Code for `<thead>` section of table.

The <tfoot> Tag

The table footer appears one time at the bottom of a table. It is identified with a <tfoot> tag. The present table contains three rows in the footer. All rows have a spanned label area and a single cell for data.

```
<tfoot>
<tr>
  <td colspan="4">Shipping</td>
  <td>50.00</td>
</tr>
<tr>
  <td colspan="4">Sales Tax</td>
  <td>155.77</td>
</tr>
<tr>
  <td colspan="4">Total</td>
  <td>$ 2,431.16</td>
</tr>
</tfoot>
```

Figure 9.8.5: Code for <tfoot> section of table.

The <tbody> Tag

The table body displays the main content of the table. It appears in the <tbody> section, and contains as many rows as there are data items to report. In the present table, data appear in five separate cells across the row.

```
<tbody>
<tr>
  <td>11111</td>
  <td>Microsoft Windows XP Professional</td>
  <td>2</td>
  <td>169.99</td>
  <td>339.98</td>
</tr>
<tr>
  <td>22222</td>
  <td>Microsoft Office XP Professional</td>
  <td>2</td>
  <td>449.99</td>
  <td>999.98</td>
</tr>
<tr>
  <td>33333</td>
  <td>Adobe Photoshop 7.0</td>
  <td>1</td>
  <td>579.95</td>
  <td>579.95</td>
</tr>
<tr>
  <td>44444</td>
  <td>HP PhotoSmart 7550 Printer</td>
  <td>1</td>
  <td>299.99</td>
  <td>299.99</td>
</tr>
<tr>
  <td>55555</td>
  <td>USB Device Cable</td>
  <td>1</td>
  <td>5.49</td>
  <td>5.49</td>
</tr>
</tbody>
```

Figure **9.8.6**: Code for `<tbody>` section of table.

After coding the three sections of the table, you might wish to view the output to make sure the table is properly structured before applying styles to it. Figure **9.8.6** shows the current structure of the example table with borders added to make the cells visible.

Sales Order

Your Name
Your Address
City, ST 55555

No.	Description	Quantity	Price	Amount
11111	Microsoft Windows XP Professional	2	169.99	339.98
22222	Microsoft Office XP Professional	2	449.99	999.98
33333	Adobe Photoshop 7.0	1	579.95	579.95
44444	HP PhotoSmart 7550 Printer	1	299.99	299.99
55555	USB Device Cable	1	5.49	5.49
	Shipping			50.00
	Sales Tax			155.77
	Total			$ 2,431.16

Figure 9.8.7: Organization of unstyled table.

Applying Header Styles

Once the column groups and individual columns are styled, additional special styling may be needed for the `<thead>` section of a table. This is the case in the current table where the address lines (the cell in the row with id="ADDR") should be aligned left, and the column headings (the cells in the row with id="HEAD") should be centered, bold, and with background and text colors. This additional styling is added to the style sheet in Listing 8-48 and applied in Figure 9.8.9.

```
<style type="text/css">
  table                          {border:outset 1px; border-collapse:collapse}
  table caption                  {font:bold 14pt; color:#707070}
  table td                       {border:inset 1px; padding:3px}

  table tr#ADDR td               {text-align:left; background-color:#FFFFFF}
  table tr#HEAD td               {font-weight:bold; text-align:center;
                                  background-color:#A0A0A0; color:#FFFFFF}
</style>
```

Figure 9.8.8: Additional style sheet entries for `<thead>` styling of example table.

Sales Order

Your Name
Your Address
City, ST 55555

No.	Description	Quantity	Price	Amount
11111	Microsoft Windows XP Professional	2	169.99	339.98
22222	Microsoft Office XP Professional	2	449.99	999.98
33333	Adobe Photoshop 7.0	1	579.95	579.95
44444	HP PhotoSmart 7550 Printer	1	299.99	299.99
55555	USB Device Cable	1	5.49	5.49
			Shipping	50.00
			Sales Tax	155.77
			Total	$ 2,431.16

Figure 9.8.9: Table with `<thead>` styling applied.

Applying Footer Styles

The table footer is styled in the same general fashion as the header. In this case, the "Shipping," "Sales Tax," and "Total" labels should be right aligned, and the total line should display in bold with background and text colors. These additions are made in Listing 9.8.10 and shown in Figure 9.8.11.

```
<style type="text/css">
  table                          {border:outset 1px; border-collapse:collapse}
  table caption                  {font:bold 14pt; color:#707070}
  table td                       {border:inset 1px; padding:3px}



  table tr#ADDR td               {text-align:left; background-color:#FFFFFF}
  table tr#HEAD td               {font-weight:bold; text-align:center;
                                  background-color:#A0A0A0; color:#FFFFFF}

  table tfoot td                 {text-align:right}
  table tfoot tr#TOTAL           {font-weight:bold;
                                  background-color:#A0A0A0; color:#FFFFFF}
</style>
```

Figure **9.8.10**: Additional style sheet entries for `<tfoot>` styling of example table.

All cells in the footer section (table tfoot td) should be right aligned. The footer row identified by id="TOTAL" (table tfoot tr#TOTAL) should be bold with background and text colors. Again, if all rows and cells in the `<tfoot>` section are styled the same, then that styling can be applied through the single tfoot selector.

Sales Order

Your Name
Your Address
City, ST 55555

No.	Description	Quantity	Price	Amount
11111	Microsoft Windows XP Professional	2	169.99	339.98
22222	Microsoft Office XP Professional	2	449.99	999.98
33333	Adobe Photoshop 7.0	1	579.95	579.95
44444	HP PhotoSmart 7550 Printer	1	299.99	299.99
55555	USB Device Cable	1	5.49	5.49
			Shipping	50.00
			Sales Tax	155.77
			Total	$ 2,431.16

Figure **9.8.11**: Table with `<tfoot>` styling applied.

Note that when you use table row groups, the `<thead>` and `<tfoot>` sections must be coded before the `<tbody>` section to pass W3C XHTML 1.1 validation.

9.8: Table Groupings is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

10: Playing Multimedia

10.1: Audio and Video Formats

10.2: Playing Audio

10.3: Playing Video

10.4: Third Party Content

10: Playing Multimedia is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

10.1: Audio and Video Formats

One of the exciting results of the movement to broadband Internet connectivity is the growing reliance on sound and movies for delivering Web page content. High-speed network connections are capable of transmitting data at several megabytes per second which is a crucial need for delivering large audio and video files to the desktop.

Transmission Methods

There are two ways to receive audio and video files at your browser. One method is by **downloading** the file from a Web server. In this case, there are delays while the file is being transmitted to your browser. Despite the delay in downloading, modern browsers can begin playing the media before it all arrives at your desktop through *progressive download* methods. That is, playback takes place at the same time the file is being downloaded. After the file is downloaded, it is available on your PC for rapid reloading and replaying. Downloaded files can be saved to your desktop and thereafter accessed and played locally.

The second way of receiving multimedia content is through **streaming** methods. If audio and video files are hosted on a "media server" then that machine can transmit files as a continuous stream. It is not necessary to wait for a download to finish before playback begins. Usually there are a few seconds of delay while initial content is transmitted to the PC; thereafter, the browser receives a continuous stream of sound and pictures. While the transmitted portion of the file is being played, the next portion of the stream is being transmitted to and **buffered** (temporarily saved) on the PC to await playback. This *interleaving* of transmission and playback provides a continuous stream of multimedia content. Often, media files can be streamed at a rate that matches the bandwidth at which you are connected to the Internet. Streamed files are not downloaded; therefore, you do not receive a copy of the file.

Streaming methods are also used for live broadcasts. There are a few seconds of delay for initial transmissions to be received and buffered at the PC. While playback of the buffered content continues, the next portion of content is transmitted to and buffered on the PC to await playback. This ability of your computer to play buffered content at the same time that it receives and buffers transmissions produces a continuous stream of audio and video output.

Media Playback

In order to receive audio and video files, live media streams, and other types of media files, you must have the appropriate plug-in or helper application installed on your computer. When the browser loads a Web page that contains multimedia content, it will search for the appropriate plug-in configured to display the file type. If it cannot find the plug-in on the user's computer, the browser will prompt the user to download the file. The most common types of browser plug-ins include:

- Windows Media Player - Displays Windows Media Player for streaming audio and video
- Apple Quicktime Player - Displays the Apple Quicktime Player for audio and video
- RealPlayer - Displays the RealPlayer for streaming audio and video
- Adobe Flash Player - Displays .swf format files. These can contain audio, video, and iteractivity
- Adobe Shockwave Player - Displays high-performance media created using Adobe Director
- Adobe Reader - Used to Display documents in .pdf format
- Java Runtime Environment - Used to run applications utilizing Java technology.

The player or application needed depends on the formats in which media content is provided.

The following sections include use of Windows Media player; however, Apple QuickTime or RealPlayer can also be used. You should have the latest version installed. When using Windows Media Player, there are two ways to provide Web access to media files. You can create links to the files or you can embed the media within a Web page.

Linking to Media

You can link to media files just like you link to Web pages, with an `<a>` tag. In this case, the href attribute gives the URL to the file that is located either locally on the PC or remotely on a Web server. When linking to a media file, either a Media Player opens externally and plays the media file or a separate browser tab opens and plays the file.

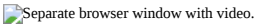


Figure **10.1.1**: A separate browser window opens and plays a media file.

Embedding Media

The Media Player can be embedded on the Web page rather than opening in a separate window. In this case, the player is an element of the page, permitting close integration of sounds and pictures with other content on the page. When the player is embedded, the media plays only while viewing the page and stops when another page is accessed.



Figure **10.1.2**: Embedding a video on a Web page.

The way in which access is provided to media files depends to a large extent on the capabilities of your Web clients. For browsers with slower Internet connections, presenting links to media files along with relevant information about file sizes gives the user the choice about whether to proceed with a download. An embedded player set for automatic playback could cause lengthy delays and an unhappy experience for the user.

For Web clients with broadband connections, the choice has more to do with page design considerations. If you anticipate continued browsing of your site with background or external playback, then you can provide links to media files. On the other hand, if playback is integral to other content on your page, then you will likely embed the media, and start playback when the page is opened.

Audio Formats

Several different sound formats are used for Web playback. They are characterized by the brands of computer systems they are designed to serve as well as by the sound quality that is provided. Sound quality depends on the type of sound compression technique used to digitize the file. This compression method is encapsulated within a sound **codex** (compression / decompression

algorithm) that reduces file sizes by removing unnecessary signals or collapsing redundant or repeating signals. The most popular sound formats are briefly described in the following table.

Format	Description
AIFF	Audio Interchange File Format. Developed for use on Apple Macintosh computers. It is also supported on Windows platforms using the extension **.aif**
AU	An older type of sound file that generally has poorer sound quality than new types of audio formats. Developed for use primarily on UNIX workstations. Provides mono playback.
MIDI	Musical Digital Interface. Limited to playback of sounds generated by music synthesizers.
MP3	Uses advanced compression algorithm that results in smaller file sizes. Podcasts typically use this format.
RA	RealAudio. Proprietary format for streamed audio in real time. Produces small file sizes for lower bandwidth connections. Requires special player.
WAV	Developed for Windows computers. One of the most common formats for download files.
WMA	Windows Media Audio. Developed for streamed audio files. Very good quality for a variety of bandwidths. One of the most common formats.
M4A	Audio only MPEG-4 format supported by Quicktime, iTunes, and iPods

Figure **10.1.3**: Sound file formats.

Video Formats

As with audio formats, video file formats are characterized by the brands of computer systems they are designed to serve, and by the video quality that is provided. Video quality also depends on the type of codex used to digitize the file. The most popular video formats are summarized in the following table.

Format	Description
AVI	Audio Video Interleave. Standard video format for Windows. Common for download files.
MPEG	Moving Pictures Expert Group. High compression and small file sizes for both audio and video playback.
RV	Real Video. Proprietary format for streamed video in real time. Produces small file sizes for low bandwidth connections.
MOV (QuickTime)	Developed for Apple Macintosh computers. Files have .mov extension. One of the most common formats for Web downloads. Requires special player.
WMV	Windows Media Video. Developed for streamed video files. Very good quality for a variety of bandwidths. One of the most common formats.
FLV (Flash Video File)	Flash compatible format exported by the Adobe Flash Video Exporter plug-in. This format is commonly used on YouTube.
M4V, MP4 (MPEG-4)	Supported by Quicktime, iTunes, and iPods.

Figure **10.1.4**: Video file formats.

10.1: Audio and Video Formats is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

10.2: Playing Audio

Audio files can be downloaded from the Web for local playback. Audio files in MPEG and WAV formats are the most common types for downloads. Audio files can also be streamed from a media server computer. These can be in the form of live broadcasts or digitized files for on-demand retrieval. WMA format is a popular Windows format for streamed audio.

Linking to Audio Files

An audio file is made available for download and playback by placing it in your Web directory. Access to the file is through a simple link using an `<a>` tag coded with the relative URL to the file. When the link is clicked, the file is opened on the client's desktop in the external Media Player. The player provides progressive download so that the user does not have to wait for the entire file to download before playback begins. After the file is downloaded, it becomes available on the uers's PC and can be replayed locally.

The following graphic and text links connect to a WAV audio file appearing in the same directory as this Web page. A click on a link opens the external Media Player for file download and playback. The exact appearance of the Media Player on your desktop depends on how it is configured.



PeopleWillCome.wav (426 KB)

Figure **10.2.1**: Links to play an audio file.

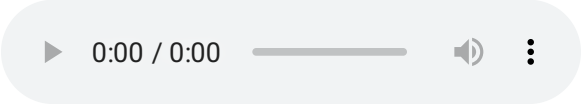
The above links are coded as shown below. The href attribute of the `<a>` tags surrounding both the graphic image and text are URLs for the local WAV audio file.

```
<a href="PeopleWillCome.wav"><img src="PeopleWillCome.gif" alt="Kansas ballpark"/></a>
<a href="PeopleWillCome.wav">PeopleWillCome.wav (426 KB)</a>
```

Figure **10.2.2**: Code to link to audio file.

The `<audio>` Element

HTML5 introduced a new `<audio>` tag for standardizing the way audio files can be played in a browser. Codecs for audio can still differ between different system types so for fallback functionality you may need to supply multiple sources to an **audio** element. The browser will try to load the sources in order and if it cannot use a `<source` it will display the error message.




```
<audio controls>
  <source src="PeopleWillCome.wav" />
  <source src="PeopleWillCome.mp3" />
 Your browser does not support the audio tag.
</audio>
```

Figure **10.2.3**: Code to size and pad a paragraph.

Embedding Audio Files - Deprecated

Embedding the audio player on a Web page is accomplished with an `<object>` tag. This tag is a generalized way of embedding numerous kinds of objects within a Web page. Its general format for embedding audio files is shown below.

```
<object data="url" classid="classID">
  <param name="url" value="url"/>
  <param name="autostart" value="true|false"/>
</object>
```

Figure 10.2.4: General format for `<object>` tag.

The `<object>` tag is an in-line tag which means that it must be enclosed inside a block-level tag to pass XHTML 1.1 validation.

The following are common attributes of the `<object>` tag:

data Required: attribute provides the name of the file to be played.

classid attribute is a number that identifies the type of object to embed in the page. The classid identifies an ActiveX control that must be installed on the user's PC. If the ActiveX control is not installed, the browser can automatically download and install it. Values for common object players are listed below:

- Windows Media Player - classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6"
- QuickTime Player - classid="CLSID:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
- RealPlayer - classid="CLSID:CFCDAA03-8BE4-11cf-B84B-0020AFBBCCFA"
- Flash Shockwave Player - classid="CLSID:D27CDB6E-AE6D-11cf-96B8-44455350000"

codebase Specifies a relative path for the location and download of the plug-in if needed. Values for common object players are listed below:

- Windows Media Player - codebase="http://activex.microsoft.com/activex...ion=6,0,02,902"
- QuickTime Player - codebase="http://www.apple.com/qtactivex/qtplugin.cab
- Flash Shockwave Player - codebase="http://download.macromedia.com/pub/s...rsion=8,0,22,0"

type Optional; Specifies a valid MIME type such as audio/wav of the audio or video file.

title Optional; Provides a brief text description displayed by browsers or assistive technology applications.

In its default settings, the Player displays both a video screen and playback controls. The screen is used to display **visualizations** which are graphical displays that change in response to the audio signal. The size of the Player can be changed from its default size by coding CSS **width** and **height** properties, and by giving the dimensions in pixels.

Embedding the Media Player for an audio presentation is shown in Figure 9-6 with its code given in Listing 9-3. The URL points to an audio file in the same directory containing this Web page. The Player is displayed in its default size with its visualization screen, and playback does not commence until the "Play" button is clicked. Please note that Figure 9-6 may only display in Internet Explorer.

```
<object data="PeopleWillCome.wav" classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
  <param name="url" value="PeopleWillCome.wav"/>
  <param name="autostart" value="false"/>
</object>
```

Figure 10.2.5: Code to embed the Media Player to play an audio file.

The `<object>` tag has a selection of startup param (parameter) settings for the Player. You must code the name="url" setting and specify the value="url" location of the audio file. Whether or not the file begins playback immediately after the page is loaded is given in the name="autostart" parameter. The default setting is value="true"; setting value="false" requires the user to initiate playback by clicking the "Play" button on the controls.

Embedding Audio Files - Other Browsers - Deprecated

Internet Explorer requires that the classid attribute (to indicate the player's ActiveX control) in order to properly render the `<object>` tag. However, other browsers such as Firefox and Chrome require the `<object>` tag without the classid. To resolve this issue, conditional comments can be used to direct non-IE browsers to the standard `<object>` tag. A sample code block is shown below:

```
<div>
<object data="PeopleWillCome.wav" classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
  <param name="url" value="PeopleWillCome.wav"/>
  <param name="autostart" value="False"/>

<!--[if !IE]>-->

<object data="PeopleWillCome.wav" style="width:100px;height:50px">
  <param name="url" value="PeopleWillCome.wav"/>
  <param name="autostart" value="False"/>

 </object>

<!--<![endif]-->

</object>

</div>
```

Figure **10.2.6**: Code to embed in IE browser.

The conditional comments begin with `<!--[if !IE]>-->` and end with `<!--<![endif]-->` . These statements are only processed by Internet Explorer. In this case, the comments indicate that Internet Explorer should ignore the code contained between the opening and closing comment tags. IE will render the first `<object>` tag and ignore the second. Other browsers will render the first `<object>` tag and then the second `<object>` tag since browsers render in a top-down line-by-line fashion. The second `<object>` tag will replace the first object block and display correctly. The second `<object>` (Non-IE browsers) must also contain height and width values in order to render properly in Firefox. The CSS height and width properties can be applied as shown in the example above.

The following are `<param />` media attribute values that can be used with the `<object>` tag:

- src - Required; provides the name of the file to be played
- loop - Optional; determines how many times the media file will repeat; Value may be a numeric value or true for continuous play
- hidden - Optional; Hides the media console. Values include false (default) and true
- autoplay - Optional; Determines if the media player will play automatically when the page loads. Values true or false (default)
- controller - Optional; indicates whether the media control console will display. Values true or false

Background Sounds - Deprecated

For certain pages, you may not wish to supply either a Player display or tags to control it. This would be the case in playing a background sound of music or narration that begins automatically when the page is loaded, and stops when a different page is accessed. To play a background sound, all that is required is a hidden player that auto-starts the audio file.

```
<object data="BkgnSound.wav" classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6"
    style="width:0px;height:0px">
    <param name="url" value="BkgndSound.wav"/>
    <param name="autostart" value="true"/>

    <object>

    <!--[if !IE]>-->

    <object data="BkgnSound.wav">
        <param name="url" value="BkgnSound.wav"/>
        <param name="autostart" value="true"/>

    </object>

<!--<![endif]-->

</object>
```

Figure **10.2.2**: Code to play a background sound.

Again, since the Player is not visible, it does not matter where this `<object>` tag is coded on the page. As soon as the page finishes loading, download and playback begins. When navigation moves to a different page, the audio stops.

10.2: Playing Audio is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

10.3: Playing Video

There are no significant differences between accessing and playing video files versus audio files. Video files can be downloaded from the Web for local playback. Files in AVI and MPG formats are the most common types for downloads. Video files in WMV format can be streamed from a media server. These can be live broadcasts or digitized files for on-demand retrieval.

Linking and Embedding Video Files

A video file is downloaded by creating an `<a>` link with the URL to the file. The file is downloaded and opened in the external Media Player in the same fashion as for audio files.

The following link connects to a WMV video file appearing in the same directory as this Web page. When the link is clicked, the file opens in the external Media Player. The exact manner in which the Media Player appears on your desktop depends on how it is configured.

Casablanca

Casablanca.wmv (484 KB)

Figure **10.3.1**: Links to play a video file.

```
<a href="casablanca.wmv"><img src="casablanca.gif" alt="Casablanca"/></a>
<a href="casablanca.wmv">Casablanca.wmv (484 KB)</a>
```

Figure **10.3.2**: Code to link to video file.

Video files can be embedded in a Web page by using the `<video>` tag in HTML5. The following example is from w3.org - HTML5 Video Events and API



Figure **10.3.3**: Using a video tag to play video files.

<video controls="" id="video" mediagroup="myVideoGroup" poster="http://media.w3.org/2010/05/sintel/poster.png" preload="none"> </video>

```
<video id='video' controls preload='none' mediagroup='myVideoGroup'
    poster="http://media.w3.org/2010/05/sintel/poster.png" >
    <source id='mp4' src="http://media.w3.org/2010/05/sintel/trailer.mp4" type='video/mp4'>
    <source id='webm' src="http://media.w3.org/2010/05/sintel/trailer.webm" type='video/webm'>
    <source id='ogv' src="http://media.w3.org/2010/05/sintel/trailer.ogv" type='video/ogg'>
    <p>Your user agent does not support the HTML5 Video element.</p>
</video>
```

Figure **10.3.4**: Code to use a video to play video files.

Accessing Video Streams

With availability of URL addresses for external sites, you can link to streamed video content. As with audio files, you may be able to grab the URL from the link if the video is playing through Windows Media Player. While the video is playing, right-click on the screen and choose "Properties" from the pop-up menu. From the Properties window, copy the full URL for the file being streamed.

The following embedded video is linked by this technique. In this case, the URL points to a movie trailer located through a Web search.

Figure **10.3.5**: Embedding a streamed video file.

```
ect id="PlayerX" classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6" style="width:400px; height:300px">
        <param name="url" value="http://www.youtube.com/v/VsvaFxkXrbg&hl=en_US&fs=1&"/>
        <param name="autostart" value="False"/>
        <!--[if !IE]>-->
        <object data="http://www.youtube.com/v/VsvaFxkXrbg&hl=en_US&fs=1&" id="PlayerX" style="width:400px; heigh
                <param name="url" value="http://www.youtube.com/v/VsvaFxkXrbg&hl=en_US&fs=1&"/>
                <param name="autostart" value="false"/>
        </object>
        <!--<![endif]-->
        <p> A sample Video Stream </p>
</object>
```

Figure **10.3.6**: Code to embed a streamed video file.

Many Web-based video files have costs or fees for viewing. Also, you will often encounter video players embedded in pages of the site with settings that do not permit access to the Properties window for extracting URLs. You will simply need to be patient and persistent to find video to which you can link. Moreover, URLs are changed routinely. Later you may not be able to link to the video at its original location.

Live video streams can be captured and played through the Media Player. The object tag can be further used for live video streams.

10.3: Playing Video is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

10.4: Third Party Content

In some cases, it is necessary to embed third party content such as external HTML pages or multimedia content into the current page. This can be accomplished using the `<iframe>` tag.

The `<iframe>` is used to specify an inline frame or inline window. Inline frames are often used in online advertising. Inline frames serve as a window within the current page to display other content such as a second HTML page. The second page contains the content that is embedded within the current page. The `<iframe>` is also commonly used to embed Youtube videos.

The `<iframe>` tag is coded as follows:

```
<iframe> </iframe>
```

Figure **10.4.1**: `<iframe>` tag

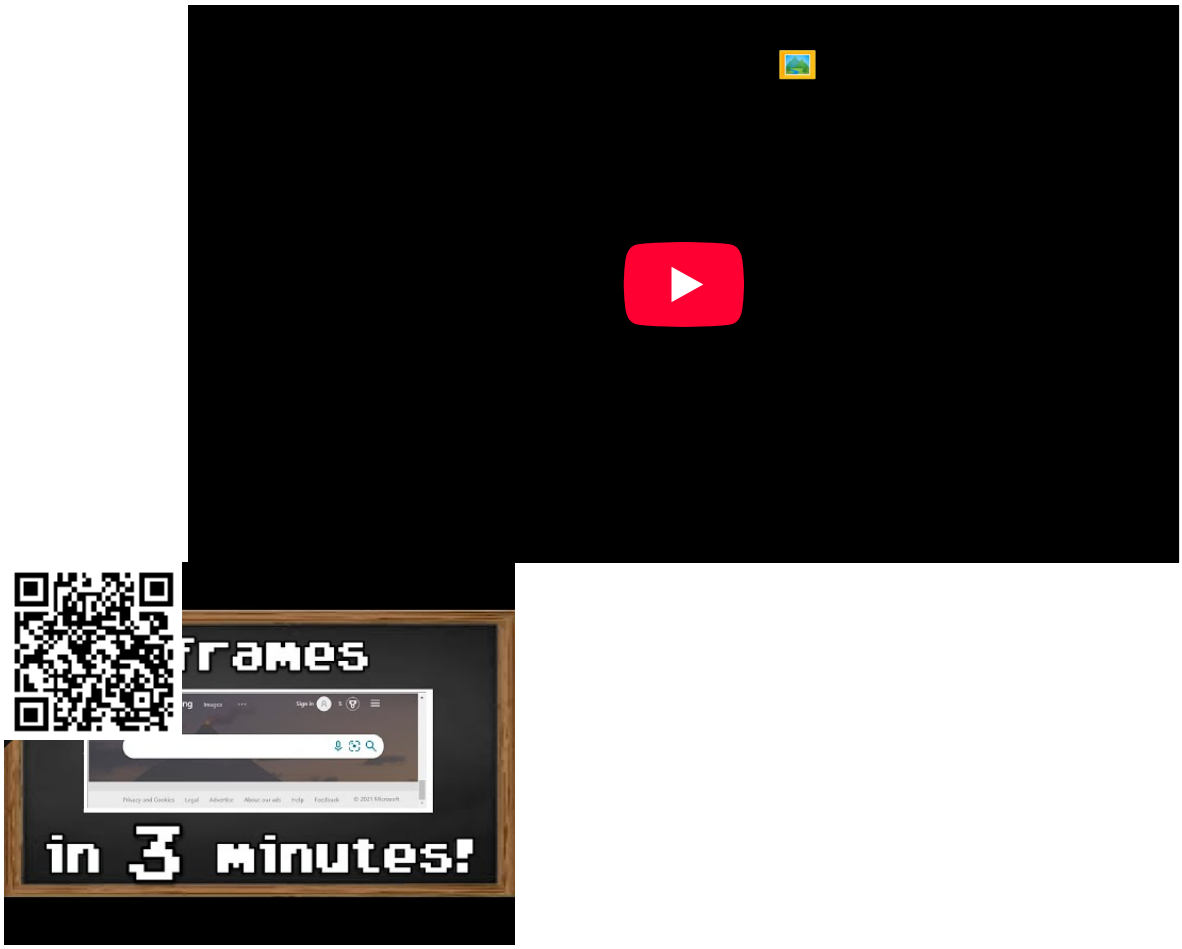
Note that a closing `</iframe>` tag is required even though it encloses nothing. You can additionally include any number of floating frames on a Web page.

The following table lists the attributes associated with the tag:

Attribute	Description
src="URL"	URL to the external content
height="pixels"	A pixel value. Specifies the height of the iframe window.
name="iframe name"	name associated with the iframe. This value is used by links to target the iframe window.
sandbox = "sandbox" - Applies all restrictions "allow-forms" - Re-enables form submission "allow-pointer-lock" - Re-enables APIs "allow-popups" - Re-enables popups "allow-same-origin" - Allows the iframe content to be treated as being from the same origin "allow-scripts" - Re-enables scripts "allow-top-navigation" - Allows the iframe content to navigate its top-level browsing context	Enables an extra set of restrictions for the content in an `<iframe>`
width="pixels"	A pixel value. Specifies the width of the iframe window

Figure **10.4.2**: `<iframe>` attributes.

The example below demonstrates how to embed a Youtube video using the iframe.




```
rame width="560" height="315" src="https://www.youtube.com/embed/aRGdDy18qfY" title="YouTube video play
meborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope;
ture-in-picture; web-share" allowfullscreen></iframe>
```

Figure 10.4.3: Examples of embedding a YouTube video.

The **src** attribute identifies the page to load initially into the frame. The **name** attribute assigns a name to the iframe as a target for links. You do not necessarily need to target the frame with links. You may simply wish to display a single external document inside the frame. In which case, you need only to specify a URL in the src attribute of the unnamed frame.

Both the width and height attributes can be used to set the dimension of the iframe. These dimensions can also be set using CSS.

HTML 5 includes the new sandbox attribute that can be used with the `<iframe>` tag. The sandbox attribute is used to restrict content that can appear in the iframe window. The sandbox is a useful security feature. When it is included with the `<iframe>` tag, it tells the the browser to disallow features that might cause a security risk to the site and its users. When the attribute is present, it will do as follows:

- treat the content as being from a unique origin
- block form submission
- block script execution
- disable APIs
- prevent links from targeting other browsing contexts
- prevent content from using plugins (through `<embed>`, `<object>`, `<applet>`, or other)
- prevent the content to navigate its top-level browsing context
- block automatically triggered features (such as automatically playing a video or automatically focusing a form control)

The iframe can also be used to display a single document or it can be a target for multiple linked documents. For example, click on the following links to display different pictures in the inline frame.

- Artemis
- Colossus
- Gardens
- Halicarnassus
- Lighthouse
- Pyramid
- Zeus



Figure **10.4.4**: An interactive `<iframe>`

The HTML code used to create the iframe is shown below:

```
<iframe name="TheFrame">  </iframe>

<div>
    <a href="Artemis.gif" target="TheFrame"> Artemis </a>
    <a href="Colossus.gif" target="TheFrame"> Colossus </a>
    <a href="Gardens.gif" target="TheFrame"> Gardens </a>
    <a href="Halicarnassus.gif" target="TheFrame"> Halicarnassus </a>
    <a href="Lighthouse.gif" target="TheFrame"> Lighthouse </a>
    <a href="Pyramid.gif" target="TheFrame"> Pyramid </a>
    <a href="Zeus.gif" target="TheFrame">  Zeus </a>
</div>
```

Figure **10.4.5**: Code for an interactive `<iframe>` object.

10.4: Third Party Content is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

11: Creating Forms

11.1: Form Controls
11.2: Creating Forms
11.3: Textbox Controls
11.4: Textarea Control
11.5: Radio Buttons
11.6: Checkboxes
11.7: Drop-Down Lists
11.8: Submit and Reset Buttons
11.9: Other Buttons
11.10: Grouping and Labels
11.11: Submitting Forms
11.12: Forms and CSS

11: Creating Forms is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.1: Form Controls

An HTML **form** provides a way for users to interact with a Web page. A form is, basically, a data capture device. It presents the user with one or more data input or selection **controls**, or **fields**, through which the user submits information to a Web page. On the basis of this submitted information, the page can respond to the user. This response can vary depending on the purpose of the form. The submitted data may be used,

- to direct visitors to a different page, much like what happens when clicking a link;
- to present visitors with personalized pages containing information and links pertinent to their interests or preferences;
- to trigger a complex search process to locate information, products, or services about which the user is interested;
- to perform personalized displays of options or calculations of prices of those products or services;
- to trigger a credit card check during purchase of products or services;
- to interface with the organization's accounting and billing systems to finalize purchase;
- to generate automated email responses as purchase confirmations;

and the list can go on and on. The point is that forms are the triggers for a whole host of activities that transform Web sites from simple electronic "page turners" into full-featured information processing systems.

Form Controls

Forms gather information from users by displaying special form controls that permit the user to enter data or make selections. The variety of standard controls that can be coded on a Web form are shown below.

Control Type	Browser Display
Textbox:	Entered information
Password:	•••••
Textarea:	
Radio Button:	Radio Button
Checkbox:	Checkbox
Drop-down List:	Item 1
Submit Button:	Submit
Submit Graphic:	Submit
Reset Button:	Reset

Figure **11.1.1**: Varieties of form controls.

Note that there are three basic categories of controls. The fields labeled "Textbox," "Password," and "Textarea" present input areas where the user can enter information as typed characters. The controls labeled "Radio Button," "Checkbox," and "Drop-down List"

provide options for the user to select from displayed items. The controls labeled "Submit Button," "Submit Graphic," and "Reset Button" are clickable controls for submitting the form data for processing or refreshing the entire form by clearing entries and selections.

HTML 5 introduces several new interactive form controls. Many of the the form controls include built-in validation. Currently, not all browsers support these controls. If a control is not supported, it will display as a standard input control with no dynamic functionality. Below are some of the most common and widely supported HTML 5 form controls.

Control Type	Browser Display
Email Address Input:	myemail@email.com
URL Input:	http://www.google.com
Search Field Input:	Search Text
Datalist	
Calendar:	mm/dd/yyyy **Note:** type="date" is supported only in Chrome, Safari, and Opera.
Color Picker:	
Range/Slider Input:	(Range 1-110)
Number:	(Enter number between 0 and 10)
Telephone:	**Note:** type="tel" is supported only in Safari 8.

Figure **11.1.2**: Commonly supported HTML 5 form controls.

Form Processing

A form by itself cannot provide much in the way of processing power. Therefore, HTML forms are backed by processing routines run remotely at the Web server to handle data submitted from these forms. Form information arriving at the Web server is processed by programs, or **scripts**, written in server languages such as ASP.NET, PHP, or Java. These are full-featured programming languages that often interact with databases and files on the server, thus offering a full range of information processing capabilities.

Although a large part of form processing takes place on the Web server, certain kinds of processing can take place locally at the browser. These processing scripts are written in the JavaScript language, the default scripting language included with modern browsers. This language has many of the features of regular programming languages. Its processing capabilities, though, are limited to what can be achieved on the local desktop PC.

In this tutorial, focus is only on setting up forms on a Web page -- how to create the input, selection, and submission controls. Server and browser processing is beyond the scope of interest.

11.1: Form Controls is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.2: Creating Forms

An HTML form consists of at least 1 of the form controls listed in the previous section. The form can appear anywhere within the body of a Web page. In fact, the entire page can be a form.

The `<form>` Tag

The form controls must be surrounded by a `<form>` tag, the general format for which is shown below.

```
<form
  id="formId (optional)"
  name="formName (optional)"
  action="url (required)"
  method="get (default)|post"
  autocomplete="on (default)|off"

>
  ...form controls
</form>
```

Figure **11.2.1**: General format for `<form>` tag.

All controls that are part of a form must be enclosed inside a `<form>` tag. The tag can appear anywhere on the page as long as it encloses all its controls. If the entire page consists only of form controls, you can code the opening tag immediately following the `<body>` tag and the closing tag immediately preceding the `</body>` tag to encompass the entire page as a form. Then, form controls can appear anywhere on the page and be part of the form.

The id Attribute

The **id** attribute assigns unique identification to the form. This id is used by Javascript or other client-side scripts to reference the form. The id can also be used to apply styles to the form. You can use any identification of your choosing for a form; the id value must be unique and should not include embedded blank spaces or special characters.

The action Attribute

User information entered into a form is made available to a server page containing a script to processes that information. The script may appear on the same page as the form or the information may be sent to a different page.

The **action** attribute identifies the server page to which form information is submitted. If the processing page is the same page as the form page, or if the processing page is in the same directory as the form page, then the URL is simply the name of the page. If the page is at a remote location, a complete URL is coded. The action attribute must appear in all `<form>` tags.

The method Attribute

The **method** attribute specifies the manner in which form information is transmitted to the page identified in the action attribute. The two possible values are get and post, the former being the default. When the get method is used, information from the form is appended to the action URL. The get method is best suited for sending small amounts of data; when the post method is used, the information is transmitted to the action page as a separate data stream. Thepost method offers better security since the submitted data is not visible in the browser's address bar.

The autocomplete Attribute

The **autocomplete** attribute is used to dynamically fill form fields. Autocomplete can be disabled by setting value to off. This attribute is new to HTML 5.

All that is required at present is to code the basic `<form>` tag to enclose its controls. The minimal setup for a page containing a form is shown below, delaying for the moment considerations about form submission for processing.

```
<!DOCTYPE html>

<html lang="en">

<head>
  <title>Form Page</title>
  <meta charset="utf-8">
</head>
<body>

<form action="  url  >

  ...form controls

</form>

</body>
</html>
```

Figure 11.2.2: General page layout for a form.

11.2: Creating Forms is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.3: Textbox Controls

The most commonly encountered type of form control is the **textbox**. This control presents a standard text entry box into which information is typed. A text input field is created using an `<input>` tag in the following format.

```
<input type="text | password | email (HTML 5)|url (HTML 5) "
 id="id"
 name="name"
 size="n"
 maxlength="n"
 value="text"
 disabled="disabled"
 readonly="readonly"
 accesskey="keyboard character"
 tabindex="numeric value"
 required="required (HTML 5)"
 placeholder="text (HTML 5)"
 autocomplete="on|off (HTML 5)"
 autofocus="autofocus (HTML 5)"
>
```

Figure **11.3.1**: General format for `<input type="text|password">` tag.

The type Attribute

The `<input>` tag includes a **type** attribute to define this as a text entry field. Many different types of text input controls are available.

type="text"

If type="text", or if no type attribute is specfied, the field type is a standard textbox for entering information.

type="password"

If type="password", a similar control is created but with typed characters echoed as bullet characters to disguise the entered information.

type="email" (HTML 5)

The input type="email" specifies a text box that accepts information that must be in e-mail format, such as "admin@itwebtutorials.net". Only browsers that support the HTML 5 email attribute will verify the format of the information. Older browsers will treat this as a standard text box control.

type="url" (HTML 5)

The type="url" is intended to accept any valid URL value, such as "http://itwebtutorials.net". Only browsers that support the HTML 5 url attribute will verify the format of the information. Older browsers will treat this as a standard text box control.

The control types are shown below along with the coding to produce them. The controls appear in a borderless table in order to align them and their prompt labels. Enter any text into the fields to view their appearance.

Please enter the following information:

Name:	
Password:	

Figure **11.3.2**: Varieties of `<input type="text|password"/> tag.` tag.

```
<form action="ThisPage.htm">
<p>Please enter the following information:</p>

<table>
<tr>
  <td>Name: </td>
  <td>  <input id="TheName" type="text">  </td>
</tr>
<tr>
  <td>Password: </td>
  <td>  <input id="ThePassword" type="password">  </td>
</tr>
</table>

</form>
```

Figure **11.3.3**: Code to display varieties of `<input type="text|password">` tag.

The id Attribute

You will nearly always need to identify your text fields, as you will any other form control. Providing an **id** is important because it provides identification for the data value entered into the field by the user. Both browser or client-side scripts written to process this information does so through the control id. Controls that are assigned an **id** value can also be styled using a CSS id selector.

The name Attribute

The **name** attribute provides another way to identify a form control. While client-side scripts process control information using the id, server-side scripts typically use the name attribute value to interact with the control. In most cases, the id and name value assigned to a form control can be the same.

The size Attribute

Unless you specify a **size** for a textbox, it is displayed at its default size, which is large enough for approximately 20 typed characters. In most cases you will want to specify a size that is suggestive of the number of characters expected to be entered. For example, the three textboxes below are sized at 15 (City), 2 (State), and 10 (Zip code) characters, respectively. Instead of using the size attribute you can specify a field size using the style sheet width property.

City:	
State:	
Zip:	

Figure **11.3.4**: Setting display sizes for textbox controls.

```
<form action="ThisPage.htm">
<table>
<tr>
  <td>City: </td>
  <td><input type="text" id="City"></td>  size="15"  ></td>
</tr>
<tr>
  <td>State: </td>
  <td><input type="text" id="State"   size="2"  ></td>
</tr>
<tr>
  <td>Zip: </td>
  <td><input type="text" id="Zip"   size="10"  ></td>
</tr>
</table>
</form>
```

Figure 11.3.5: Code to set display sizes for textbox controls.

The maxlength Attribute

A textbox can hold up to 256 characters, irrespective of its display size. When typing reaches the right boundary of the field, the text scrolls to permit additional characters to be typed. As a general rule, you should not permit the user to enter more than the maximum number of expected characters. When capturing data for server processing, this maximum is often given by the field sizes in the database that stores the values.

You can set a maximum number of allowable characters by coding the **maxlength** attribute. When this value is coded, the user is not be able to type more than the specified number of characters into the field.

```
<form action="ThisPage.htm">
<table>
<tr>
  <td>City: </td>
  <td><input type="text" id="City" size="15"   <maxlength="15"  ></td>
</tr>
<tr>
  <td>State: </td>
  <td><input type="text" id="State" size="2"   maxlength="2"  ></td>
</tr>
<tr>
  <td>Zip: </td>
  <td><input type="text" id="Zip" size="10"   maxlength="10"  ></td>
</tr>
</table>
</form>
```

Figure 11.3.6: Code to set maximum characters entered into textbox controls.

The value Attribute

A textbox can include the **value** attribute to pre-enter text in the field. You can provide a default value for the field, or give suggestions about the content expected.

Name:

Enter your full name here

Figure **11.3.7**: Pre-filling a textbox control.

```
Name: <input type="text" id="FullName" size="30"   value="Enter your full name here"  />
```

Figure **11.3.8**: Code to pre-fill a textbox control.

The disabled Attribute

The **disabled** attribute is used to disable and grays out the control so that the user can not interact with it. The **disabled** attribute is often associated with controls that are assigned default values using the value attribute. With the **disabled** attribute, users are not able to change or copy the default value.

The readonly Attribute

The **readonly** attribute is similar to the disabled attribute and is used to prevent the user from changing the value of the control. The **readonly** attribute is often associated with controls that are assigned default values using the **value** attribute. Unlike controls assigned the **disabled** attribute, the contents of a control with the **readonly** attribute can be copied.

The accesskey Attribute

The accesskey attribute is used to configure a shortcut key for the form control. This allows the user to press a key to place focus on the form control.

Name:

```
Name: <input type="text" id="FullName" size="30"   acesskey="n"  >
```

Figure **11.3.9**: Code to provide access key.

The method for accessing the shortcut key various by browser, with most using the [ALT]+ shortcut key. Firefox uses [ALT][SHIFT] + shortcut key.

The tabindex Attribute

The **tabindex** attribute is used to specify the tab order of the textbox if the tab key is used to navigate the form controls. If numeric value assigned to the attribute is 2, the textbox would have focus set when the user presses the tab key twice.

The required Attribute

The **required** attribute is new to HTML 5. It can be included with the text, url, email, and password text box controls. When the attribute is set to **true**, it specifies that the field must contain a value before submitting the form.

The placeholder Attribute

The **placeholder** attribute is new to HTML 5. It is used to specify a short hint that describes the expected value of an input field. The placeholder text is display in the text box until the user inputs a value.

The autocomplete Attribute

By default the **autocomplete** attribute is set to **true**. The attribute specifies whether or not the browser should predict and display options to fill in the textbox. Set autocomplete to **false** to disable the autocomplete functionality.

The autofocus Attribute

The **autofocus** attribute specifies if the text box should receive focus automatically when the page loads.

Using Quotes in Textboxes

When pre-filling a textbox through the value attribute, you *cannot* code quotation marks surrounding portions of text. This is because the value attribute itself is surrounded by quotes, and any quotation marks inside these container quotes disrupts their pairing. So, for example, the following code is *invalid* and does not produce a proper textbox display.

```
<input id="QuoteText" type="text" size="45"
  value="Here is a   "quotation"   appearing in the textbox."/>
```

Here is a

Figure 11.3.10:Invalid pre-filling of a textbox with quoted content.

You can, however, use the special **"** character to display quotation marks inside a textbox value attribute.

```
<input id="QuoteText" type="text" size="45"
  value="Here is a &amp;quot;quotation&amp;quot; appearing in the textbox."/>
```

Here is a "quotation" appearing in the textbox.

Figure 11.3.11: Valid pre-filling of a textbox with quoted content.

No special precautions need to be taken, however, for user text entry. Quoted text can be entered into a textbox without presenting problems. The same goes for HTML tags. They can be coded inside a value attribute and also entered by the user without causing problems.

Styling Textboxes

If you wish to change the default appearance of a textbox, you can use style sheet properties to do so. As an example, the following textbox is decorated with different borders, background color, and foreground color, and displays a different font face and size from the normal Arial 10-point font.

Name:

Enter your full name here

Figure 11.3.12: Styling a textbox control.

```
<style>
  .textbox   {width:200px; font-family:comic sans ms; font-size:10pt;
             background-color:#F0F0F0; color:#0000FF; border:ridge 5px}
</style>

Name: <input type="text" id="FullName"   class="textbox"
maxlength="20" value="Enter your full name here">
```

Figure **11.3.13**: Code to style a textbox control.

Other HTML 5 Text Controls

HTML 5 introduces an assortment of new form controls that can be used to collect text data. The display and support of these new controls varies by browser. Browsers that do not support the new text controls will simply ignore the new elements and display them as a standard text box. In addition to the HTML 5 text controls introduced above, other new text form controls include the telephone, search, datalist, range, number, calendar, and color controls.

Form Control	Display	Code
Telephone - accepts a telephone number		<input type="tel">
Search - accepts a search term		<input type="search">
Datalist - specifies a list of pre-defined options for a text box		<input list="colors"> <datalist id="colors"> <option value="red"> <option value="green"> <option value="blue"> </datalist>
Range(Slider) - provides a visual interface that accepts a numeric value. The default range is 1 to 100	Low High	<input type="range" min="1" max="10">
Number (Spinner) - displays an interface that accepts numerical data and rovides feedback to users		<input type="number" min="1" max="10">
Calendar (date)- accepts date information	mm/dd/yyyy	<input type="date">
Color picker - accepts color value		<input type="color">

11.3: Textbox Controls is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.4: Textarea Control

The `<textarea>` tag is used when you need a larger input area than is provided by a textbox. This tag creates a multiline input box that displays horizontal and vertical scroll bars and word-wraps the text. A typical textarea control resembles the following.

Enter your comments:



Figure **11.4.1**: A `<textarea>` control.

The `<textarea>` control is a container tag with the general format shown in Figure **11.4.1**.

```
<textarea
  id="id"
  cols="n"
  rows="n"
  maxlength="n"
  rows="n"
  readonly="readonly"
  disabled="disabled"
  placeholder="short hint text"
  autofocus="autofocus"
  wrap="soft|hard"
>
  ...pre-entered text
</textarea>
```

Figure **11.4.2**: General format for `<textarea>` control.

The name Attribute

As with all form controls, the `<textarea>` should be assigned a name to identify the control. This name should be representative of the information contained in the field. The namevalue is used by server-side scripts when processing form data.

The id Attribute

As with all form controls, the `<textarea>` should be assigned an id. This id should be representative of the information contained in the field. The id is primarily used by client-side scripts when processing form data.

The cols Attribute

The visual width of the textarea is given by the **cols** attribute. It specifies the width as the approximate number of characters appearing across a line of text. If this attribute is not present, a default width of approximately 20 characters is used. A style sheet width property can be used in place of the cols attribute to set the width of the textarea.

The rows Attribute

The height of the textarea is given by the **rows** attribute. This value specifies the number of text lines that are visible in the input area. The default is two rows of visible text. A style sheet height property can be used in place of the rows attribute to set the height of the textarea.

The maxlength Attribute

As a general rule, you should not permit the user to enter more than the maximum number of expected characters. When capturing data for server processing, this maximum is often given by the field sizes in the database that stores the values.

You can set a maximum number of allowable characters by coding the **maxlength** attribute. When this value is coded, the user is not be able to type more than the specified number of characters into the field.

The disabled Attribute

The **disabled** attribute is used to disable and grays out the control so that the user can not interact with it. The **disabled** attribute is often associated with controls that are assigned default values using the **value** attribute. With the **disabled** attribute, users are not able to change or copy the default value.

The readonly Attribute

The **readonly** attribute is similar to the **disabled** attribute and is used to prevent the user from changing the value of the control. The **readonly** attribute is often associated with controls that are assigned default values using the **value** attribute. Unlike controls assigned the **disabled** attribute, the contents of a control with the **readonly** attribute can be copied.

The required attribute is new to HTML 5. It can be included with the text, url, email, and password text box controls. When the attribute is set to **true**, it specifies that the field must contain a value before submitting the form.

The placeholder Attribute

The **placeholder** attribute is new to HTML 5. It is used to specify a short hint that describes the expected value of an input field. The placeholder text is display in the text box until the user inputs a value.

The autofocus Attribute

The **autofocus** attribute specifies if the text box should receive focus automatically when the page loads.

The wrap Attribute

The **wrap** attribute configures line breaks within the information entered. When the wrap attribute is set to **soft**, the text input is not wrapped. When the attribute is set to **hard**, text in the textarea is wrapped when submitted.

Pre-filled Content

A textarea can be pre-filled with text. The information to display in the textarea is coded between the opening and closing `<textarea>` ... `<textarea>` tags.

It is important to remember that any spaces, line breaks, or blank lines appearing in the editor coding are reproduced in the textarea, in much the same way that the browser displays information coded inside a `<pre>` tag. In the following example, editor coding in Figure **11.4.3** arranges the tag and its content on separate lines and indented for readability within the editor. Its browser display in Figure **11.4.4**, however, is not as normal word-wrapped paragraphs; rather, the editor coding is faithfully reproduced.

```
<textarea id="MyTextarea" cols="50" rows="6">
  These are two paragraphs appearing
  inside a textarea.

  This content, with line breaks and blank lines coded in the
  editor For readability, Is exactly reproduced within the textarea.
```

Figure **11.4.3**: Coding a textarea with hard-coded spaces and line breaks.



Figure **11.4.4**: A textarea displaying hard-coded spaces and line breaks.

To permit pre-fill text to be automatically spaced and word-wrapped inside a textarea, the text should be coded immediately following the opening tag and followed immediately by the closing tag. No hard spaces or line breaks should be coded. The following example shows pre-entered paragraphs coded as *single lines* in the code editor to permit proper word wrapping in the textarea.

```
 id="MyTextarea" cols="50" rows="6">
 two word-wrapped paragraphs appearing in a textarea.

ent, without extra line breaks or blank lines coded in the editor, is properly word-wrapped inside the textarea as
a>
```

Figure **11.4.5**: Coding a textarea for word-wrapped content.



Figure **11.4.6**: A textarea displaying word-wrapped content.

HTML Tags and Quotes in a Textarea

HTML tags entered into a textarea do not have any formatting effects. They are treated as standard text characters. In the following illustration, HTML tags appear in a textarea, either as pre-filled text or as user data entry, but do not apply their formatting.

```
<textarea id="TagInput" cols="40" rows="6">
The <p> tag appears around a block of
text to offset it from surrounding
text by blank lines. A <div> tag, in
contrast, does not produce blank lines
surrounding its content.
</textarea>
```

Figure **11.4.7**: Coding a textarea with HTML tags.



Figure **11.4.8**: A textarea with HTML tags.

Unlike the case with textbox values, quotation marks can appear within pre-filled text and can also be entered by the user without causing textarea display or data-entry problems.

Styling a Textarea

A textarea uses a default Courier New font face with 12-point type. You can change this setting as well as style the textarea using a style sheet as shown below.

```
<style type="text/css">
  textarea {style="border:inset 5px; background-color:#F0F0F0; color:#0000FF;
            font-family:comic sans ms; font-size:10pt; padding:7px}
</style>

<textarea id="Comments" cols="50" rows="4">
This control is styled with 10-point Comic Sans
MS type face with gray background and blue
foreground colors.

The border is inset style, five pixels in width.
Padding of 7 pixels appears around the text.
</textarea>
```

Figure **11.4.9**: Coding to style a textarea.



Figure **11.4.9**: A styled textarea.

11.4: Textarea Control is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.5: Radio Buttons

Radio buttons are collections of circular buttons that can be clicked "on" and "off." A filled-in, darkened center in the button indicates that the button is chosen; a open center indicates that the button is not chosen. Radio buttons normally provide users with *mutually exclusive* choices. That is, only one of a group of buttons can be chosen. When a different button is clicked, the previous button is turned off. A set of radio buttons is shown below.

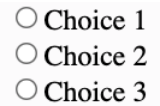


Figure **11.5.1**: A set of radio buttons.

The `<input type="radio">` Tag

Radio buttons are defined with one or more `<input type="radio">` tags. A separate tag is required for each button in a set. The general format for this tag is shown below.

```
<input type="radio"
  id="id"
  name="name"
  value="text"
  checked="checked"
  disabled="disabled"
  readonly="readonly"
  required="required"
>
```

Figure **11.5.2**: General format for `<input type="radio">` tag.

The `<input type="radio">` tag only displays a button. In order to provide a label for the button, text can appear either before or after the button code to place it to the left or right of the button.

The id Attribute

An **id** can be assigned to a button if there is a need to identify it for individual script processing, normally by a browser script. Otherwise, an id is not needed.

The name Attribute

Radio buttons are usually grouped to present a set of related choices for the user. In this case, only one of the options can be chosen -- they are mutually exclusive choices. When a button is clicked for selection, and that button is highlighted, any previously chosen button becomes de-selected and un-highlighted. A group of radio buttons is made to act in this way by assigning the *same* **name** to all buttons in the group, as shown in the following code for the buttons displayed above.

```
<input type="radio"   name="Choice"  >Choice 1<br>
<input type="radio"   name="Choice"  >Choice 2<br>
<input type="radio"   name="Choice"  >Choice 3<br>
```

Figure **11.5.3**: Coding to group radio buttons.

There are three buttons in this group, each appearing on a separate line. A text label appears to the right of the buttons to identify each choice. All buttons have the same name to enforce only one choice from the group.

The value Attribute

In a manner similar to text boxes, password boxes, and textareas, radio buttons have data values associated with the choices. These values are provided explicitly through **value** attributes coded for the buttons. When a button is chosen, its particular value is associated with the name of the button, that is, with the name assigned to all of the buttons in the group. Consider the following example.

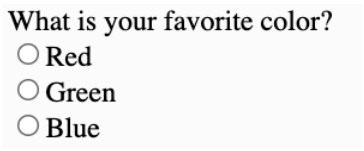


Figure **11.5.4**: A set of buttons for making a choice of a color.

```
What is your favorite color?<br>
<input type="radio" name="Color"   value="Red"  >Red<br>
<input type="radio" name="Color"   value="Green"  >Green<br>
<input type="radio" name="Color"   value="Blue"  >Blue<br>
```

Figure **11.5.5**: Code for a set of buttons to make a color choice.

All of the buttons share the name "Color" in order to set up a group of mutually exclusive choices. The particular color chosen from the group is given by the value associated with the choice. So, if the user clicks the first button, the color "Red" is chosen. More accurately, the value "Red" becomes associated with the name "Color" as indicative of the choice.

It is normally always necessary to code value attributes for radio buttons. This is the way in which buttons take on values and how a browser script or server program determines which button is clicked. Also, it is not necessary to assign a value that matches the button label. Often, abbreviated codes are used for values, while a more descriptive label is displayed.

```
What is your favorite color?<br>
<input type="radio" name="Color"   value="R"  >Red<br>
<input type="radio" name="Color"   value="G"  >Green<br>
<input type="radio" name="Color"   value="B"  >Blue<br>
```

Figure **11.5.6**: Code for a set of buttons to make a "coded" color choice.

A selection of one of the above buttons submits the name and value Color=R, Color=G, or Color=B to the processing program, which then determines what action to take on the value code submitted.

The checked Attribute

When a set of radio buttons is first displayed, all of the buttons appear unchecked; that is, none are selected and highlighted. You can, however, force one of the buttons in the group to appear pre-checked, or pre-selected. You do this by coding the **checked="checked"** attribute in the `<input type="radio">` tag of the button you want to appear "on" when the page is loaded.

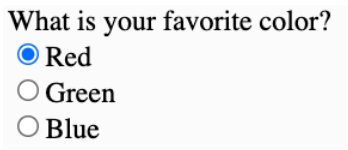


Figure **11.5.7**: VA set of buttons with the first choice pre-selected.

```
What is your favorite color?<br>
<input type="radio" name="Color" value="R"   checked="checked"  >Red<br>
<input type="radio" name="Color" value="G">Green<br>
<input type="radio" name="Color" value="B">Blue<br>
```

Figure 11.5.8: Code to pre-select a button.

The disabled Attribute

The **disabled** attribute is used to disable and grays out the control so that the user can not interact with it. The **disabled** attribute is often associated with controls that are assigned default values using the **value** attribute. With the **disabled** attribute, users are not able to change or copy the default value.

The readonly Attribute

The **readonly** attribute is similar to the **disabled** attribute and is used to prevent the user from changing the value of the control. The **readonly** attribute is often associated with controls that are assigned default values using the **value**attribute. Unlike controls assigned the **disabled** attribute, the contents of a control with the **readonly** attribute can be copied.

The required Attribute

The required attribute is new to HTML 5. It can be included with the text, url, email, and password text box controls. When the attribute is set to **true**, it specifies that the field must contain a value before submitting the form.

11.5: Radio Buttons is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.6: Checkboxes

A **checkbox**, like a radio button, provides the user with choices. In the case of a series of checkboxes, however, the choices are not mutually exclusive; rather, the user can select *one or more* of the choices. For those boxes that are checked, a checkmark appears in the box. Unchosen boxes remain empty. A set of checkboxes is illustrated below.



Figure **11.6.1**: A set of checkboxes.

The `<input type="checkbox">` Tag

The general format for coding a checkbox is similar to that for a radio button. An `<input type="checkbox">` tag is required for each checkbox.

```
<input type="checkbox"
  id="id"
  name="name"
  value="text"
  checked="checked"
  disabled="disabled"
  readonly="readonly"
  required="required"
>
```

Figure **11.6.2**: General format for `<input type="checkbox">` tag.

The `<input type="checkbox">` tag only displays a checkbox. In order to provide a label for the checkbox, text can appear either before or after the tag to place it to the left or right of the checkbox.

The id Attribute

An **id** can be assigned to a checkbox if there is a need to identify it for script processing, normally by a browser script. Otherwise, an id is not needed.

The name Attribute

Checkboxes may need to be named with **name** attributes so that chosen values can be associated with the names for server script processing. As is the case with radio buttons, all checkboxes in a group can have the same name. Unlike radio buttons, however, a common name does *not* force the choices to be mutually exclusive. The name simply takes on more than one value if more than one choice is made. This presents some processing peculiarities, but it is valid to do this. A better solution is to assign different names to the checkboxes. Each name then becomes associated with a different value irrespective of the number of checks made by the user. If a set of checkboxes is involved only in browser processing, then id values, rather than names, can be coded for the checkboxes.

The value Attribute

Similar to a radio button, the value associated with a checkbox is given in the checkbox's **value** attribute. When a form is submitted for server processing, this value is associated with the name of the particular checkbox. For browser processing, the value is associated with the checkbox's id. Values do not have to match the labels. Normally, abbreviated codes are used for values, and more descriptive text is used for labels.

The checked Attribute

Checkboxes can be pre-checked by coding **checked="checked"** for as many different checkboxes as you want pre-checked.

The various settings for checkboxes are shown below for the set appearing above in Figure 10-21. In this example, all checkboxes are assigned different names to differentiate them for server processing. No checkboxes are pre-checked. No id values are assigned since these checkboxes are processed by a server script rather than a browser script.

```
What are your favorite colors?<br>
<input type="checkbox" name="Color1" value="R">Red<br>
<input type="checkbox" name="Color2" value="G">Green<br>
<input type="checkbox" name="Color3" value="B">Blue<br>
<input type="checkbox" name="Color4" value="Y">Yellow<br>
<input type="checkbox" name="Color5" value="M">Maroon<br>
<input type="checkbox" name="Color6" value="P">Purple<br>
<input type="checkbox" name="Color7" value="A">Aqua<br>
<input type="checkbox" name="Color8" value="T">Teal<br>
```

Figure 11.6.3: Code for a set of checkboxes.

The disabled Attribute

The **disabled** attribute is used to disable and grays out the control so that the user can not interact with it. The **disabled** attribute is often associated with controls that are assigned default values using the **value** attribute. With the **disabled** attribute, users are not able to change or copy the default value.

The readonly Attribute

The **readonly** attribute is similar to the **disabled** attribute and is used to prevent the user from changing the value of the control. The **readonly** attribute is often associated with controls that are assigned default values using the **value** attribute. Unlike controls assigned the **disabled** attribute, the contents of a control with the **readonly** attribute can be copied.

The required Attribute

The required attribute is new to HTML 5. It can be included with the text, url, email, and password text box controls. When the attribute is set to **true**, it specifies that the field must contain a value before submitting the form.

11.6: Checkboxes is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.7: Drop-Down Lists

A **drop-down list**, presents a list of items from which one or more are chosen. By clicking the down-arrow at the right of the list, the full list is exposed. An item is chosen by clicking an entry.

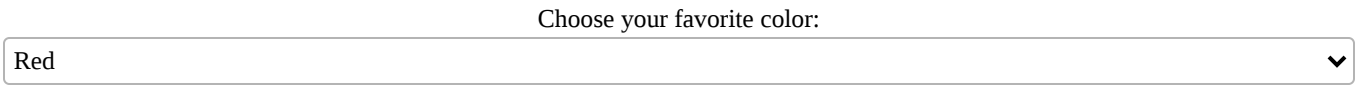


Figure **11.7.1**: A drop-down list.

The `<select>` and `<option>` Tags

The menu of choices is created with the `<select>` tag. Inside this tag are `<option>` tags, one for each item in the list. The general formats for the `<select>` and `<option>` tags are shown below.

```
<select
  id="id"
  name="name"
  size="n"
  multiple="multiple"
  disabled="disabled"
>
  <option
    value="text"
    selected="selected"
   disabled="disabled"
>
     Label
  </option>
  ...

</select>
```

Figure **11.7.2**: General formats for `<select>` and `<option>` tags.

The id Attribute

An **id** can be assigned to a drop-down list if there is a need to identify it for browser script processing. Otherwise, an id is not needed.

The name Attribute

The drop-down list can be assigned a name with the **name** attribute coded in the `<select>` tag. The list name becomes associated with the value of the item selected for certain types of server processing.

The `<option>` Tag

Items are defined for appearance in a drop-down list with the `<option>` tag. There are as many tags as there are items in the list. The label for the option is entered between the opening and closing tags. This is the text string that is visible in the list and becomes the default value for the selection.

The value Attribute

By default, the value associated with a drop-down option is given by the text label included in the `<option>` tag. However, a **value** attribute can be coded to supply a different value from the label. You might choose to do this when the labels are extended text strings but you need only to collect abbreviated codes for the values.

The code below shows the basic tags and attributes needed to create a drop-down list. It produces the list shown above in Figure 10-23.

```
Choose your favorite color:<br>
<select id="Color">
  <option value="R">Red</option>
  <option value="G">Green</option>
  <option value="B">Blue</option>
  <option value="Y">Yellow</option>
  <option value="M">Maroon</option>
  <option value="P">Purple</option>
  <option value="A">Aqua</option>
  <option value="T">Teal</option>
</select>
```

Figure **11.7.3**: Basic code for a drop-down list.

The size Attribute

The **size** attribute of the `<select>` tag indicates the number of items that are visible in the list. By default, the list displays only one item. If a size is specified, the list displays that number of items and, if necessary, a scroll bar to access them. A drop-down list that uses the size attribute or displays more than one option is often referred to as a **selection list**. The list shown below uses the attribute size="4" to display four items at a time.

Choose your favorite color:

Red
Green
Blue
Yellow

Figure **11.7.4**: A selection list permitting multiple selections.

The selected Attribute

Items in the list can be pre-selected by coding the **selected="selected"** attribute in the associated `<option>` tag. Multiple pre-selections can be made only when multiple="multiple" is coded for the `<select>` tag. The selection list in Figure 10-26 has three items (Red, Blue, Aqua) pre-selected.

The disabled Attribute

The **disabled** attribute is used to disable and grays out the control so that the user can not interact with it. The **disabled** attribute is often associated with controls that are assigned default values using the **value** attribute. With the **disabled** attribute, users are not able to change or copy the default value.

The `<optgroup>` Tag

The items in a drop-down list can be grouped using the `<optgroup>` element. The tag includes a label attribute that is used to specify the label for each group of options in the list. The code below shows how the tag is used.

```
Select a Tree:<br>
<select id="Trees">
<optgroup label="Pines">
        <option value="Longleaf">Long Leaf Pine</option>
        <option value="Loblolly">Loblolly Pine</option>
</optgroup>
<optgroup label="Oaks">
        <option value="Red">Southern Red Oak</option>
        <option value="Live">Live Oak</option>
        <option value="White">White Oak</option>
        <option value="Scarlet">Scarlet Oak</option>
</optgroup>
<optgroup label="Elms">
        <option value="American">American Elm</option>
        <option value="Slippery">Slippery Elm</option>
</optgroup>
</select>
```

Select a Tree:

Long Leaf Pine

Figure **11.7.5**: Code to group items in the dropdown

11.7: Drop-Down Lists is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.8: Submit and Reset Buttons

All forms that are submitted for server processing must include at least one "submit" button to transmit form information to the Web page identified in the action attribute of the `<form>` tag. Either a standard button or a graphic image can be used for form submission.

The `<input type="submit">` Button

A form submission button can be created with an `<input type="submit">` tag and can appear anywhere on the form. The default appearance of the button with its "Submit Query" label is shown below.



Figure 11.8.1: Default appearance of a form submission button.

The general format for the `<input type="submit">` tag to define a form submission button is given in Figure 11.8.2.

```
<input type="submit"
  id="id"
  name="name"
  value="text"
>
```

Figure 11.8.2: General format for `<input type="submit">` button.

The id Attribute

An **id** can be assigned to a submit button if there is a need to identify it for browser script processing. Otherwise, an id is not needed.

The name Attribute

The submit button can be assigned a name with the **name** attribute. A name is required for certain types of server processing.

The value Attribute

The **value** attribute provides two kinds of identification for the button. On the one hand, this value is associated with the button name and indicates to a server script which button was clicked; on the other, this value is used as the label for the button. If a value attribute is not coded, the button is labeled "Submit Query"; however, you can assign any text string to provide helpful identification of the button as a submit button.

Typical coding and display for a submit button is shown below.

```
<input type="submit" name="SubmitButton" value="Submit Form">
```



Figure 11.8.3: Typical code and display for `<input type="submit"/>` button.

The `<input type="image"/>` Tag - TOP

An alternative to the standard submit button is to use a graphic image to trigger form submission. For example, the following "Go" button functions identically to a standard submit button.

```
<input  type="image"  name="SubmitButton" src="GoButton.gif"
  alt="Click to Submit">
```



Figure 11.8.4: A graphic submit button.

An image button is created with an `<input type="image">` tag whose general format is given in Figure 11.8.5.

```
<input type="image"
  id="id"
  name="name"
  src="url"
  alt="text"
>
```

Figure 11.8.5: General format for `<input type="image">` button.

The tag uses type="image" to identify this as a graphic form submission control. The particular image that is used is given by the URL in the src attribute. Other attributes of an `<img>` tag can be coded, including alt for an alternative text description of the image. As with standard submit buttons, you may need to assign a name or id to identify the button to server or browser scripts.

The `<input type="reset">` Tag

One other button type is available for forms. A "reset" button can be created to permit users to clear all information from a form, either to permit entry of new information or to clear incorrect information. Its default appearance is shown below.

Reset

Figure 11.8.6: Default appearance of a form reset button.

This button is created by coding an `<input type="reset">` tag. You can name or id the button and can replace the default "Reset" label by coding its value attribute. The action of the button is to automatically reset the form, clearing all text input areas and resetting radio buttons, checkboxes, and drop-down lists back to their default appearances.

The following example shows use of a reset button. Enter text into the textbox; then click the reset button. Information in the textbox is cleared.

Enter text:

Clear Form

Figure 11.8.7: Using a form reset button.

Styling Buttons

Submit and reset buttons can take advantage of style sheet settings to change their appearance. Example styling is shown by the following two buttons.

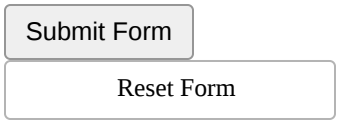


Figure **11.8.8**: Styled submit and reset buttons.

```
<style>
  .submit {width:150px; background-color:#4682B4; color:#FFFFFF;
  font-family:times new roman; font-size:14pt}
  .reset  {width:150px; background-color:#4682B4; color:#FFFFFF;
  font-family:times new roman; font-size:14pt}
</style>

<input type="submit" name="SubmitButton" value="Submit Form" class="submit">
<input type="reset" name="ResetButton" value="Reset Form" class="reset">
```

Figure **11.8.9**: Code to style form submit and reset buttons.

11.8: Submit and Reset Buttons is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.9: Other Buttons

The two primary types of buttons appearing on forms are submit and reset buttons. These two buttons are associated with form submission to programs that process form information. Other general-purpose buttons can be defined. These are normally associated with browser scripts for local processing, of forms and otherwise, by JavaScript routines embedded on the Web page.

The `<input type="button">` Tag

An `<input type="button">` tag is used to create general-purpose buttons to call browser scripts located on the same Web page. Its general format is shown below.

```
<input type="button"
  id="id"
  name="name"
  value="text"
>
```

Figure **11.9.1**: General format for `<input type="button">` tag.

An **id** or **name** may be necessary if the button is referenced in a script; otherwise these attributes need not be coded. A **value** attribute is required to supply a label for the button; otherwise, an unlabeled button appears.

This type of button usually has the purpose of responding to a mouse click and activating an embedded script or calling a JavaScript function to perform some processing.

The `<button>` Tag

The `<button>` tag provides versatility in producing buttons. It can be set up as a submit button, a reset button, or a general purpose button. As a container tag, it can enclose any text or HTML code to provide text and/or graphic displays on the face of the button. The general format for this tag is shown below

```
<button
  type="submit|reset|button"
  id="id"
  name="name"
>
 ...text and HTML tags
</button>
```

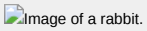
Figure **11.9.2**: General format for `<button>` tag.

When defined as **type="submit"** or **type="reset"**, the button works like any other submit or reset button. When defined as **type="button"**, it requires a browser script to take action on a button click.

The label for the button appears between the opening and closing tags. This can be a simple text label, or HTML tags can be coded to decorate the text or to include a graphic image on the button.

The following general-purpose button illustrates a layout using HTML tags to supply a graphic image and text for the face of the button.

Rabbit




```
<style>
 button     {float:left; height:100px; width:180px; border:outset 3px;
             margin-right:10px}
 span#Title {font-family:impact; font-size:14pt; color:#DAA520}
 span#Text  {font-family:verdana; font-size:10pt}
 img        {width:50px; height:39px}
 iframe     {width:225px; height:185px; border:outset 10; margin-right:65px}
</style>

<button type="button"
 onclick="
   if (document.getElementById('Text').innerHTML == 'View Picture') {
     document.getElementById('Picture').src='W-Colossus.gif'
     document.getElementById('Text').innerHTML='Hide Picture' }
   else if (document.getElementById('Text').innerHTML== 'Hide Picture') {
     document.getElementById('Picture').src=''
     document.getElementById('Text').innerHTML='View Picture' }"
>
  <span id="Title"> Rabbit </span><br/>
  <img src="bunny.gif" alt="Image of a rabbit."/><br/>
  <span id="Text"> View Picture </span>
</button>

<iframe id="Picture" scrolling="no"></iframe>
```

Figure **11.9.3**: Code to style and script a `<button>` tag.

Notice how the text and image appearing on the button are coded inside the opening and closing `<button>` tags in much the same way as they would be coded on a page. (Note: An `<iframe>` tag normally is coded with a name attribute that identifies it as the target for links. Since the frame is scripted, however, an id is assigned for script reference.)

11.9: Other Buttons is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.10: Grouping and Labels

You can visually group together a set of controls on your form by placing a **group box** around them. A group box displays a labeled border around the set of controls.



Figure **11.10.1**: Using a group box around a form control.

The `<fieldset>` and `<legend>` Tags

A box is placed around a form control with the `<fieldset>` tag. The box is labeled with text coded in an enclosed `<legend>` tag. A fieldset is an in-line HTML element; therefore, it must be enclosed inside a block-level tag. The general formats for the two tags are shown below.

```
<fieldset>
  <legend> text </legend>
  ...form control
</fieldset>
```

Figure **11.10.2**: General formats for `<fieldset>` and `<legend>` tags.

A example group of fieldsets is shown in Figure **11.10.3**.

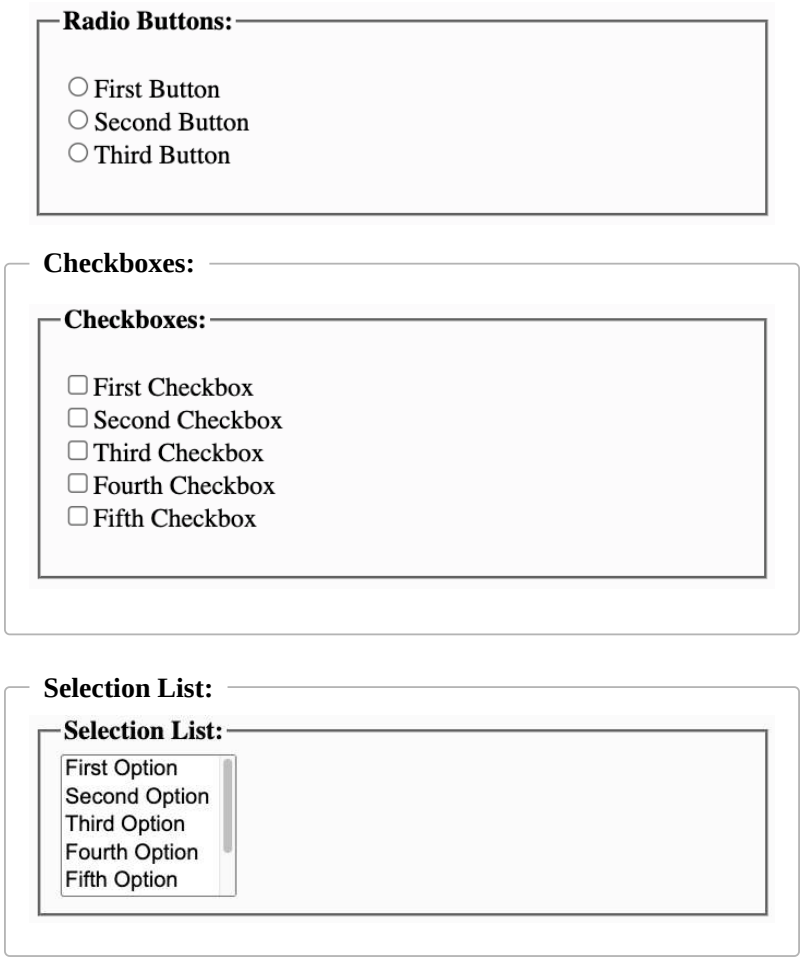


Figure **11.10.3**: Example fieldsets.

Code for these fieldsets is shown below. Since a fieldset expands to fill the *width of its container*, it is usually necessary to assign it a fixed width to control its placement and appearance. Note that the example fieldsets are assigned a width of 180 pixels to keep them from each stretching across the width of their enclosing division, permitting them to display side by side; also, they are vertically aligned at the top of their container division. A style sheet is applied to both the `<fieldset>` and `<legend>` tags to effect their styling.

```
<style>
 fieldset {width:180px; vertical-align:top}
 legend   {font-family:verdana; font-size:9pt; font-weight:bold; color:royalblue}
</style>

<div> <-- Block level container for fieldsets -->
    <fieldset>
        <legend> Radio Buttons: </legend>
        <input name="Radio" type="radio" /> First Button <br />
        <input name="Radio" type="radio" /> Second Button <br />
        <input name="Radio" type="radio" /> Third Button <br />
    </fieldset>

    <fieldset>
        <legend> Checkboxes: </legend>
        <input name="Box1" type="checkbox"/>  First Checkbox <br />
        <input name="Box2" type="checkbox"/> Second Checkbox <br />
        <input name="Box3" type="checkbox"/>  Third Checkbox <br />
        <input name="Box4" type="checkbox"/> Fourth Checkbox <br />
        <input name="Box5" type="checkbox"/>  Fifth Checkbox <br />
    </fieldset>

    <fieldset>
        <legend> Selection List: </legend>
        <select size="5">
            <option> First Option </option>
            <option> Second Option </option>
            <option> Third Option </option>
            <option> Fourth Option </option>
            <option> Fifth Option </option>
            <option> Sixth Option </option>
            <option> Seventh Option </option>
        </select>
    </fieldset>
</div>
```

Figure **11.10.4**: Code to group and label form controls.

The `<label>` Tag

Some form controls automatically have labels associated with them (buttons) while most do not (text boxes, checkboxes, radio buttons, text areas, and menus). The `<label>` tag is used to define a label for HTML form controls that do not have implicit labels. If you click the text within the label element, it toggles the control.

The "for" attribute associates a label with another control explicitly: the value of the for attribute must be the same as the value of the "id" attribute of the associated control element.

The code below demonstrates the use of the `<label>` tag:

```
<form action="ThisPage.htm">
<p> Please enter the following information: </p>

<table>
<tr>
  <td> Name:  </td>
  <td>  <label for="TheName">Enter Name</label><input id="TheName" type="text">  </td>
</tr>
<tr>
  <td> Password:  </td>
  <td>  <label for="TheName"> Enter Password </label><input id="ThePassword" type="password">  </t
</tr>
</table>

</form>
```

Figure **11.10.5**: Code to demonstrate use of the `<label>` tag.

The text between the opening and closing `<label>` tag becomes the label for the control. If the `<label>` tag is coded to the left of the control, the label appears on the left. On the other hand, if the `<label>` tag is coded to the right of the control, the label appears on the right.

The label tag is important for form page accessibility. It is helpful to visually challenged individuals who are using assistive technology software. The software is able to match up the label with the form control. all form controls are required to have an associated label.

Form Control Layout using CSS

In previous sections of this tutorial, tables were used to control the layout of form control elements. An alternative is to use CSS to format and display form controls. The advantage to using CSS is that it requires less code - elimnation of table tags and the line break tag. The code below demonstrates how to layout form controls using CSS.

```
<style>

    form#myForm {width:400px}
    label {width: 120px; float:left; text-align:right;
            margin-top: 10px; padding-right:10px}
    input {margin-top: 10px;}
    div#mySubmit {margin-top:10px;padding-bottom:10px}

</style>

-----------------------------------------------

<div>

<form>

    <div>
        <label for="FName"> First Name: </label>
        <input type="text" id="FName"/>
    </div>

    <div>
        <label for="LName"> Last Name: </label>
        <input type="text" id="LName"/>
    </div>

    <div id="mySubmit">
    <input type="submit" value="Submit Form"/>
    </div>

</form>

</div>
```

Figure **11.10.6**: Forms and CSS

First Name:

Last Name:

Submit Form

Using the CSS method, `<div>` elements are used to organize form elements. Each `<label>` and `<input>` control pair are coded within there own division container. The entire form area is given a width of 400px using the "myForm" id selector. Label element selectors are key to aligning the text - this selector is configured to float on the left with the text aligned to the right. The float causes the input controls to wrap around the right side of the labels. The input controls are also configured with a top margin. The submit button is styled using the "mySubmit" id selector. Left margin and bottom padding are applied to the submit button.

11.10: Grouping and Labels is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.11: Submitting Forms

It is important to remember that the purpose of forms is to collect information from users. Of course, this information has some further purpose, usually involving processing the submitted information in some fashion. Although browser-based scripts written in JavaScript can perform certain preliminary manipulation of the data, the bulk of processing activity takes place on the server, using a server-based programming language.

The manner in which server-based processing takes place is important to understand when designing forms. Although you will not be learning form processing here, you need a general understanding of how the server identifies form information submitted to it.

When a submit button or submit image is clicked, information from the form is packaged for delivery to the page identified in the action attribute of the `<form>` tag. When forms are submitted to a Web server, this information is transmitted by the method indicated, which can be either get or post. Usually, form information is submitted to the server using the post method, meaning that it arrives at the server in a separate data stream from the URL request. When the information arrives, it is made available to the URL page and its processing script.

Name/Value Pairs - TOP

Form information is identified and processed at the server through *names* and *values*. Each field on a form is assigned a name when the form is created. This unique identifier is used to reference that particular form control and the value associated with it. When information is typed into a textbox, for example, that text becomes the value of the control and can be referenced through the name assigned to the textbox. Likewise, when the user clicks on a checkbox or selects from a drop-down list, the data value associated with the checkbox or selection choice is referenced through the name assigned to that particular control.

Thus, form information ends up as field names and associated data values submitted by the user. Information arrives at the processing page as a string of **name/value** pairs in the format:

```
fieldname1=value1&fieldname2=value2&fieldname3=value3.
```

Control names and their associated values are paired with an equal sign (=) and are connected with other pairs of names and values by ampersands (&). One of the first tasks of a processing script is to **parse** this string of characters into separate names and values in order to extract the information entered into the form. Thereafter, the script manipulates the information according to whatever processing steps are necessary.

A Form Example - TOP

The following form does not include all the possible field types, but it does illustrate the approach to creating a simple form.

Membership Application Form

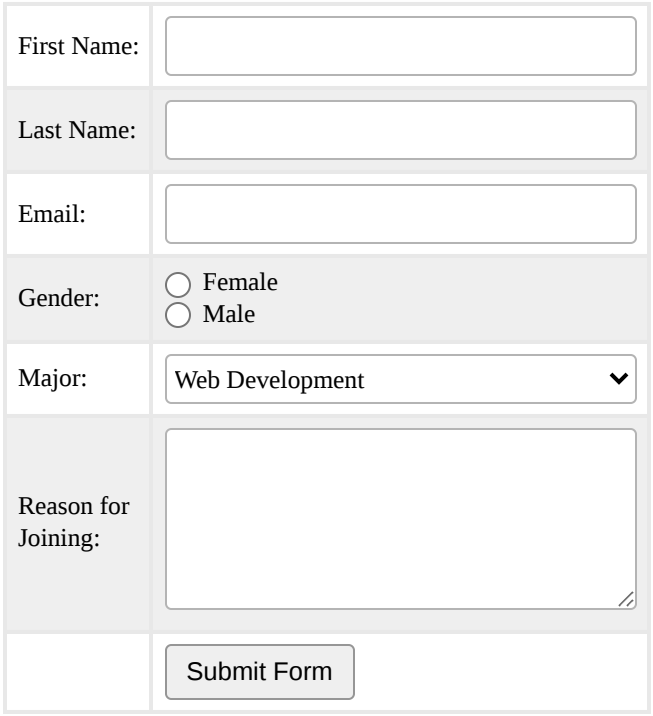


Figure 10-43. A data entry form.

The form elements have been coded inside a table structure to help align the labels and fields. You should recognize all of the form controls in the following code listing.

```
<form name="MyForm" action="MembershipForm.htm" metho

<h2> Membership Application Form </h2>

<table>
<tr>
  <td> First Name:  </td>
  <td><input type="text" name="FirstName" size="15" m
</tr>
<tr>
  <td> Last Name:  </td>
  <td><input type="text" name="LastName" size="15" ma
</tr>
<tr>
  <td> Email:  </td>
  <td><input type="text" name="Email" size="30" maxle
</tr>
<tr>
```

```
        <td> Gender:  </td>
        <td><input type="radio" name="Gender" value="F"/> F
            <input type="radio" name="Gender" value="M"/> M
      </tr>
      <tr>
        <td> Major:  </td>
        <td><select name="Major">
              <option> Web Development </option>
              <option> Digital Media </option>
              <option> Database Administration </option>
              <option> Networking </option>
              <option> Software Development </option>
              <option> Systems Analysis </option>
            </select></td>
      </tr>
      <tr>
        <td> Reason for <br/> Joining:  </td>
        <td><textarea name="Reason" cols="30" rows="5"></te
      </tr>
      <tr>
        <td></td>
        <td><input type="submit" name="SubmitButton" value=
      </tr>
      </table>

      </form>
```

Listing 11-24. *Code for a data entry form.*

The action attribute of the `<form>` tag is the MembershipForm.htm page. That is, information submitted from the form is made available to this page when it is transmitted to the server. The method is post, meaning that form name/value pairs are sent as a separate data stream to the URL specified in the actionattribute. The entire set of name/value pairs for the filled-out form would resemble the following:

```
FirstName=Alice&Lastname=Underwood&Email=a_underwood@h
Gender=F&Major=Web+Development&Reason=To+meet+new+frie
+Web+pages&SubmitButton=Submit+Form
```

Note that blank spaces in the string are replaced by the plus sign (+).

If the get method is used, the entire set of name/value pairs for the filled-out form would be appended as a query string and passed to the URL specified in the actionattribute as shown below:

```
MembershipForm.html?FirstName=Alice&Lastname=Underwood
Gender=F&Major=Web+Development&Reason=To+meet+new+frie
+Web+pages&SubmitButton=Submit+Form
```

When this input string is received at the server, it is broken down into its component parts so that individual names and values are associated:

```
FirstName=Alice
LastName=Underwood
Email=a_underwood@hotmail.com
Gender=F
Major=Web Development
Reason=To meet new friends and make Web pages
```

The manner in which this information is processed by the receiving page depends on the script contained on that page.

Form Submission - TOP

Form submission data is typically processed by a client-side script, server-side script, or a combination of both. Client-side scripts are most commonly written in JavaScript. Since JavaScript is built into the browser, processing takes place at the browser or client-side without having the data passed to a web server for processing. Client-side scripts are faster and generally used to validate form input data.

Server-side scripting is a technology by which a script is run on a web server to dynamically generate and process web pages and form input. Common server-side scripting technologies include PHP, Ruby on Rails, Microsoft's ASP.NET, Adobe Coldfusion, and Java. Since server-side scripts are run on the web-server, these scripts have the ability to interactive with databases and file handling.

In most web applications, form data is first processed by a client-side script for the purpose of data validation and page customization. Next, the data is passed on to a server-side script where another level of data validation occurs followed by the data being written to a database.

11.11: Submitting Forms is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

11.12: Forms and CSS

Prior to CSS, tables were used to control the layout of form control elements. In fact it is not uncommon today to find form controls organized on a page using tables.However, the table approach is outdated and does not provide support for users with accessibility issues. An alternative and modern approach is to use CSS to format and display form controls. The advantage to using CSS is that it requires less code - elimination of table tags and the line break tag. The code below demonstrates how to layout form controls using CSS.

```
 <style>

form {padding:10px}
label {width: 120px; float:left; text-align:right;
        margin-top: 10px; padding-right:10px;clear:lef
input {margin-top: 10px;display:block}

#mySubmit {margin-top:10px;padding-bottom:10px}

 </style>

-----------------------------------------------

 <div>

     <form>

     <label for="FName"> First Name: </label>
     <input type="text" id="FName">
     <br>
     <label for="LName"> Last Name: </label>
     <input type="text" id="LName">

     <input type="submit" value="Submit Form">

     </form>

 </div>
```

First Name:

Last Name:

Submit Form

Using the CSS method, label element selectors are key to aligning the text - this selector is configured to on the left with the text aligned to the right. The float causes the input controls to wrap around the right side of the labels. The input controls are also configured with a top margin. The submit button is styled using the "mySubmit" id selector. Left margin and bottom padding are applied to the submit button.

A more complex form is shown below. Notice that the same CSS can be used to control the layout of the form.

First Name:

Last Name:

State:

Georgia

Male

Female

Submit Form

```
<style>

 form {padding:10px}
 label {width: 120px; float:left; text-align:right;
        margin-top: 10px; padding-right:10px;clear:le
 input {margin-top: 10px;display:block}

 #mySubmit {margin-top:10px;padding-bottom:10px}

  </style>

-----------------------------------------------

<div>

<form>

<label for="FName"> First Name: </label>
```

```
    <input type="text" id="FName">

    <br>

    <label for="LName"> Last Name: </label>
    <input type="text" id="LName">

    <br>

    <label id="LState" for="State"> State: </label>

    <select id="State">
    <option value="GA"> Georgia </option>
    <option value="FL"> Florida </option>
    <option value="SC"> South Carolina </option>
    </select>

    <br>

    <label for="male"> Male </lable>
    <input type="radio" name="gender" id="male" value="ma
    <br>
    <label for="female" id="lfemale"> Female </label>
    <input type="radio" name="gender" id="female" value="

    <br>

    <input type="submit" value="Submit Form">

    </form>

    </div>
```

This demonstrates the flexibility of using CSS to control the layout of a form control. Regardless of the number of controls and labels, the same CSS can be used to organize the form controls with the use of a table or excessive line breaks.

In addition to the layout of the form, CSS can be used to enhance the overall look and usability of the form. The examples below illustrate how CSS is used to first organize the layout of the form, and then enhance the look and usability.

First, we begin with the basic form layout as described in the previous sections.

Name:

Address:

City:

State:

Georgia

Zip:

Submit Form

Next, we add a fieldset and legend tag to group the related form control elements and include a caption. The entire fieldset is placed inside of a div element for further styling.

Mailing Address

Name:

Address:

City:

State:

Georgia

Zip:

Submit Form

```
<form>

<div id="fieldset">
<fieldset>
<legend> Mailing Address </legend>

<label for="FName"> First Name: </label>
<input type="text" id="FName">
```

```
    <label for="LName"> Last Name: </label>
    <input type="text" id="LName">


    <label id="LState" for="State"> State: </label>

    <select id="State">
    <option value="GA"> Georgia </option>
    <option value="FL"> Florida </option>
    <option value="SC"> South Carolina </option>
    </select>

    <label for="male"> Male </label>
    <input type="radio" name="gender" id="male" value="ma
    <br>
    <label for="female" id="lfemale"> Female </label>
    <input type="radio" name="gender" id="female" value="

    <input type="submit" value="Submit Form">
    </fieldset>

    </div>
    </form>
```

First, we edit the CSS to style the `<div>`. The `<div>` is styled to include a background color, width, and a thin border.

```
<style>
   div#fieldset {background-color:tan;width:500px;border:
</style>
```

Mailing Address

Name:

Address:

City:

State:

Georgia

Zip:

Submit Form

Next, the legend text is styled, and a background color is applied

```
legend {font-size:20px;background-color:green;color:wh
```

Mailing Address

Name:

Address:

City:

State:

Georgia

Zip:

Submit Form

Padding is added to increase the amount of space surrounding the legend text.

```
legend {font-size:20px;background-color:green;color:wh
```

Mailing Address

Name:

Address:

City:

State:

Georgia

Zip:

Submit Form

Notice that the default fieldset border is still visible. We can remove this by setting the border width to 0.

```
fieldset {border:0}
```

Mailing Address

Name:

Address:

City:

State:

Georgia

Zip:

Submit Form

Finally, we adjust the fieldset margin settings so that the legend aligns properly.

```
fieldset {border:0;margin: 0 0 0 -10px}
```

11.12: Forms and CSS is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

12: HTML/CSS Appendix

12.1: New HTML5 Tags

12.2: HTML Reference

12.3: CSS Reference

12.4: HTML Special Characters

12.5: HTML Colors

12: HTML/CSS Appendix is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

12.1: New HTML5 Tags

12.1: New HTML5 Tags is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

12.2: HTML Reference

12.2: HTML Reference is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

12.3: CSS Reference

12.3: CSS Reference is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

12.4: HTML Special Characters

12.4: HTML Special Characters is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

12.5: HTML Colors

12.5: HTML Colors is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by LibreTexts.

Index

Glossary

Sample Word 1 | Sample Definition 1

Detailed Licensing

Overview

Title: Web Site Design - HTML (2022)

Webpages: 97

All licenses found:

- CC BY-SA 4.0: 99% (96 pages)
- Undeclared: 1% (1 page)

By Page

- Web Site Design - HTML (2022) - *CC BY-SA 4.0*
  - Front Matter - *CC BY-SA 4.0*
    - TitlePage - *CC BY-SA 4.0*
    - InfoPage - *CC BY-SA 4.0*
    - Table of Contents - *Undeclared*
    - Licensing - *CC BY-SA 4.0*
  - 1: Creating Web Pages - *CC BY-SA 4.0*
    - 1.1: Internet History and Usage - *CC BY-SA 4.0*
    - 1.2: Serving Web Pages - *CC BY-SA 4.0*
    - 1.3: HTML Tags and CSS Styles - *CC BY-SA 4.0*
    - 1.4: HTML Tags and CSS Styles - *CC BY-SA 4.0*
    - 1.5: Creating Your First Web Page - *CC BY-SA 4.0*
  - 2: Web Site Design - *CC BY-SA 4.0*
    - 2.1: Goal identification - *CC BY-SA 4.0*
    - 2.2: Design Priniciples - *CC BY-SA 4.0*
    - 2.3: Choosing Colors - *CC BY-SA 4.0*
    - 2.4: Typography - *CC BY-SA 4.0*
    - 2.5: Accessibility - *CC BY-SA 4.0*
    - 2.6: Web Site Development - *CC BY-SA 4.0*
    - 2.7: Mobile Web Development - *CC BY-SA 4.0*
  - 3: Basic Document Layout - *CC BY-SA 4.0*
    - 3.1: Structuring Page Content - *CC BY-SA 4.0*
    - 3.2: Document Structure Tags - *CC BY-SA 4.0*
    - 3.3: Paragraphs - *CC BY-SA 4.0*
    - 3.4: Line Breaks - *CC BY-SA 4.0*
    - 3.5: Headings - *CC BY-SA 4.0*
    - 3.6: Horizontal Rules - *CC BY-SA 4.0*
    - 3.7: List Structures - *CC BY-SA 4.0*
    - 3.8: Linking Pages - *CC BY-SA 4.0*
    - 3.9: Displaying Pictures - *CC BY-SA 4.0*
  - 4: Basic Document Styling - *CC BY-SA 4.0*
    - 4.1: Applying Styles - *CC BY-SA 4.0*
    - 4.2: Margin Styles - *CC BY-SA 4.0*
    - 4.3: Alignment Styles - *CC BY-SA 4.0*
    - 4.4: Width and Height Styles - *CC BY-SA 4.0*
    - 4.5: Basic Color Styles - *CC BY-SA 4.0*
    - 4.6: Basic Color Styles - *CC BY-SA 4.0*
    - 4.7: ID and Class Selectors - *CC BY-SA 4.0*
  - 5: Text Formatting - *CC BY-SA 4.0*
    - 5.1: Font Settings - *CC BY-SA 4.0*
    - 5.2: Text Properties - *CC BY-SA 4.0*
    - 5.3: Font Style Tags - *CC BY-SA 4.0*
    - 5.4: Text and Background Colors - *CC BY-SA 4.0*
    - 5.5: Special Characters - *CC BY-SA 4.0*
  - 6: Graphic Images - *CC BY-SA 4.0*
    - 6.1: Graphic Formats - *CC BY-SA 4.0*
    - 6.2: Displaying Images - *CC BY-SA 4.0*
    - 6.3: Background Images - *CC BY-SA 4.0*
    - 6.4: Image Transparency - *CC BY-SA 4.0*
  - 7: Applying Special Styles - *CC BY-SA 4.0*
    - 7.1: Styling Lists - *CC BY-SA 4.0*
    - 7.2: Borders and Padding - *CC BY-SA 4.0*
    - 7.3: Sizing Elements - *CC BY-SA 4.0*
    - 7.4: Positioning Elements - *CC BY-SA 4.0*
  - 8: Linking Pages - *CC BY-SA 4.0*
    - 8.1: Text Links - *CC BY-SA 4.0*
    - 8.2: Graphic Links - *CC BY-SA 4.0*
    - 8.3: Image Maps - *CC BY-SA 4.0*
    - 8.4: Redirecting Pages - *CC BY-SA 4.0*
    - 8.5: Traversing Folders - *CC BY-SA 4.0*
  - 9: Using Tables - *CC BY-SA 4.0*
    - 9.1: Creating Tables - *CC BY-SA 4.0*
    - 9.2: Table Borders - *CC BY-SA 4.0*
    - 9.3: Table Captioning - *CC BY-SA 4.0*
    - 9.4: Table Captioning - *CC BY-SA 4.0*
    - 9.5: Table Size and Alignment - *CC BY-SA 4.0*
    - 9.6: Aligning Text and Cells - *CC BY-SA 4.0*
    - 9.7: Column and Row Spanning - *CC BY-SA 4.0*
    - 9.8: Table Groupings - *CC BY-SA 4.0*
  - 10: Playing Multimedia - *CC BY-SA 4.0*
    - 10.1: Audio and Video Formats - *CC BY-SA 4.0*

- 10.2: Playing Audio - *CC BY-SA 4.0*
- 10.3: Playing Video - *CC BY-SA 4.0*
- 10.4: Third Party Content - *CC BY-SA 4.0*

- 11: Creating Forms - *CC BY-SA 4.0*
  - 11.1: Form Controls - *CC BY-SA 4.0*
  - 11.2: Creating Forms - *CC BY-SA 4.0*
  - 11.3: Textbox Controls - *CC BY-SA 4.0*
  - 11.4: Textarea Control - *CC BY-SA 4.0*
  - 11.5: Radio Buttons - *CC BY-SA 4.0*
  - 11.6: Checkboxes - *CC BY-SA 4.0*
  - 11.7: Drop-Down Lists - *CC BY-SA 4.0*
  - 11.8: Submit and Reset Buttons - *CC BY-SA 4.0*
  - 11.9: Other Buttons - *CC BY-SA 4.0*
  - 11.10: Grouping and Labels - *CC BY-SA 4.0*
  - 11.11: Submitting Forms - *CC BY-SA 4.0*
  - 11.12: Forms and CSS - *CC BY-SA 4.0*
- 12: HTML/CSS Appendix - *CC BY-SA 4.0*
  - 12.1: New HTML5 Tags - *CC BY-SA 4.0*
  - 12.2: HTML Reference - *CC BY-SA 4.0*
  - 12.3: CSS Reference - *CC BY-SA 4.0*
  - 12.4: HTML Special Characters - *CC BY-SA 4.0*
  - 12.5: HTML Colors - *CC BY-SA 4.0*
- Back Matter - *CC BY-SA 4.0*
  - Index - *CC BY-SA 4.0*
  - Glossary - *CC BY-SA 4.0*
  - Detailed Licensing - *CC BY-SA 4.0*