11.2.6

Web Scraping with Beautiful Soup

How can we use our knowledge of HTML to scrape a website? Let's consider an example.

Create a new Jupyter notebook, name it `html_case_study.ipynb`, and then paste the following code into the first cell:

```
from bs4 import BeautifulSoup as soup
```

*Note** Recall that Beautiful Soup is the library that we use to parse an HTML document—so we can choose and extract the information that we want from it.

In the next cell, paste the HTML code from `index7.html` as follows:

```
html = """
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    <title>Document</title>
</head>

<body>
    <div>
        <h2>First heading</h2>
        <p>First paragraph</p>
    </div>

    <div>
        <h2>Second heading</h2>
        <p>Second heading</p>
    </div>
</body>

</html>
"""
```

You've now pasted the entire block of code from `index7.html` into a Python string. Remember that in Python, we can enclose a string that spans multiple lines inside a pair of triple quotation marks ("""). In the preceding code, notice that we've also assigned this string to a variable named `html`.

Later, we'll use an automated browser to extract HTML code from websites. But for now, we'll paste HTML code for the sake of simplicity.

Next, in a new cell, enter and run the following code:

```
html_soup = soup(html, 'html.parser')
print(html_soup.prettify())
```

The preceding code returns our entire block of earlier HTML code! In the preceding code, the first line parses that block by using Beautiful Soup. We'll discuss how this works in more detail later, but for now, note that this allows us to get data from a selected portion of the document. The second line then prints that block to the screen.

In the next cell, enter and run the following code:

```
html_soup.title
```

The preceding code gets the HTML code of the `title` element. Specifically, it returns `<title>Document</title>`. That is, `html_soup` is an object that holds the parsed HTML code. And, its `title` attribute contains the HTML code of the title.

What if we care about only the text of the `title` element and not the HTML tags? We can further narrow our search to return only the title of the HTML document, which is `Document`. To do so, enter and run the following code in the next cell:

```
doc_title = html_soup.title.text
print(doc_title)
```

Now, let's find out if we can retrieve an element that contains other elements within it. To do so, enter and run the following code in the next cell:

```
div = html_soup.find("div")
print(div)
```

The preceding Python code uses the `find` method of the `html_soup` object to return an instance of a `div`, as the following code shows:

```
<div>
<h2>First heading</h2>
<p>First paragraph</p>
</div>
```

Recall that the HTML document contains two `<div>` tags. Why did the Python code return only one? The reason is that on its own, the `find` method returns only the first instance of the element that it's searching for. Later, we'll go over how to retrieve multiple elements.

Now that we know that the Beautiful Soup `find` method returns the first instance of an element, let's use it to retrieve the text of the first `h2` element in the HTML document. To do so, enter and run the following code in the next cell:

```
doc_heading = html_soup.find("h2").text
```

Next, let’s try scraping the text of the first paragraph (`p`) element on the page and saving it to a variable named `doc_paragraph`. To do so, enter and run the following code in the next cell:

```
doc_paragraph = html_soup.find("p").text
```

Finally, let's store the information that we scraped in a Python data structure. In this case, we'll use a dictionary. To do so, enter and run the following code in the next cell:

```
doc_info = {"title": doc_title,
            "heading": doc_heading,
            "paragraph": doc_paragraph}
```

Depending on our needs, we can export a dictionary of scraped data to a JSON file, a Pandas DataFrame, or a MongoDB database.

You’ve now learned the basics of an HTML document. As you learn about web scraping, knowing how HTML is structured will help you identify where to find the data that you want on a webpage. Next, you’ll learn about CSS. It will help you in that effort by targeting a single HTML element or a group of related ones.