11.6.1

# Performing an Automated Web Scrape Across Multiple Pages

**Robin** has successfully scraped data from a single webpage. But to complete her task of scraping planetary climate data from the NASA website, she needs to use automated web scraping to collect data that spans multiple pages.

We've learned a lot about web scraping so far. Specifically, we've learned how to identify HTML elements that we're interested in, how to use DevTools and CSS selectors to drill down into the elements, and how to use Splinter and Beautiful Soup to extract and parse data. But, not all data resides on a single webpage. So in this lesson, we'll add to our set of skills by learning how to scrape data across multiple pages. We'll accomplish this by using Splinter to automate the clicking of buttons. We'll also learn how to scrape tables from websites.

Let's return to the Quotes to Scrape (http://quotes.toscrape.com/) website. We'll use a new notebook to scrape the quotes from the first page, click the Next button, scrape more quotes, and so on until we've scraped the quotes from five pages.

To do so, we’ll use the following process:

1. Set up a browser instance with Splinter and have it scrape the first Quotes to Scrape (http://quotes.toscrape.com/) webpage. Specifically, have the browser navigate to that website and then copy the content.
2. Create a Beautiful Soup object of the scraped content, which we’ll use in the next step.
3. Use Beautiful Soup to parse the content and find all the quotes on the page.
4. Use Splinter to automatically click the Next button (which appears at the bottom of each page) and then scrape the next page.
5. Repeat Steps 2ndash;4 until five pages have been scraped.

To begin, create a new Jupyter notebook named `scrape_across.ipynb`, and then paste the following code into the first cell:

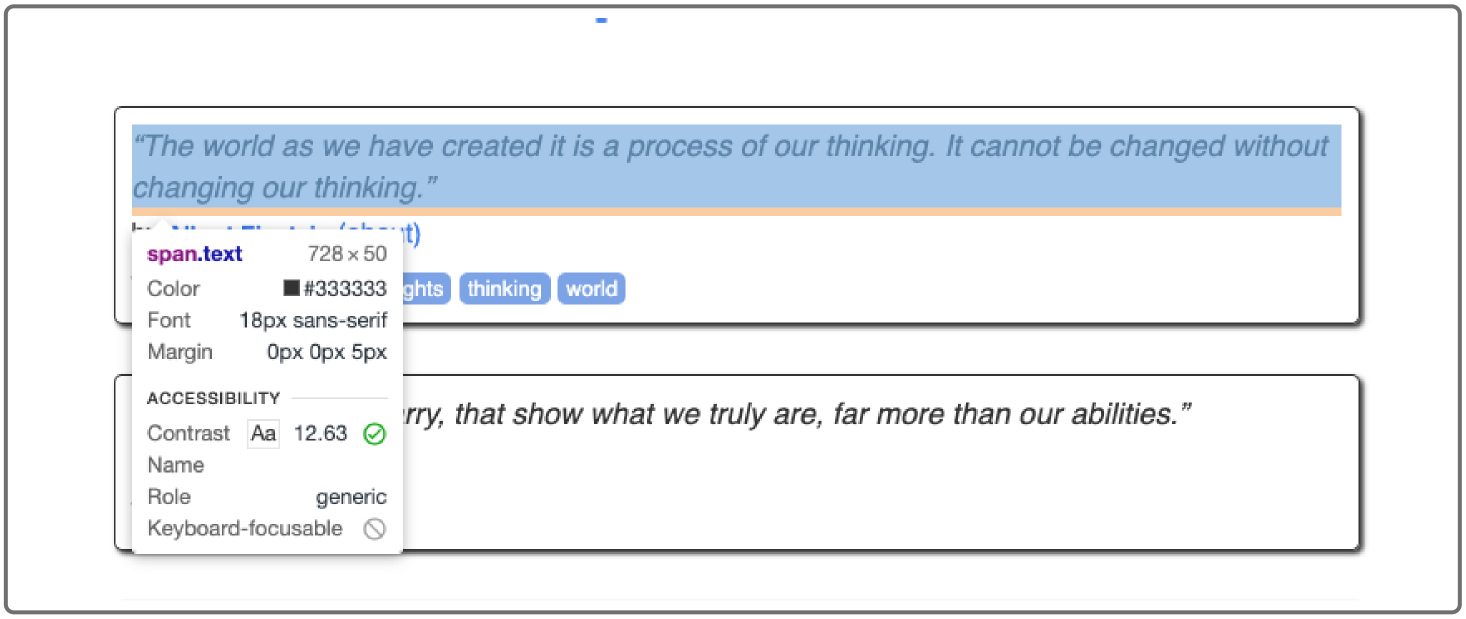
```
from splinter import Browser
from bs4 import BeautifulSoup as soup
from webdriver_manager.chrome import ChromeDriverManager

executable_path = {'executable_path': ChromeDriverManager().install()}
browser = Browser('chrome', **executable_path, headless=False)

url = 'http://quotes.toscrape.com/'
browser.visit(url)
html = browser.html
quote_soup = soup(html, 'html.parser')
```

In the preceding code, the Splinter browser visits the URL and then the HTML content of the page gets assigned to the `html` variable. Beautiful Soup then parses the HTML content.

Our first task is to identify the `class` attribute of each quote on the page. Using DevTools, we learn that each quote exists inside a `span` element that belongs to the `text` class, as the following image shows:


“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”

span.text 728 × 50
Color #333333
Font 18px sans-serif
Margin 0px 0px 5px
ACCESSIBILITY
Contrast Aa 12.63
Name
Role generic
Keyboard-focusable

rry, that show what we truly are, far more than our abilities.”

So, we’ll scrape all the quotes on this page by using the `find_all` method to extract all the `span` elements that have the `text` class. To do so, enter and run the following code in the next cell:

```
# Scrape all quotes on a single page
quotes = quote_soup.find_all('span', class_='text')

for quote in quotes:
    print(quote.text)
```

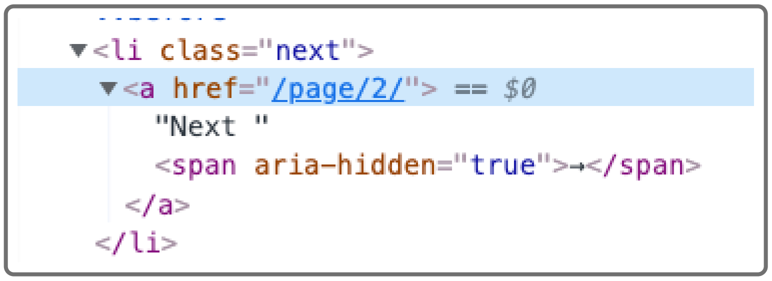
In the preceding code, the `find_all` method returns a list of all the `span` elements that have the `text` class. Each item in the list resembles `<span class="text" itemprop="text">"QUOTE"</span>`. Because we're interested in only the quote portion, we use its `text` attribute in a `for` loop.

## Click a Button

Our next step is to automate clicking the Next button to go to the next page.

To begin, let's try to identify the `class` or `id` attribute of this button. Using DevTools, we observe the following HTML code:

```
<li class="next">
  <a href="/page/2/">      "Next "
      <span = aria-hidden="true">+</span>
  </a>
</li>
```



```
▼<li class="next">
  ▼<a href="/page/2/"> == $0
     "Next "
     <span aria-hidden="true">→</span>
   </a>
 </li>
```

In the preceding code, notice that the `<a />`, or anchor, tag contains the link. And, the `<li />`, or list item, tag contains the `<a />` tag. The `<li />` tag has a `class` attribute named `next`, but the `<a />` tag doesn't have a `class` or `id` attribute. So, we can't use such an attribute to easily find the link.

Now, before moving on, try to confirm that the link of the Next button doesn't have a `class` or `id` attribute in the following Skill Drill:

**SKILL DRILL**

Print a list of all the links on the first page by finding all the anchor tags. Can you find the Next button in that list? Does it confirm our observation that this link doesn't have an `id` or `class` attribute?

Fortunately, we can use a method from Splinter to find a button according to its text. To do so, enter and run the following code in the next cell:

```
browser.links.find_by_partial_text('Next').click()
```

The preceding code navigates the browser to the next page. The `find_by_partial_text` method finds a clickable link according to its text—in this case, "Next". The `click` method, which is chained to it, performs the automated clicking action.

DEEP DIVE

## Put It All Together

You've done all the difficult work. Now, you'll put all the pieces together and write a script that scrapes five pages of the Quotes to Scrape (http://quotes.toscrape.com/) website. Here's the process at a high level:

1. Use Splinter to extract the HTML content from the current page.
2. Use Beautiful Soup to parse that HTML content.
3. Search for all the `span` elements that have a `text` class, and then print them.
4. Use the Splinter `find_by_partial_text` method to automatically click the Next button.

5. Repeat Steps 1–4 until five pages have been scraped.

To begin, create a new Jupyter notebook named `put_it_together.ipynb`.

As before, we want to set up the automatic browser to visit the website and then parse the HTML content with Beautiful Soup. That will set us up to scrape the first page of quotes. To do so, enter and run the following code in the first cell:

```
from splinter import Browser
from bs4 import BeautifulSoup as soup
from webdriver_manager.chrome import ChromeDriverManager

# Set up Splinter
executable_path = {'executable_path': ChromeDriverManager().install()}
browser = Browser('chrome', **executable_path, headless=False)

url = 'http://quotes.toscrape.com/'
browser.visit(url)
html = browser.html
quote_soup = soup(html, 'html.parser')
```

This time, we'll use a `for` loop to automatically visit multiple pages. We'll use `range(1, 6)` in our `for` loop to visit the first five pages of the website.

**NOTE**

The Python `range` function returns a sequence of numbers that we can iterate through. In this case, the sequence starts at 1 and stops at 5 (right before reaching 6).

To do this, enter the following code in the next cell:

```
for x in range(1, 6):
   html = browser.html
   quote_soup = soup(html, 'html.parser')
```

```
    quotes = quote_soup.find_all('span', class_='text')
    for quote in quotes:
        print('page:', x, '----------')
        print(quote.text)
    browser.links.find_by_partial_text('Next').click()
```

Let's more closely examine the preceding code:

1. The `for` loop will iterate five times.
2. During each iteration, `browser.html` extracts the HTML content from the page.
3. Beautiful Soup parses that HTML content with `soup(html, 'html.parser')`.
4. Beautiful Soup finds all the `span` elements that have a class of `text` on the page.
5. Another `for` loop prints the `text` attribute of each quote.
6. The Splinter `find_by_partial_text` finds the Next button to navigate to the next page. Then, the `click` method automatically clicks the button.

**IMPORTANT**

Beautiful Soup can search for text in many ways. But, the syntax typically remains the same: first find a tag and then find an attribute. We can search for elements by using only a tag, such as `<span />` or `<h1 />`. But, a `class` or `id` attribute makes the search more specific.

By including an attribute, we have a far better chance of scraping the data that we want.

Go ahead and run the code in this cell. Thanks to the `print` statements, five pages of quotes appear.

What would happen we ran `soup.find_all('div', class_='quote')` instead of `soup.find_all('span', class='text')` ?

- ○ We would scrape the parent element and grab everything instead of just the quotes.
- ○ We would scrape only the text from the quotes anyway.
- ○ There would be an error because scraping the entire element would return too many items.

Check Answer

Finish ▶

Finally, we want to close the browser session. To do so, enter and run the following code in the next cell:

```
browser.quit()
```

Now, test your skills by scraping another website yourself in the following Skill Drill.

**SKILL DRILL**

Text your scraping skills by visiting Books to Scrape (http://books.toscrape.com/) and scraping the book URLs from the first page.

Now that you've learned how to perform an automated web scrape across multiple pages, you'll next learn how to scrape data that exists in a table.