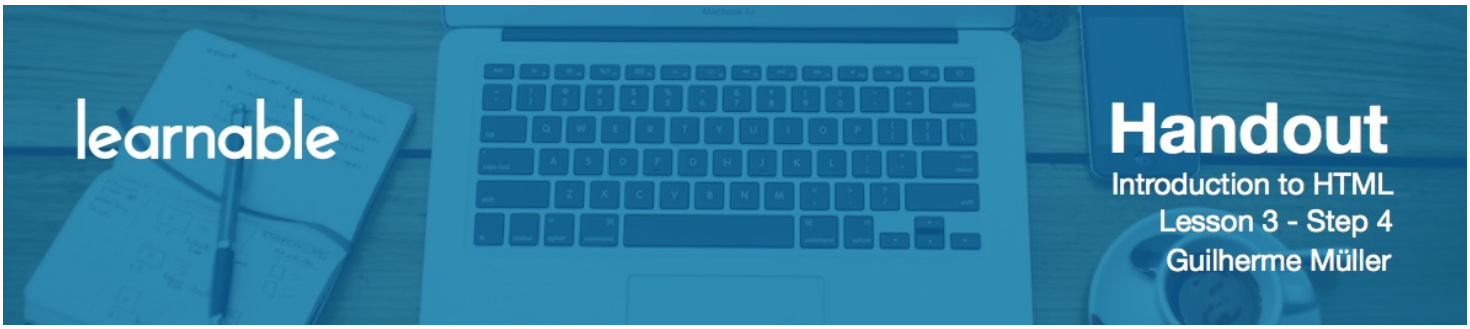

Introduction

In this lesson, we are going to use the anchor tag to create links to other pages and to the sections on the same page. To prepare for the lesson open up the *"links-begin.html"* file. There are also three other files: *"test.html"*, another *"test.html"* inside the folder called *"folder"* and a *"picture.jpeg"* file. Let's try to create a link to this first *"test.html"* file located in the same folder as our "links-begin.html" file.

Links are the core of the hypertext and one of the main features of the revolution of the Internet. The tag used to mark up links is the `a` tag. There is also a `link` tag but that one is used for another purposes.

Basic links

Let's wrap text in the first paragraph with an `a` tag:

```
<p>
    <a>Link to file</a>
</p>
```

This can be considered a link but the tag is lacking it's most important attribute called `href`, which stands for **hyper reference**. The `href` attribute provides the destination of the link:

```
<p>
    <a href="test.html">Link to file</a>
</p>
```

Having this in place, we explain the browser that the link should lead to *"test.html"*.

There is also a back link present in the *"test.html"* so that the user can return to the original *"links-begin.html"* file. Open up *"test.html"* file and check that out:

```
<p>
    <a href="links-begin.html">Back</a>
</p>
```

Now let's create another link, this time leading to the *"test.html"* inside the folder.

Relative paths

To do that, we have to specify the path to the file like this:

```
<p>
    <a href="folder/test.html">Link to file in folder</a>
</p>
```

This is called a **relative** path and it is similar to what we did in the first example. Relative paths specify a path to a file related to where the linking file resides. This basically means that the browser has to enter the provided folder and then look for a *"test.html"* file inside. In the previous example we just put *"test.html"* in the `href` attribute. That meant that the browser should search for a *"test.html"* file located in the same folder as the linking file. Go on and test that in the browser.

In the *"folder/test.html"* file we also have a back link, taking us to the *"links-begin.html"* file. But if you open this file in the editor, you'll notice that the path for `href` is a bit different. We have the `../` before the name of the file. This is a command meaning that the browser has to go up a folder and then search for the *"links-begin.html"* file there. This is also a relative path because we're specifying the position of *"links-begin.html" *related to the *"test.html"* file.

Absolute and external links

We can also use **absolute** paths for links. Absolute paths, as the name implies, point directly to the file with a full address. For example, the *"links-begin.html"* file may reside on the desktop on the PC and therefore has an absolute path of *file:///C:/Users/Guilherme/Desktop/HTML Course/links-begin.html*. You can put this path in the `href` attribute and then the link will point to the file no matter where the other file is located.

```
<p>
    <a href="file:///C:/Users/Guilherme/Desktop/HTML Course/links-begin.html">Link to file</a>
</p>
```

Another common use for absolute links is linking to an external website. For example, we can create a link to *learnable.com* like this:

```
<p>
    <a href="http://learnable.com">Link to external site (absolute link)</a>
</p>
```

Notice that we need to specify the protocol: for websites it will be *http*. Sometimes you will also need to provide *www* however most sites do not require it anymore.

Sometimes you'll want the link to open in another browser's tab. There were a lot of discussions recently whether websites should open other tabs or the decision should remain only with the user. Nevertheless this technique is very widely used, despite some usability problems.

Opening links

Before showing you the solution let me note that for this kind of interaction I'd recommend a combination of HTML and JavaScript, because JavaScript is the front-end layer that should be responsible for actions like this.
As an example let's create a link to *learnable.com* that opens in another tab:

```
<p>
    <a href="http://learnable.com" target="_blank">Link to external site opening in another window (method not recommended – use
javascript)</a>
</p>
```

Here you can see that the `target` attribute is used. This is an attribute that was not valued in previous versions of HTML, but as long as many people use this, it was included in the specification of HTML 5. The target attribute may accept other values but the most common is `_blank` which will make the browser open another tab with the link.

We can also link using other protocols like mailto

We can also create links with other protocols like `mailto`. `mailto` is used to link to an email address:

```
<p>
    <a href="mailto:email@example.com ">Link with mailto: protocol</a>
</p>
```

When you click on this link, the action will vary with the browser and OS. Be careful, because a link like this can also make the user experience worse. In older versions of Windows, for example, you would be prompted to configure or open Outlook Express even if you didn't use this program. Today a browser like Firefox will ask the user what he wants to do with the link and may even offer to open Gmail.

Next, we have an image in our document that can also have a link attached:

```
<p>
    <a href="test.html"><img src="picture.jpg" alt="Tree with pink blossoms"></a>
</p>
```

The link works absolutely the same as in the previous examples. Some browsers will put a border around the image as well but that can be changed with CSS.

Anchor to current page

Another use for links is to link within the same page. We can achieve that using **anchors**. Let's suppose we're building a long document and we need to link to a title with an id of "content" somewhere in the page. As we've already seen, ids can be used with HTML, CSS, and JavaScript: they are very useful when you want to select something inside the document. As for the link, we can point it to a piece of content, using the hash symbol followed by the id:

```
<p>
    <a href="#content">Link to current page</a>
</p>
[…]
<h2 id="content">Content with ID</h2>
```

When you click on this link, the browser will go to that element, or to the end of the page, if the element can't be reached.

Another use for such links is to go to the top of the page. We can do that just by putting a hash inside the `href` attribute:

```
<p>
    <a href="#">Link to top or &ldquo;dummy link&rdquo;</a>
</p>
```

This is also called a **dummy link**, because basically it is going nowhere. It's common to initially code links like this, when we have no clear destination for them yet.

Block-level links

In the previous versions of HTML it was incorrect to have a link wrapping around block-level elements. Let's talk about those elements a bit. In HTML we have **inline** and **block** elements.

Inline elements are those which by default will appear in a continuous line like flowing text. For example, the `strong`, `em`, and anchor tags are examples of inline elements.

Block-level elements, in turn, will by default start in a new line like all the headings (`h1` through `h6`), paragraphs, lists and many other tags that we are going to meet throughout the course.

There is a rule saying that you cannot wrap an inline element around a block element. For example, we can't wrap an `em` tag around a `p` tag, but the opposite is possible. With that said, block-level links are an exception to that rule:

```
<a href="test.html">
    <h1>Block level link</h1>
    <p>
        The <code>a</code> tag containing block tags like <code>h1</code> and <code>p</code>.
    </p>
</a>
```

This is an interesting feature and it has it's implications but it should be used with caution. The link can become too long, causing problems with search engines and accessibility.