**Creating Pages in HTML**

In this lesson the basics of HTML coding will be developed. An emphasis is placed on document content and structure. This can be completed entirely with a web browser and text/HTML editor.

At this point in the course it is not uncommon to have ideas about what web pages should look like. However, in most real world web pages, what pages "look like" is secondary. A more important first question is: "What do I want my web page to say?" This course places an emphasis on content first, presentation second. The strength of this approach to web design is that it lends itself to designing web pages that are robust, with content that is flexible and deliverable to users regardless of which web-enabled technologies or configurations they're using. The creed among web developers who practice this approach, and among standards organizations who promote it, is that content and structure should be separate from presentation.

A website that will serve as a portfolio for displaying the work will be created during this lesson.

**Pre-Coding**

Pre-coding steps are helpful to go through before beginning to work with the nuts and bolts of web page development.

Just as there are pre-writing steps that ought to be done prior to writing an essay, there are pre-coding steps to do before coding a website. It is important to plan ahead in order to reduce the number of mistakes made while constructing a web page/site. This will save time, and in the working world will save money.

The heart of this design philosophy is the notion that content and structure must be kept separate from presentation. When this is true, the content and structure of a web page can be delivered to all sorts of web-enabled devices, not confined by browser, operating system, screen size, or type of device. The same content can be presented in different ways for these different devices and configurations, but it's still the same content.

*Pre-Coding: Instructor*

### Tips for Delivering This Lesson:

* Students are always eager to get started constructing their website. The tendency for most students is to skip preliminary planning and start coding a site from the onset. Emphasize that initially in the process of developing a website the developer should spend plenty of time communicating with the client to understand what the client’s needs are. Using paper to develop a preliminary sketch and file diagram reinforces the notion that planning ought to be done first.
* When students are ready to set up their folder structure, it can be helpful for students to watch the instructor step through the process. Proceed slowly and emphasize the naming protocol for folders and files that are created in this activity, since these folders and files will be referenced extensively throughout the curriculum, especially when linking to content between files. This is especially critical if your class is structured in a way that encourages students to begin working at their own pace.
* Some students may have grand ideas and may sketch a complex site and design that is beyond the scope of this course. Be honest with these students that their designs may require advanced skills beyond those taught in the course, but if these students prove throughout the course to have the aptitude to learn the advanced skills required, their original sketch can be used to motivate them. Perhaps by the end of the course they will have successfully realized their vision.

**Student Output:**

When this lesson is complete, students will have created a blank home page and three sub-folders.

*Pre-Coding: Student*

## Activities:

The following are the initial pre-coding steps to take in preparing to design your web portfolio site.

1. **Consider the site's purpose.**

If you are clear about the site's purpose (to sell, to inform, to entertain, etc.) and who you are communicating to, you are much more likely to create a better site. For this web portfolio the purpose is simply to inform an audience about what skills you are developing. At first, the audience may only be your instructor, but you may want make it more general so that it could be modified into a resume, in which case the audience would include prospective employers.

1. **Sketch the homepage**.

Before sitting down to a computer it is sensible to make a sketch on paper of the content of your website. Start with the home page, since the home page serves as the portal through which most other site content is accessed. This should be an outline of content only - remember that at this point in the design process we're not so interested in how the page will appear visually. That comes later. In your sketch be sure to include at least each of these essential elements:

* A main heading that reads "Web Portfolio"
* A block of text containing information about you. At a minimum this should include your name and email address.
* A block of text containing information about the course. This might include information such as the name of the course, teacher, section, school, and year.
* A paragraph that provides an overview of the content of your portfolio.
* A sub-heading, which you will enter an overview of what your web page is about.

1. **Diagram the site's page and folder structure**.

Websites are a collection of linked files. Files need to be organized into a system of folders, especially if you are building a complex site. A common way of organizing files for a website is to have the home page and all secondary pages in the main folder, then create subfolders for images, styles, and scripts. If it were a larger site, you would additionally create subfolders for content. One way of organizing content would be to use a folder system that corresponds with the structure of the site as users experience it. That way, if you hand the site to a client or have others assisting you in the construction of the site, they'll be able to easily locate the files they need.

1. **Set up the folder structure for containing files that make up your portfolio site**.

Open your file management program. Make a new folder named "**Portfolio\_YourName**". This main folder, also known as a parent folder or root folder will hold all the files that you create in this class. Note that there should not be any spaces in file names or folder names. The underscore is a common convention for indicating separations between words in a folder or file name. Servers may be case sensitive so be consistent about capitalization. Most of your web pages will be stored in this folder. However, in later units you will begin to add images and other content to your web pages that require the use of supplemental files. For organizational purposes it's a good idea to store these supplemental files in separate subfolders. For your portfolio, create the following subfolders inside your main folder:

* images
* styles
* scripts

1. **Create a new file for the home page**.

Open your text editor program. Create a new file, and save the file in the root folder as "**index.html**". For now, this is just a blank file but you'll add content to it in the next lesson.

### About the filename index.html

When users request a web address (URL) that does not include a filename (for example, www.somedomain.com) the server automatically looks at that address for a file named index.html. If it finds that file, that's the file it sends to the browser. If index.html is not available, the server has a short list of other filenames it will try (e.g., index.htm, index.shtml, default.htm, etc.). These vary by server. The .htm extension on web pages came about in the early days of the Web, when developers using Microsoft Windows were limited to file extensions with three characters. However, these days it is more common for HTML web pages to end in .html.

**Basic HTML Markup**

In this module, the basics of HTML coding are taught.

* HTML tags, including opening and closing tags, and learn the distinction between elements and attributes
* Essential tags for a website including tags that comprise the two main sections of a web page, the <head> and <body>
* Create a "bare bones" template and use this to create a home page and a few sub-pages for their portfolio
* Introduction to some of the most common HTML elements

HTML stands for *HyperText Markup Language*. HTML is the language that has historically been used to create documents on the web. It is plain text, but includes a variety of codes or "tags" that define the structure of the document, and allow documents to include headings, paragraphs, images, links, lists, tables, and other features.

HTML has undergone various revisions over the years. Some tags have become deprecated, or old-fashioned, when new versions of HTML are developed. The latest version of HTML, and the version taught in the course, is HTML5 .

This will lay the foundation for the portfolio home page.

*HTML Syntax: Instructor*

### Tips for Delivering This Lesson

* This lesson includes a few questions for assessing whether students are understanding some of the basic concepts concerning HTML
* The second of these questions, "What are the two main sections of an HTML document?" seems simple enough, but it is worth emphasizing as it pertains to the most basic structure of a web document. The fact that there is only one head and one body to a document sometimes eludes students and they end up using multiple body tags in a document, or overlapping these elements rather than opening and closing them in proper sequence. It is important that they fully understand this basic structure, and the notion that most HTML elements are container elements that have other elements inside them. Without this basic understanding, students will struggle to learn HTML as more and more elements are introduced.
* Despite the previous point, since this curriculum teaches HTML5 it should be noted that technically <head> and <body> are no longer required. In fact, many of the rules that are taught in this course (e.g., the importance of closing elements after they've been opened) are no longer requirements—they're all optional. HTML5 is much more relaxed than previous versions of HTML, especially XHTML, which was extremely strict about rules not being violated. Despite this fact, students should learn the rules and be encouraged to follow them, simply because they're a good practice, and because most web pages in use today still use earlier versions of HTML. Whether you choose to share this knowledge about HTML5 or keep it secret is up to you.
* Some students are confused by the <title> element and how that differs from heading elements such as <h1>. Another source of confusion is the difference between head, heading (e.g., <h1>), and header (e.g., <th> in tables). It may be tempting to explain the differences in these terms in this lesson, but we recommend waiting until students need to understand the differences. That way they can immediately apply the knowledge to their web pages, which may make it seem less abstract.

*HTML Syntax: Student*

**Activities**

1. Look over the handout Common HTML Tags and carefully examine the common elements.
2. Referring to the Common Tags sheet, record your responses on paper to the following questions:
3. Which pair of tags marks the start and end of your HTML document?
4. What are the two main sections of an HTML document?
5. Which element serves as a container for all the visible content of a web page?
6. Which element is used to identify the main heading on a page?
7. Which element is used to add an image to a page?

*Essential Tags: Instructor*

### Tips for Delivering This Lesson

* As students begin coding, it's important to reinforce good coding habits (such as using lower case for HTML elements, and indentating code to make it more readable). Having clean, easy-to-read code will help tremendously as students' web pages become increasingly complex. Also good coding habits will help students throughout their lives if they ultimately pursue additional education in computing, especially if they learn computer programming languages.
* Be sure that students successfully complete all steps in the assigned activity. Future lessons depend on all files being created, properly named, and initially populated with a bare bones template. Consider having students pair and check each others' work.

Give them this simple list to verify each other’s work:

* 1. Are there six pages in the parent portfolio folder?
  2. Does each page have a distinct title that reflects the content of the page?
  3. Are the essential tags on each page?

### Example Output

When this lesson is complete, students will have have created five new HTML files (accessibility.html, graphics.html, javascript.html, tools.html, and usability.html). For now, each file is populated solely with a bare bones HTML template, just like the index.html file. The index.html file currently has no visible content in the body.

*Essential Tags: Student*

**Activities**

1. Open a text editor program and navigate to the "portfolio" folder.
2. Type the following markup into your file. This is the basic HTML structure of a web page. Note that you'll personalize the highlighted text contained in the title tags.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Web Portfolio: Your Name</title>

</head>

<body>

<!-- The content of your web page will go here -->

</body>

</html>

Tips for Coding

* Add extra blank lines
* Indenting content that is inside of other content, so you can see the relationship between all the parts of the page

**Remember:** Extra spaces and blank lines will be ignored when the HTML is displayed by a browser.

Let's now examine each of the tags:

* The first line is the **DOCTYPE**. It specifies the version of HTML you are using. HTML5 has a very simple DOCTYPE. All prior versions of HTML and XHTML had much longer and complicated DOCTYPE statements, but they can easily be found by searching the web, and can be copied and pasted into your web page. The DOCTYPE statement is necessary because it tells the browser which version of HTML you're using, so the browser knows how to process your code. . A common mistake among web developers is neglecting to include a DOCTYPE statement. Without a DOCTYPE statement, browsers have to guess which version of HTML you're using, and sometimes they get it wrong.
* **<html>** is typed before all the text in the document. This marks the beginning of the html document. It has a corresponding **</html>** tag that marks the end of the document. The entire web page, other than the DOCTYPE statement, is wrapped between these two tags.
* **<head>** Web pages are divided into two main sections: the *head* and the *body*. The *head* provides information about the document, including the author, description, keywords, title, and other information. The *head* section is closed with the **</head>** tag. In our "bare bones" document there are only two elements inside the *head*. They are:
* **<title>** You must give your document a title. This title doesn't actually appear within the web page, but appears in the title bar of the browser window. This is also the title of the page that will be displayed by default in search engine results or in user's Favorites. The title is closed with **</title>**
* **<meta>** is a tag that has many purposes, depending on what attribute it has. In our "bare bones" document, the attribute is *charset*, which is set to "utf-8". This is a required tag, which tells the browser which character set the web page is encoded in. There are many possible character sets, but "utf-8" is an international character set that is one of the most common. The <meta> tag is not a container tag. Therefore, it has no corresponding closing tag.
* **<body>** The body section contains the contents of your document
* **<!-- Comment -->** Comments are intended solely for people reading the source code, and are not visible when someone views the web page in a browser.

1. Save the index.html file. Now open this file in your browser. You will notice that the screen is blank. This is because you don't have any content yet in your body section. However, you should see your title displayed in browser's title bar, usually across the top of the browser window.
2. Return to the text editor and the index.html file. While you're creating files using the "bare bones" template, you should go ahead and create the other files that will comprise your website. Later in this course, you'll be adding content to each of these pages, but for now they'll be blank, just like the home page. Simply copy the code from index.html and paste it into the new pages. Each time you do this, change the <title> element to reflect the content of the new page. For example, change the title in graphics.html to something like "Web Portfolio: Your Name's Graphics Page".   
     
   Save each new file in your root folder with the following file names:

* accessibility.html
* graphics.html
* javascript.html
* usability.html
* tools.html

*Common Tags: Instructor*

### Tips for Delivering This Lesson

* Students will very likely begin to move at different paces in this lesson. The instructions are easy to follow if students are proficient readers. However, students for various reasons may begin to struggle with the content. This is an opportunity for instructors to spot the students who may require extra help and encouragement.
* This may also be a good time to introduce supplemental texts or websites to support student learning of HTML code.
* To foster independence and problem solving skills, it may be helpful to introduce a help protocol whereby students are encouraged to first reread directions, second consult texts or tutorial sites, third ask peers and lastly seek the instructor's help.
* This lesson introduces a simple sequence that will be used throughout the curriculum:
  1. Open a web page in both the HTML editor and web browser
  2. Save the page in the HTML editor
  3. Switch to the web browser and refresh
  4. Repeat steps 2 and 3

Since this will be such a frequently used sequence, you may want to step through it with students and spend some time practicing. Also, it is helpful to teach students shortcuts such as:

* 1. Ctrl + S (Windows) to Save
  2. F5 (Windows) to Refresh
  3. Alt + Tab (Windows) to switch between open software applications
* This lesson requires a great deal of coding for a neophyte coder. Students may benefit from viewing a model of how the page should look

### Example Output

When this lesson is complete, students will have added basic HTML content to the index.html file in their portfolio.

*Common Tags: Student*

**Learner Outcomes**

At the completion of this exercise:

* you will be able to use some common tags for adding content to a web page including <h1>, <h2>, <h3>, <p>, and <div>
* You will be able to apply the concepts of nesting and assigning attributes to tags.
* You will be able to properly utilize headings and paragraphs to structure content on a web page

**Activities**

1. Return to index.html in the text editor.
2. Review the outline of your home page that you sketched.
3. The outline should include:
   * A main heading that reads "Web Portfolio"
   * A block of text containing information about you. At a minimum this should include your name and email address.
   * A block of text containing information about the course. This might include information such as the name of the course, teacher, section, school, and year.
   * A paragraph that provides an overview of the content of your portfolio.
   * A sub-heading for each of the remaining units of this curriculum (Unit 2 through 7).
   * Some placeholder content beneath each sub-heading (something like "This section will be completed soon.")
4. Now your task is to translate that outline into HTML code. To do so, consider each piece of content, and ask "What is this?" Try to answer that question with an HTML tag. Is it a heading? Is it a subheading? Is it a paragraph? The more your work with HTML, the easier it will be to figure out which tags are most appropriate for a given piece of content.
5. Begin with the main heading of the page, "Web Portfolio". What is that? It's the main heading of the page. In HTML terms, it's an **<h1>**. To mark up your heading as an **<h1>**, simply place the cursor on a blank line directly below the body tag, **<body>**, and enter this:

**<h1>**Web Portfolio**</h1>**.  
  
The **<h1>** tag marks the main heading within an HTML document. Browsers typically display this heading in a larger font than all other headings and text, although its appearance can be changed using CSS (more on that in Unit 3).

When finished, save, then refresh your browser to see your work. If you did everything correctly, you should see your heading at the top of your web page.

1. Next, markup the two blocks of content, the one with information about the class and the one with information about you. What are these? They're not headings. They're not paragraphs. In fact, they're just blocks of related text for which there really isn't a direct matching HTML tag. For blocks of related content, the best choice is the **<div>** element. So, wrap each of your two blocks in **<div>** and **</div>**. Since each of these blocks of text contains more than one item, we'll probably want each item to appear on a line by itself, so we'll want to use a **<br/>** tag to force a line break. Here's an example:
2. <div>
3. Judy Jetson<br/>
4. judy@ohs.edu
5. </div>

Make one **<div>** for each of your blocks of text. Then, refresh your browser and check.

1. The remaining content from your sketch included an overview paragraph, a sub-heading for each unit 2 through 7, and some text beneath each sub-heading to serve as a placeholder until you're ready to add more content. Since each unit is a sub-heading, the appropriate HTML markup would be **<h2></h2>**. And the text that follows each heading is a paragraph (paragraphs don't have to be more than one sentence), so that should be marked up with **<p></p>**. Here's what the markup for Unit 2 would look like:

**<h2>**Unit 2**</h2>**  
**<p>**This section will include links to my work for this unit, once completed.**</p>**

After you have added the Unit 2 content to your web page, save and refresh your browser to check your work. If all is well, copy and paste this same content for the remaining units (but remember to change the Unit number after pasting).

1. Finally, take all the content you added in the previous step (everything from the overview paragraph down to the final paragraph under Unit 7) and wrap it in a <div> element. The opening <div> tag should precede the overview paragraph, and the closing </div> should be at the bottom of your document, just before the closing </body> tag. The purpose of this div is to provide a container for all the main content of the page. This isn't required in a web page, but it will come in handy later (in Unit 3) when we start to stylize the page using CSS.

**HTML Lists**

This module will introduce you to HTML lists. You will work through two lessons that will help you to understand how lists are used on the Web, and will guide you through creating your own lists on your portfolio home page. The list that you create in Lesson 2 will serve as the navigation menu for your website.

Often when we consider the content of our web pages and ask "What is this?" the answer is: It's a list! News sites feature lists of today's top stories, sometimes organized by category. Music sites feature lists of artists or songs. Video sites feature lists of videos. Sports sights include lists of teams, lists of scores, and rankings.

If it's a list, it should be marked up as a list. There are two common types of HTML lists, those in which the order matters, and those in which it doesn't. If the order matters (as in sports rankings), we use an **ordered list**, which in HTML is created with the **<ol>** element. By default, ordered lists are displayed by browsers with each list item numbered sequentially.

If the order doesn't matter, we use an **unordered list**, which in HTML is created with the **<ul>** element. By default, unordered lists are displayed by browsers with a bullet in front of each list item .

Although lists typically show up with either numbers or bullets, they don't always. Web authors have control over this with cascading style sheets (CSS). You'll learn more about this in Unit 3.

Regardless of whether a list is ordered or unordered, the items in an HTML list are marked up with the <li> element (li stands for *list item*). These are container tags, so each item in the list must have an opening tag (<li>) and a closing tag (</li>)

In this lesson you will review web pages, looking for ordered and unordered lists. Then, you will create an unordered list of your own for organizing some of the content on your portfolio's home page.

In the previous lesson you learned about ordered and unordered lists. One very common type of list that appears on web pages is a navigation menu. What is a navigation menu? It's a list of links. Therefore, it should be coded as such. In this lesson, you create an unordered list of links that will serve as the navigation menu for your web portfolio.

*Lists: Instructor*

### Tips for Delivering This Lesson

* This lesson provides a good opportunity to stress that most tags in HTML are container tags, and be sure that students understand this concept. All lists are containers. They have opening and closing tags. Inside these containers, there are multiple list items. List items are also containers: They contain text. It's also important for students to understand that some element are dependent on others. For example, even in HTML5, which is generally pretty forgiving, it is not permissible to have a list item that isn't part of a list.
* This lesson also provides another good opportunity to stress the importance of using indentation in the code. Indenting <li> tags slightly more than <ol> tags makes it easy to see which elements are components of which other elements.

### Example Output

When this lesson is complete, students will have modified their index.html file by changing some content to unordered lists.

Lists: Student

**Activities**

1. Browse the web, and try to find at least three ordered lists and three unordered lists. You don't need to view the source code for this activity - just try to find content that appears to be each of these types of lists. Remember that some lists won't necessarily show up with numbers or bullets, since these can be turned off or changed using CSS. Show your instructor what you found.
2. Now return to index.html in your portfolio website. Previously, you added content to this page, which included two <div> elements that contain blocks of content, one with information about you, and one with information about your web design course. In that lesson you asked "What is this?" but that was probably before you started thinking about how many lists there are on the Web. Now when you ask that question related to the content inside your two <div> elements, you'll probably answer: It's a list! And indeed, you would be right! The first div includes a list of information about you (e.g., your name, your email, etc.). The second div includes a list of information about your school. Are these lists ordered or unordered?
3. If you answered "unordered" to the previous question, you're right again!
4. Go ahead and modify the content of each of the two div elements so the content is marked up as an unordered list. You will no longer need the <br/> elements since each list item will automatically appear on a new line.

Creating a Navigation Menu: Instructor

### Tips for Delivering This Lesson

* Students may have a hard time making the connection between the list of words they're creating and a navigation menu like those they typically see on websites. This may be an opportunity to reinforce or revisit concepts taught during Basic HTML Markup: Common Tags, specifically the strategy of asking "what is this?" in order to decide which HTML element best represents the content. A navigation menu is a list of web pages. Therefore, it should be coded as a list.
* This lesson also provides a good opportunity to remind students of the difference between HTML and CSS. In the current unit, students are simply creating content and structure. In the next unit, they'll learn about CSS, which will enable them to transform their list of web pages so that it looks more like they expect a navigation menu to look.

### Example Output

When this lesson is complete, students will have added an unordered list of pages to their index.html file, which ultimately will become the navigation menu.

Creating a Navigation Menu: Student

**Learner Outcomes**

At the completion of this exercise:

* you will be able to create a navigation menu using an unordered list.

**Activities**

1. Return to index.html in your portfolio website. Add a new div element. Place it **after** the div element that contains school course information and **before** the div element that contains the series of headings and paragraphs you created for each unit. When you create the new div element, be sure to include its closing </div> tag.
2. Inside this new div, create another unordered list. This one should contain the following items (each item represents a separate web page that you will be creating in your portfolio):
   * Home
   * Accessibility
   * Usability
   * Graphics
   * Javascript
   * Tools
3. Save, and refresh your page in the browser to check your work. Does the new content appear as a bulleted list? If so, congratulations! You've created a navigation menu. Ok, so it doesn't look like, or function like, a navigation menu yet. That will come later. In the next module, you'll learn to create links so each of the items menu can be turned into a link to the other pages of your portfolio site. Finally, in Unit 3, you'll learn how to stylize your list using CSS so that it looks more like you expect a navigation menu to look.

**Creating Links**

This lesson will introduce students to techniques for creating links on their web pages. Students will learn how to create links using both absolute and relative paths and be able to differentiate between the two. In addition, students will explore using named anchors to create jump-to links on a page, and will learn to create e-mail links.

This lesson will introduce you to techniques for creating links on your web pages. You will learn how to create links using both absolute and relative paths, and be able to differentiate between the two. In addition, you will explore using named anchors to create jump-to links on a page, and will learn to create e-mail links.

The World Wide Web was built on the principal of *hypertext*. Prior to hypertext, documents were all standalone. They might refer to each other in text, but there was no direct connection between one document and another. With hypertext, readers were suddenly able to quickly jump from one document to another, which revolutionized the way we access information.

Links to web pages refer to the address, or **URL** (Uniform Resource Locator), of the web page. URL's consist of various parts. For example, consider the following URL, which is the home page for this course curriculum:

**http://www.washington.edu/accessit/webd2/student/index.html**

This URL consists of four parts, separated by forward slash (/):

* **http://** - This is the *Internet protocol*, and tells the browser how to connect with the server hosting the URL. Most documents on the web begin with http://, but they might also begin with https://, ftp://, telnet://, or others.
* **www.washington.edu** - Domain name where the file is located.
* **/accessit/webd2/student/** - folder or directory where the file is located (in this example, the file index.html is stored in a subdirectory three levels deep)
* **index.html** - the filename

The file name index.html could optionally be omitted from the above example. When the server discovers that *student* is a directory rather than a file name, it would check the *student* directory for one of the standard home page file names, and in this case would find index.html.

**Linking to a URL**

Links are inserted into a document using the **<a>** tag which stands for "anchor". This element by itself does nothing. Consider this HTML:

<a>This is a link to nothing</a>

At a minimum, the <a> elements requires an href attribute, which defines the destination of the link.

When you link to external websites, you use an absolute path, as in the following example:

<a href ="http://www.somedomain.com/somedirectory/somefile.html">This is the text that users click on</a>

Note that the destination path (contained within quotes) gives the full address to the target page, starting with "http://". If you were to copy all the characters between the quotation marks and insert them into the address bar of a browser, the browser would open the website. This type of URL is referred to as an **absolute URL**. In the next lesson, you will learn about its opposite, the **relative URL**.

In the previous lesson you learned that a web address (URL) consists of various parts. You also created a link in your portfolio home page to your school's home page. This link used an **absolute** address (the full address to the school website). Now you will learn to make links to pages within your own website, using a **relative address**.

A relative address is one that refers only to a portion of the web address, rather than to the full address. For example, assume you are webmaster of the domain mydomain.com, and want to add a link from http://www.mydomain.com/file1.html to http://www.mydomain.com/file2.html. In file1.html, you don't need to link to the full address. You can simply link to the file name (file2.html), since both files are located in the same place.

A relative address is an address that is *relative* to the location of the linking file.

In this lesson you create two types of links that make web surfing even more convenient: a jump-to link and an email link. Jump-to links are typically used on very long pages, where users can benefit from having a link at the top of the page that allows them to jump to a certain section of the page without scrolling.

This is particularly useful for some users with disabilities, who need to bypass the navigational links on a page. For example, suppose a user who is physically unable to use a mouse is trying to access a link on a website, but that link is in the main content of the page and there are dozens of navigational links above it. Using the tab key, it will take forever for this user to get to the link they're wanting to select. As the web designer, you can help by adding a same-page "skip to main content" link at the top of the page that jumps directly to the main content, bypassing the navigation menu.

In this lesson, you will add a skip navigation link to your portfolio home page, thereby allowing users to skip directly to the main content of your website.

You will also add an email link, which if clicked opens a blank, pre-addressed email message for users whose browsers and email software support it. Up to this point your email address on your portfoio is just text. You will make your email address a live link.

Linking to External Sites: Instructor

### Tips for Delivering This Lesson

* This lesson begins with a discussion of the parts of a web address. It is helpful to view several example websites and dissect their parts as a group. For classes that have no problem with this activity, it can be fun to dissect more challenging web addresses, such as those with long lists of parameters following the file name.
* Students may be unfamiliar with the **http://** prefix to a URL, since modern browsers don't require it, and many don't even display it.

### Example Output

When this lesson is complete, students will have added a link to their school home page within their index.html file.

Linking to External Sites: Student

**Activities**

1. Return to index.html in your web portfolio.
2. Locate the name of your school in the second div element.
3. Change the name of your school into a link to your school's home page. Use the above example as a model.

Linking to Pages Within Your Web Site: Instructor

### Tips for Delivering This Lesson

* This lesson provides an opportunity to discuss the difference between absolute and relative URLs. Students should know that if they're linking between pages that are stored locally, using an absolute path to these files may damage the link when the files are made public or moved to a new location. Therefore it's a good practice to use relative URLs to reference local files, and absolute URLs to reference external files, i.e., those that are stored elsewhere on the Internet.

### Example Output

When this lesson is complete, students will have added links to the navigation menu in their index.html file.

Linking to Pages Within Your Web Site: Student

**Activities**

1. Return to index.html of your web portfolio.
2. Earlier you created an unordered list to serve as a navigation menu. To make that into a functional menu, convert the text of each item (Home, Accessibility, Usability, Graphics, Tools, and Javascript) into links that target your other portfolio web pages. Since these other pages are all in the same directory as the home page, you can use a *relative URL* - you don't need to spell out the full address. As an example, here's what the link to accessibility.html will look like:

<li>

<a href="accessibility.html">Accessibility</a>

</li>

Special Types of Links: Instructor

### Tips for Delivering This Lesson

* In this lesson students create a "Skip to main content" link. When they test this link, they might not see a visible effect because their web page doesn't have that much content yet. It would therefore be helpul to share examples of sites that use this feature so they can see how it can be very useful.
* Since the primary beneficiaries of "skip to main content" links are individuals with disabilities, this lesson provides an excellent opportunity to revisit accessibility. For students to fully appreciate why these links are important, ask them to visit a site that has lengthy navigation menus and no "skip to main content" link. Almost any website will do. Assign them a specific link on a specific site, and forbid them from using a mouse for this exercise. Remind them that there are individuals with physical disabilities and people who are blind who are unable to use a mouse, so they are forced to navigate this way. For added effect, consider instructing students to wait one full second before they hit tab again, since that's a pace that's common for certain individuals with severe physical disabilities. After students have experienced how difficult it is to move around a website without a mouse, ask them to perform the same activity using one of the example sites listed above, and to press Enter when the "skip to main content" link has focus. It's best to do this activity before asking students to create their own, since that way they'll have a better appreciation for why they're doing what they're doing.
* The final activity in this lesson is to create a mailto link. As explained on the student lesson page, this might not work in some environments, depending on how email clients are configured. This is one good reason to avoid mailto: links. The other is that these links are easily harvested by spambots and used for malicious purposes. However, these links are still very common on the web, so students should understand their purpose, as well as their limitations.

### Example Output

When this lesson is complete, students will have added a "skip to main content" and "mailto" link to their index.html file.

Special Types of Links: Student

**Activities**

1. Return to the index.html file of your portfolio.
2. Find the div element that you used to wrap the main content of your page (the part with all the sub-headings and paragraphs). Add an id attribute to that div element. An id is a unique identifier for a specific element on a web page. It enables you to easily access that specific element so you can stylize it with CSS or manipulate it with Javascript. You can also link to any element with an id attribute. An id can be any text you like, but it's best to assign text that describes the function of the element. Since this particular div contains the main content of the page, we'll give it an id of "main". Here's what the opening div tag should look like when you're finished:
3. <div id="main">

1. Now that you have an id attribute marking the start of the main content, you can add a link to the top of your document that jumps directly to that content. At the very top of your web page, just inside the <body> tag, add a new div element that contains a same-page link, like this:

<div>

<a href="#main">Skip to main content</a>

</div>

1. Save, refresh, and test the new link. You should jump directly to the main content. If the page is short, you might not see any visible changes, but know that you have just made your page *much* easier to navigate for users with disabilities.
2. Next, locate your email on the page. Change the email address to an email link by coding it as follows:

<a href="mailto:judy@ohs.edu">judy@ohs.edu</a>

1. Save. Refresh your browser and try the link. If your browser and email software support this functionality, a blank email window should pop up with the email address already entered on the *To:* line. Note that some users' technology won't support this feature. Also, email addresses that are coded this way can be automatically harvested by malicious programs, and will soon thereafter be inundated with spam. For both of these reasons, including email links on websites is no longer a recommended practice, but we've included it here nonetheless because there are still lots of these links out there on the Web, it's good to understand them.

**Creating a Data Table**

This module consists of a single lesson that will introduce students to the basics of creating a data table.

HTML tables have historically been used to control the layout of a page. However, using tables for layout is no longer considered a good practice, and students in this course should know that the original (and now only) purpose of tables is to display tabular data, where rows and columns are used to communicate relationships.

This module consists of a single lesson that will introduce you to the basics of creating a data table.

Tables were introduced to the web with the original purpose of displaying data in rows and columns. In time they came to be used for an additional purpose: page layout. This lesson will focus on their original purpose. Page layout will be explored later in this course.

*Creating a Data Table: Instructor*

### Tips for Delivering This Lesson

* This lesson first presents a sample table, including the source code, then asks students to create their own table according to the lesson's specific instructions. Many students find HTML table markup to be initially confusing, since the information as arranged in the HTML source code is not easily navigated by column, as it is in the final table product. Therefore, we recommend that you walk students through the example source code to be sure they understand. Looking together at additional examples of tables may be helpful as well.
* It may also be helpful to show students how tables have been applied in modern web design.
* Another useful strategy for helping students to understand table markup is to find real world examples of data tables and discuss them as a class. Try to identify the HTML elements that are used to give the table it's structure.
* An extension of the above activity is to print one or more data tables, then ask students to write HTML elements on the table in their appropriate locations. For example, the entire table would have a <table> tag above it, and a </table> tag below it; each row would be proceeded by a <tr> tag and followed by a </tr> tag; and each cell would include either <th> and </th> tags, or <td> and </td> tags.
* When selecting real world tables for either of the above activities, it's important to select tables that are used properly (for data), as opposed to tables that are used for layout, since the latter is an obsolete practice.
* The student lesson page includes an example table (school lunch menu). This table has six columns and three rows, which is a different structure than the table students are asked to create in the lesson activity (which has two columns and seven rows). It is helpful to explicitly point this difference out to students; otherwise they may get stuck trying to force the assigned content into the example table's structure.

### Example Output

When this lesson is complete, students will have added a data table to their accessibility.html file.

Creating a Data Table: Student

## Learner Outcomes

At the completion of this exercise:

* you will be able to create a simple data table that includes <table> <caption>, <thead>, <tbody>, <tr>, <th> and <td> elements as well as scope attributes.

## How to Create an HTML Table

1. HTML tables begin with <table> and end with </table>.
2. Inside the table element, you can optionally include a caption element, which contains a brief title or description of the table's content. Most browsers display the caption above the table.
3. Like HTML documents, tables include a head and body. In tables, these are specified using the <thead> and <tbody> elements respectively. These tags are optional, but they should be included on long tables. If long tables are printed and span multiple pages, browsers will repeat the content of the <thead> element (usually the top row(s) of the table) at the top of every printed page.
4. Each row in a table begins and ends with table row (tr) elements: <tr></tr>
5. Each cell in the table begins and ends with either table header (th) elements or table data (td) elements, depending on what type of information the cell contains.
   * If a cell contains headers, it begins and ends with th elements: <th></th>
   * If a cell contains data (not headers), it begins and ends with td elements: <td></td>
6. Table header elements (th) should also include a scope attribute, which is either scope="row" or scope="col". This instructs screen readers as to which headers apply to which cells. Screen readers read tables row by row from left to right, and without this extra markup blind users would have a difficult time figuring out which headers apply to the cell they're in.

Compare the following table with the code that was used to create it:

| **School Lunch Menu** | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** |
| **Carnivores** | Sausage pizza | Corn dogs | Sloppy Joe | Beef taco | Chicken and dumplings |
| **Herbivores** | Veggie pizza | Veggie dogs | BBQ tempeh | Bean burrito | Tofu teriyaki |

<table>

<caption>School Lunch Menu</caption>

<thead>

<tr>

<td> </td>

<th scope="col">Monday</th>

<th scope="col">Tuesday</th>

<th scope="col">Wednesday</th>

<th scope="col">Thursday</th>

<th scope="col">Friday</th>

</tr>

</thead>

<tbody>

<tr>

<th scope="row">Carnivores</th>

<td>Sausage pizza</td>

<td>Corn dogs</td>

<td>Sloppy Joe</td>

<td>Beef taco</td>

<td>Chicken and dumplings</td>

</tr>

<tr>

<th scope="row">Herbivores</th>

<td>Veggie pizza</td>

<td>Veggie dogs</td>

<td>BBQ tempeh</td>

<td>Bean burrito</td>

<td>Tofu teriyaki</td>

</tr>

</tbody>

</table>

Not sure how the HTML table markup works? Ask your instructor to walk you through it. Be sure you understand the above markup before proceeding to the activity.

## Activities

1. In the current lesson, you will use an HTML table to create a web accessibility checklist, of how people with disabilities access the web, that you can refer to later as you design websites, to assist you in ensuring that your web content doesn't needlessly exclude anyone.
2. Open the file accessibility.html that you created earlier in the course. At this point, this file should include nothing but "bare bones" HTML.
3. Use the following instructions to add a table to the body of your web page. Consult the above example table markup as a model. Note that the School Lunch Menu table has three rows and six columns, whereas the Web Accessibility Checklist will have two columns and seven rows. The table you're creating should look something like this: 
4. Save your work, and check it in your web browser to be sure it looks like you expect it to. If it doesn't, that's normal - tables can be tricky! It sometimes takes several trials to get all the tags just right. Don't worry about spacing between and within cells: We'll be addressing that in a future lesson.

### Instructions for Creating Your Table

The caption of the table should be "Web Accessibility Checklist"

The table should have two columns, with the following headers:

1. User characteristic
2. Accessible design tip

In the first column (the column with header "User characteristic"), list the following user characteristics related to web accessibility, in the order in which they're presented below.

1. Unable to see
2. Unable to perceive colors
3. Unable to use mouse
4. Unable to hear
5. Prone to having seizures
6. Easily distractible

In the second column (the column with header "Accessible design tip"), the following web design issues correspond with the same-numbered user characteristic from the first column.

1. Code all images with ALT text
2. Avoid using color alone to convey information
3. Be sure all website features can be accessed using keyboard
4. Add captions to multimedia
5. Avoid content that flashes
6. Keep the design simple