CHAPTER OUTLINE

6.1 Introduction

This chapter presents two web page features—links and images—that are emblematic of what it means to be a web page. Almost all web pages include links and images, and they are usually crucial to the web page's popularity. And if you're a web page, popularity is everything.[1]

In Chapter 4, you learned a few link syntax details while implementing a `nav` container. In this chapter, you'll learn more link syntax details, plus various techniques for jumping to different link targets. In particular, you'll learn how to jump to a target location on a different web page, as well as to a target location on the current web page. Next, you'll learn how to download a file. For all the jumping and downloading operations, you can use a path to identify the location of your target, and you'll learn different techniques for specifying paths. You'll also learn techniques for formatting the links, using various CSS rules.

After learning about links, next you'll learn about another basic building block—images. You can implement a link to an image, but in this chapter, we focus on standalone images. In Chapter 4, you learned a few image syntax details while implementing a `figure` container. In this chapter, you'll learn a few more image syntax details, but most of the image discussion in this chapter will be about the different types of images that are available. In particular, you'll learn about bitmap image file formats (GIF, JPEG, and PNG) and a vector graphics file format (SVG).

6.2 a Element

To implement a link, you'll need to use the `a` element. Here's an example `a` element that implements a link to Park University's website:

```
<a href="http://www.park.edu">Park University</a>
```

[1] This rather shallow notion of success is particularly prevalent with younger web pages, where getting invited to the "popular" web pages' parties is paramount.

The text that appears between an a element's start tag and end tag forms the link label that the user sees and clicks on. So in this code, the link label is "Park University." By default, browsers display link labels with underlines. So this code renders like this:

Park University

The blue color indicates that the linked-to page has not been visited. We'll discuss visited and unvisited links later on.

When the user clicks on a link, the browser loads the resource specified by the href attribute. For this example, the "resource" is another web page, so when the user clicks on the "Park University" link, the browser loads that web page—the one at http://www.park.edu. As an alternative to specifying a web page for the href attribute's resource, you can specify an image file for the href attribute's resource. We'll show an example of that in the next chapter.

Besides enabling a user to load a resource (which usually means jumping to another web page), the a element can be used as a mechanism that enables a user to download a file of any type—image file, video file, PDF file, Microsoft Word file, and so on. To implement that download functionality, include a download attribute as shown here:

```
<a download href="http://www.park.edu/catalogs/catalog2018-2019.pdf">
  Park University 2018-2019 catalog</a>
```

As always for the a element, the browser displays the text that appears between the start tag and end tag; in this case, that's "Park University 2018–2019 catalog." When the user clicks on that text, the browser downloads the file specified by the href attribute. The user can then choose to view it or save it. Normally, the download attribute has no value, but if you (as the web programmer) want the end user to save the file using a different filename than that specified by the href atttibute, then you should include a filename for the download attribute's value, such as download="Park University 2018-2019 catalog.pdf". But be aware that the browser might override your download attribute's filename and use the href attribute's filename instead.

Continuation Rule for Elements that Span Multiple Lines

Did you notice that the preceding a element code fragment spans two lines and the second line is indented? This subsection explains that indentation. It's a style thing, and the explanation is not specific to the a element. This subsection is a digression from the rest of this chapter, and it's relevant for examples throughout the rest of the book.

The a element is a phrasing element (which means that it displays "inline," like a phrase within a paragraph). Most phrasing element code is short and can easily fit on one line, but the preceding a element is rather long and spans two lines. The solution was to press enter at the end of the a element's start tag and indent the next line. We use that same solution for other elements that are too long to fit on one line. Here's a meta element example:

```
<meta name="description"
  content="This web page presents Dean family highlights.">
```

Note that we press enter after the `name` attribute-value pair. In the `a` element example, we pressed enter after the `a` element's start tag. The goal is to press enter at a reasonable breaking point. For the preceding example, if we wait to press enter until after "Dean family," that would not be a "reasonable breaking point." It would split the `content` attribute's value across two lines. Doing so would make the code slightly harder to understand and thus defeat one of the primary purposes behind good style—understandability.

Back in Chapter 2, we introduced you to block formatting for block elements, which means the start and end tags go on lines by themselves. The `p` element is a block element, and here's a properly formatted `p` element:

```
<p>
  Known for its Computer Science program,
  <a href="http://www.park.edu/informationAndComputerScience/accolades">
  Park University</a> is tied for first in the number of Turing Award
  winners among all Missouri universities whose motto is "Fides es Labor."
</p>
```

Press enter.

Align the `a` element's continuation line with the prior line.

Within the `p` container, there's an `a` element that spans more than one line, and we align its continuation line with the line above it (we don't indent further). That's different from the prior `a` element example. Indenting continuation lines is a bit of a gray area. If the continuation line is in a container that represents something that is supposed to look like a paragraph (e.g., `p` or `blockquote`), then do not indent. Otherwise indent.

Note the line break in the source code after "program,". We inserted a line break so we could fit the `a` element's entire start tag on one line. If we attempt to put the `a` element's entire start tag on the same line as "program," then *line wrap* would occur if we printed the code on paper. Here's what that would look like:

```
<p>
  Known for its Computer Science program, <a
href="http://www.park.edu/informationAndComputerScience/accolades">
  Park University</a> is tied for first in the number of Turing Award
  winners among all Missouri universities whose motto is "Fides et Labor."
</p>
```

line wrap

Note how `href` wraps to the next line where it's not indented. The `href` is aligned with the `<p>` start tag, and that implies that `href` is separate from the `p` element rather than a part of it. And that implication makes the logic hard to follow. The moral of the story is, for statements that might be too long to fit onto one line, press enter at an appropriate breaking point to avoid line wrap.

Types of `href` Attribute Values	Where the Link Jumps To
absolute URL	Find the resource on a different web server than the current web page.
relative URL	Find the resource on the same web server as the current web page. Specify the location of the resource by providing a path from the current web page's directory to the destination web page.
jump within current web page	Find the resource within the current web page. Specify the location of the resource by providing an `id` value for an element in the web page.

FIGURE 6.1 `href` attribute values

Different Types of `href` Values

As you know, the `a` element's `href` attribute value specifies the resource that is to be loaded. In addition to specifying the resource, the `href` attribute value indicates where the link jumps to in order to find the resource. Indicating where the link jumps to is not a trivial task. Take a look at FIGURE 6.1, which provides an overview of the different link-jumping techniques employed by the `href` attribute's value.

For an example that uses an *absolute URL*, suppose you want to add a link on a Facebook page that directs the user to an Instagram page. Here's the link code to do that for a subscriber named Hannah Davis:

```
<a href=" https://www.instagram.com/hannahDavis.html">
  Hannah's Instagram</a>
```

Note the https value for the `href` attribute. You've seen http in the past; https is another popular protocol that you can use with the `href` attribute. It stands for hypertext transfer protocol secure. So the https protocol provides more security for communications than does http.

To jump to a web page that resides on the same web server as the current web page, for the link's `href` attribute, use a relative URL. The relative URL specifies a path from the current web page's directory to the destination web page. The next section provides complete details and examples.

To jump to a designated location within the current web page, for the link's `href` attribute, use a value starting with # such that that value matches an `id` attribute's value for an element in the web page. We introduced this technique earlier, and we'll provide more details and examples later in this chapter.

6.3 Relative URLs

As promised, in this section we provide additional details about relative URLs. A relative URL value allows you to jump to a web page that resides on the same web server as the current web page. It does so by specifying a path from the current directory to the destination web page. The

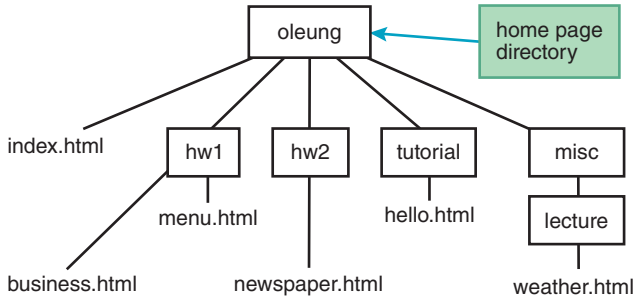

FIGURE 6.2 Example directory tree

current directory is the directory where the current web page resides. The destination web page is the page that the user jumps to after clicking on the link.

In forming a path for a relative URL value, you'll need to understand how files and directories are organized in a *directory tree* structure. Note the example directory tree in **FIGURE 6.2**. It shows the container relationships between all the files and directories that are within the `oleung` directory. The `oleung` directory is the home page directory for student Olivia Leung. A *home page* is the default first page a user sees when the user visits a website. We'll have more to say about home pages shortly, but for now, just realize that a *home page directory* is the directory where a home page resides, and a home page is the starting point for a user's browsing experience on a particular website. A *subdirectory* (also called a subfolder or a child folder) is a directory that is contained within another directory. Thus, in Figure 6.2, there are four subdirectories within the `oleung` home page directory and one subdirectory within the `misc` subdirectory. The other entities in Figure 6.2 (the words without borders) are files. The `index.html` file is in the `oleung` directory, and the other files are in the tree's subdirectories. Because `oleung` is the home page directory, you might have guessed that `index.html` is the home page file. That is indeed the case.

In forming a path for a relative URL value, you'll need to navigate between a starting directory and a target file. In doing so, you'll need to follow these rules:

- Use /'s to separate directories and files.
- Use ". ." to go from a directory to the directory's parent directory.

In the second bullet, what does *parent directory* mean? In Figure 6.2, `oleung` is the parent directory of `hw1` because `oleung` and `hw1` are connected by a single line with `oleung` on the top. That should make sense because it parallels how human parents are displayed in a genealogy tree. The point of all this is that if you're in a directory and you want to go to that directory's parent directory, you need to use `..` For example, suppose you want to provide a link on the newspaper page that takes a user to the home page, `index.html`. Because the newspaper page is in the `hw2` directory, the `hw2` directory is considered to be the current directory. The home page is in the `oleung` directory. The `oleung` directory is the parent directory of `hw2`, so you need to use `..` in order to navigate up to the `oleung` directory. Here's the relevant `a` element code:

```
<a href="../index.html">Olivia's Home Page</a>
```

Note the / (forward slash) between `..` and `index.html`. As explained earlier, the / is a delimiter that separates directories and files. The `..` refers to a directory and index.html is a file, so the / is

necessary to separate them. Does it make sense that `..` refers to a directory? Remember that `..` navigates from the current directory (`hw2` in this example) to its parent directory (`oleung` in this example), so the result is indeed a directory.

Relative Path Examples

Using Figure 6.2's directory tree, can you try to come up with the code for an `a` element that resides in the `index.html` file and takes the user to the business page? See if you can do that on your own without glancing down at the answer.

Assuming you've tried to work it out on your own, now you may look at the answer:

```
<a href="hw1/business.html">Business Page</a>
```

Note that `href`'s value starts with `hw1`. In coming up with the answer on your own, did you start with `..` instead? It's a common error among beginning web programmers to feel the need to use `..` to go up from the current web page to that web page's directory. Try to avoid that misconception. There's no need to go up. If you're in a web page, then you're already in that web page's directory. So to go from the `index.html` page to the business page, you simply go down from `oleung` to `hw1` by specifying `hw1`, then down to the business page by specifying `/business.html`.

The `index.html` page is the website's home page, and as such, it's the first page that you look at when you visit a website. To help with a user's viewing experience, home pages should normally contain links to other pages on the website. So for your next challenge, you should implement another link from the home page—this time, have the link go to the weather page. Try to come up with the code on your own. Assuming you've made an honest attempt, now you can look at the answer:

```
<a href="misc/lecture/weather.html">Weather Page</a>
```

Because the link resides on the home page, the path originates from the home page directory, `oleung`. To get to the weather page, you have to go down to the `misc` directory and then down to the `lecture` directory. In the example code, notice the `/`'s separating the two directories and the `weather.html` filename.

For one last relative URL challenge, try to come up with the code for an `a` element that resides in the business page and takes the user to the menu page. Once again, see if you can do that on your own without glancing down at the answer. Here's the answer:

```
<a href="menu.html">Menu Page</a>
```

Note that there is no directory in the `href` value—only the filename by itself. With the business and menu pages both in the same `hw1` directory, there's no need to change directories and therefore no need for a path in front of the filename.

Path-Absolute URLs

As stated previously, a relative URL is for jumping to a web page when the current web page and the target web page are on the same web server. In the prior examples, the relative URL's path started at the current web page's directory. As an alternative, you can have the relative URL's path start at the web server's root directory. A web server's *root directory* is the directory on the web

server that contains all the directories and files that are considered to be part of the web server. [2] If you want to have the relative URL's path start at the web server's root directory, preface the URL value with `/`. For example, using Figure 6.2's directory tree, if `oleung` is immediately below the web server's root directory, the following code could be added to any of the web pages shown, and it would implement a link to the `index.html` page:

```
<a href="/oleung/index.html">Olivia's Website</a>
```

A URL value that starts with a `/` is referred to as a *path-absolute URL*, and it's a special type of relative URL (it's considered a relative URL because the path is relative to the current web server's root directory). The term path-absolute URL comes from the WHATWG. Remember the WHATWG? It stands for Web Hypertext Application Technology Working Group. It's the organization that keeps track of the HTML5 standard with a living document. Its standard aligns very closely with the W3C's HTML5 standard, but because it's a living document, the WHATWG is free to update things at any time it sees fit. Unfortunately, the W3C does not have a formal term for a path that starts with /. But rest assured that the path-absolute URL syntax is valid—all major browsers have supported it for many years. By the way, if you see the term *root-relative path*, that's just another way to refer to a path-absolute URL. In the real world, both terms are popular.

6.4 `index.html` File

If a user specifies a URL address that ends with a directory name, then the web server will automatically look for a file named `index.html` or `index.htm` and attempt to load it into a browser. This occurs when you specify a URL for a link's `href` value and also when you enter a URL in a web browser's address box.[3]

The default searched-for file can be reconfigured by the web server's administrator. It's common for Microsoft IIS web server administrators to use `default.htm` as another default filename for displaying a web page when the URL address ends with a directory name. According to the coding-style conventions in Appendix A, we recommend that you use `index.html` for your home page file names because that name is the most standard. We prefer `.html` to `.htm` because `.html` is more descriptive—after all, "html" describes the code contained in the file.

If a user specifies a URL address that ends with a directory name, and the web server cannot find a file named `index.html` or `index.htm` (or possibly `default.htm`) in the specified

---

[2] As you might recall, the term "web server" can refer to (1) the physical computer that stores the web page files or (2) the program that runs on the computer that enables clients to retrieve the web pages. When talking about a "web server's root directory," we're not referring to the computer's hard drive root directory. We're referring to the directory that the web server program designates as the top-level container directory for all the web-related files that the web server computer uses.

[3] During the development process, you'll probably want to implement your web pages on your local computer's hard drive. You can load most of your hard drive web pages to a browser by simply double clicking on the file within Microsoft's File Explorer tool. But if you double click on a directory name, the index.html file won't load by default. Why? Because the web server is the thing that knows to look for the `index.html` file (i.e., it knows to search for it if it sees a directory name). And you bypass the web server if you double click a web page file within File Explorer.

directory, then the web server will either (1) load a web page that shows the contents of the specified directory, or (2) display an error page (e.g., "Directory Listing Denied" or "HTTP 404 - Page Not Found"). To avoid the directory contents web page (which is rather ugly) or the error page, you should include an `index.html` file in every directory that might be specified as part of a URL. It might seem a bit weird to have multiple `index.html` files within your website, but for larger websites, get used to it. Clearly, you need an `index.html` file in the directory that sits at the top of your website's directory tree structure—that particular `index.html` file is your website's home page. The other `index.html` files are not the official home page, but you can think of them as "home pages" for different areas within your website.

6.5 Web Design

Because a home page is the default first page a user sees when the user visits a website, it's the web developer's first—and possibly only—opportunity to make a good impression on users. So try to get it right, or your users might leave and never come back. In this section, you will learn a few tips on how to make a good first impression. These tips are part of a software area known as *web design*, which is comprised of these subareas: user interface design, user experience design, graphic design, and search engine optimization. Sorry, explaining all that is beyond the scope of this book. But we do explain some of it—we scratch the surface on user interface design and user experience design. That's a surface that needs to be scratched, so let's begin.

User Interface Design

In a general sense, *user interface design* (UID) refers to the mechanisms by which users of a product can use the product. For web pages, the mechanisms are things like text, color, pictures, buttons, text boxes, and progress bars. A good UID designer will anticipate users' needs and create an interface that meets those needs by incorporating components that are easy to understand and use.

After the home page downloads, users will want to identify the web page's main content quickly. To help in that regard, you should try to avoid clutter, and focus on clear, concise words (and graphics, if appropriate) that describe the web page's main content. Remember that for a nontrivial website, the home page is only for the main content and links to other web pages, not for lots of details. If you need to provide lots of details for something, you should put those details on a separate web page and link to that page from the home page. The link labels themselves are important. For each link label, use only one word or a few words that get to the point quickly. Don't be afraid to remove unnecessary text and to have whitespace on your home page. Whitespace can provide a nice respite for stressed-out web surfers.

In presenting the web page's content, it's important to be consistent with your text and colors. For text, you should limit the number of text fonts used. Pick pleasing fonts that go together well for your main content and your subsidiary content. Make sure that the foreground text colors contrast with the background colors so the text is easy to read. Normally, that means the contrasting colors should be different in terms of lightness and darkness. You should pick a set of colors for your text, background, and graphics that complement each other. Apply a consistent strategy for choosing which colors go to which type of content.