11.3.4

CSS Case Study

Having learned how to use CSS selectors to style HTML elements, Robin now feels ready to apply this knowledge to scraping a webpage by using Python.

You'll now put your new skills to use by creating a case study. In this case study, you'll use CSS selectors in a basic web-scraping example. In Part 1, you'll use the `class` selector. In Part 2, you'll use the `id` selector.

CSS Case Study: Part 1

To begin, create a new Jupyter notebook, and name it `css_case_study.ipynb`. Then paste the following code into the notebook, and run the code:

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

Next, we'll use the Beautiful Soup library to parse HTML code. To do so, paste the following code (which is slightly modified from the code in `index6.html`) into the next cell:

```
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>

<style>
    .even {
        color: blue;
    }

    .odd {
        color: green;
    }
</style>

<body>
    <div>
        <p class="odd" id="first">First</p>
        <p class="even" id="second">Second</p>
        <p class="odd" id="third">Third</p>
    </div>

    <div>
        <p class="even" id="fourth">Fourth</p>
        <p class="odd" id="fifth">Fifth</p>
        <p class="even" id="sixth">Sixth</p>
    </div>
</body>

</html>
"""
```

In the preceding code, notice that as before, we assigned the entire block of HTML code to the `html` variable as a string.

Now, in the next cell, paste the following code, and then run it:

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

The preceding line of code converts the HTML string into a Beautiful Soup object.

Next, we'll use that object to choose and retrieve specific parts of the HTML code. Earlier, we used CSS attributes to change the color of an entire class. And, we can use Beautiful Soup to retrieve all the elements that belong to a class. So, we'll scrape all the paragraph elements that belong to the `odd` class. To do so, paste the following code into the next cell, and then run the code:

```
odds = html_soup.find_all("p", class_="odd")
```

In the preceding code, notice that we use the Beautiful Soup `find_all` method. Whereas the `find` method returns the first result of a search, the `find_all` method returns all the results. And, this method takes two arguments:

- The first argument, `"p"`, specifies that we're searching for paragraphs—that is, `p` elements.
- The second argument, `class_="odd"`, specifies that the elements must belong to the `odd` class.

NOTE

Notice the underscore (`_`) at the end of `class_`. We use an underscore because `class` is a reserved word (https://en.wikipedia.org/wiki/Reserved_word) in Python.

Finally, we assign the results of this search to the `odds` variable.

Next, we want to iterate through the results and display them by using a `for` loop. To do so, paste the following code into the next cell, and then run the code:

```
for odd in odds:
    print(odd)
```

The preceding code displays the HTML code for each `p` element that belongs to the `odd` class. The results are as follows:

```
<p class="odd" id="first">First</p>
<p class="odd" id="third">Third</p>
<p class="odd" id="fifth">Fifth</p>
```

Now, practice retrieving selected text and elements by class yourself in the following Skill Drill:

SKILL DRILL

Use a `for` loop to print only the text of each paragraph. You can do so by using the `text` attribute. Then try doing the same task by using a Python list comprehension (https://www.w3schools.com/python/python_lists_comprehension.asp). Finally, try printing the text of all the `p` elements that belong to the `even` class.

CSS Case Study: Part 2

Now that you've identified HTML elements by using the CSS `class` selector, you'll do the same but using the `id` selector. Recall that a `class` can contain multiple elements but that an `id` must be unique.

To begin, you'll scrape the text from a paragraph element that has an `id` of `first`. To do so, paste the following code into the next cell, and then run the code:

```
first = html_soup.find("p", id="first")
print(first.text)
```

In the preceding code, the logic is much the same as before. But, we use the `find` method instead of `find_all` and the `id` argument instead of `class_`.

You've now learned the basics of how a webpage is structured and how to target its elements. So, you're ready to dive more deeply into web scraping. In the next lesson, you'll learn how to use Chrome Developer Tools (DevTools) to examine a webpage. On a page with hundreds or even thousands of elements, DevTools will ease identifying the elements that you care about.