Web Scraping In Practice

Introduction

Now that you've gotten a brief introduction to Beautiful Soup and how to select various elements from a web page, it's time to practice scraping a website. You'll start to see that scraping is a dynamic process that involves investigating the web page(s) at hand and developing scripts tailored to those structures.

Objectives

You will be able to:

- Navigate HTML documents using Beautiful Soup's children and sibling relations
- Select specific elements from HTML using Beautiful Soup
- Use regular expressions to extract items with a certain pattern within Beautiful Soup
- Determine the pagination scheme of a website and scrape multiple pages

In [1]:
```
from bs4 import BeautifulSoup
import requests
```

Grabbing an HTML Page

To start, here's how to retrieve an arbitrary web page and load its content into Beautiful Soup for parsing. You first use the requests package to pull the HTML itself and then pass that data to beautiful soup.

In [2]:
```
html_page = requests.get('http://books.toscrape.com/') # Make a get request to retrieve the page
soup = BeautifulSoup(html_page.content, 'html.parser') # Pass the page contents to beautiful soup for parsing
```

Previewing the Structure

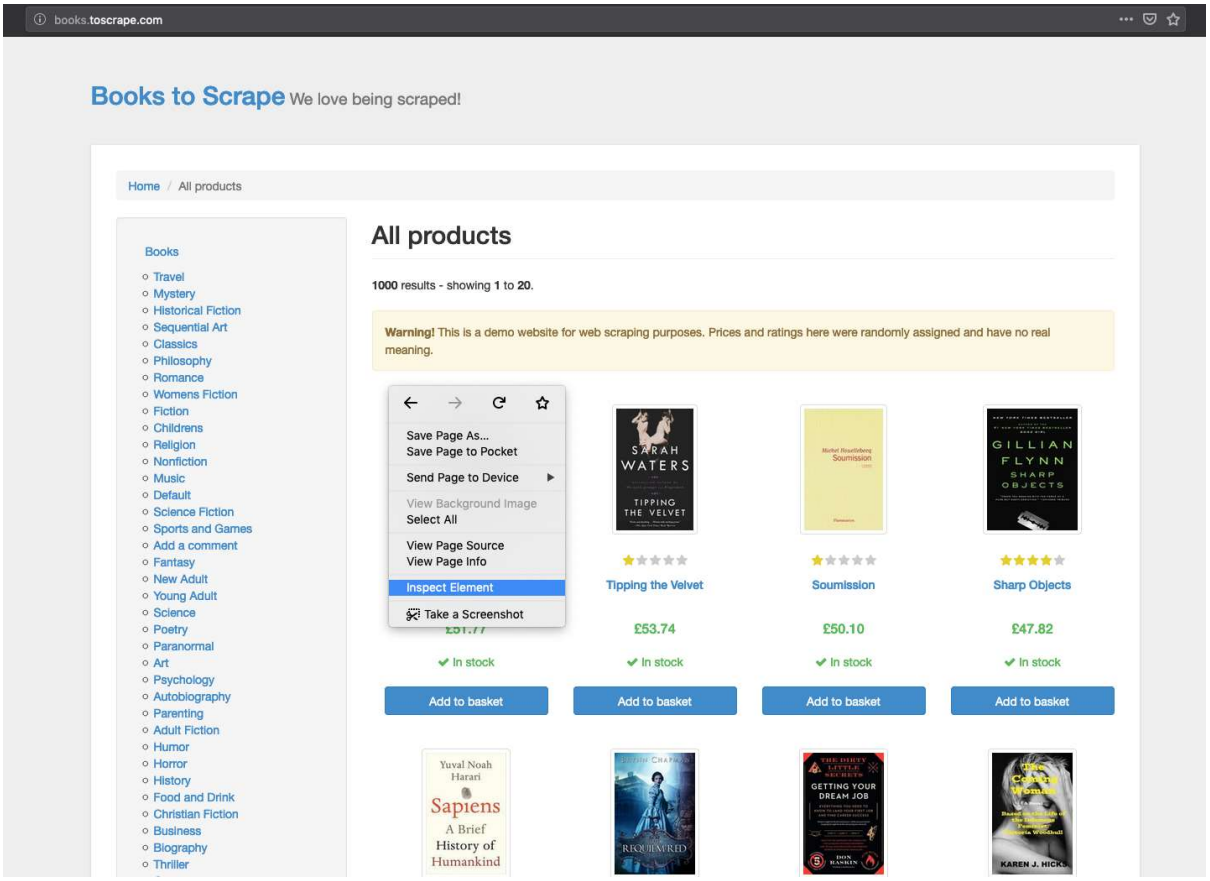
While it's apt to be too much information to effectively navigate, taking a quick peek into the structure of the HTML page is always a good idea.

In [3]:
```
soup.prettify
```

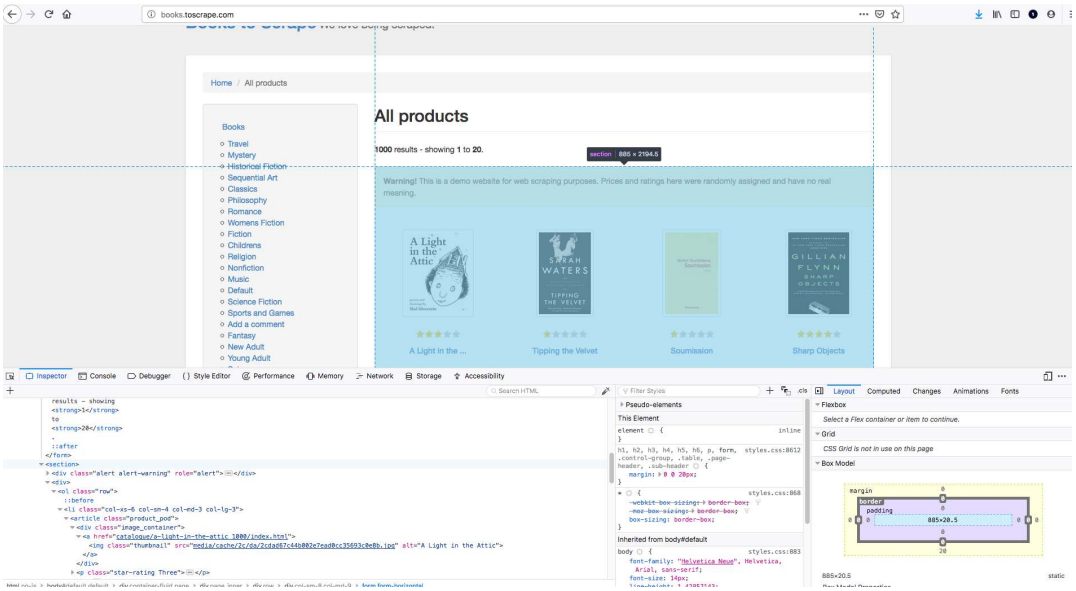
Out[3]:
```
<bound method Tag.prettify of <!DOCTYPE html>

<!--[if lt IE 7]>      <html lang="en-us" class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7]>         <html lang="en-us" class="no-js lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8]>         <html lang="en-us" class="no-js lt-ie9"> <![endif]-->
<!--[if gt IE 8]><!--> <html class="no-js" lang="en-us"> <!--<![endif]-->
<head>
<title>
    All products | Books to Scrape - Sandbox
</title>
<meta content="text/html; charset=utf-8" http-equiv="content-type"/>
<meta content="24th Jun 2016 09:29" name="created"/>
<meta content="" name="description"/>
<meta content="width=device-width" name="viewport"/>
<meta content="NOARCHIVE,NOCACHE" name="robots"/>
<!-- Le HTML5 shim, for IE6-8 support of HTML elements -->
<!--[if lt IE 9]>
        <script src="//html5shim.googlecode.com/svn/trunk/html5.js"></script>
        <![endif]-->
```

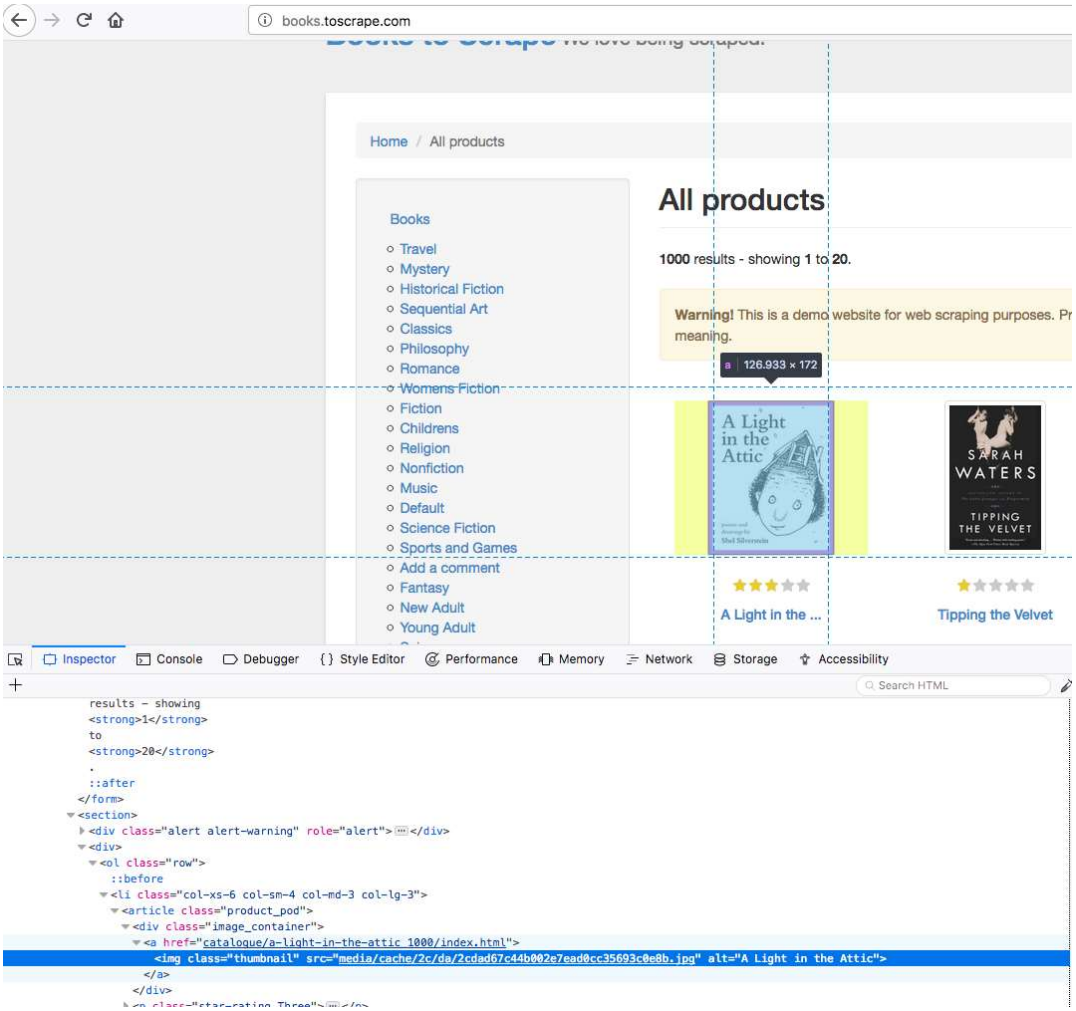
Using the Inspect Element Feature

As you can see, there's a lot going on in a production level HTML page. Rather than tediously scrolling through all of this, you'll typically have specific information you're looking to pull from a page. For example, the page you've just loaded is a mock online bookstore used for scraping practice. (As noted in the previous lesson, be careful what you attempt to scrape and at what rate/volume; many websites will quickly blacklist you if you attempt to make too many requests.) For this page, you'll see how to programmatically extract the book names, cover images, and price. Once you have a goal in mind, you can ctrl+click (Windows: right click) on the portion of the page that you're interested in and select inspect element. This will bring up the developer's portion of your web browser and allow you to preview the underlying HTML code.



This will also reveal underlying `divs` , `headers` and other containers the web designers have used to organize their web pages.





Selecting a Container

While you're eventually looking to select each of the individual books, it's often easier to start with an encapsulating container. In this case, the `section` displayed above. Once you select this container, you can then make sub-selections within it to find the relevant information you are searching for. In this case, the warning just above the div for the books is easy to identify. You can start by selecting this element and then navigating to the next div element.

In [4]:
```
warning = soup.find('div', class_="alert alert-warning")
warning # Previewing is optional but can help you verify you are selecting what you think you are
```

Out[4]:
```
<div class="alert alert-warning" role="alert"><strong>Warning!</strong> This is a demo website for web scraping purposes. Price
s and ratings here were randomly assigned and have no real meaning.</div>
```

Traversing the Soup

Now, you can navigate to the section using the next sibling method. (In actuality you need to use nextSibling twice in this case.)

In [5]:
```
# This code is a bit brittle but works for now; in general, ask, are you confident that this will work for all pages?
book_container = warning.nextSibling.nextSibling
book_container
```

Out[5]:
```
<div>
<ol class="row">
<li class="col-xs-6 col-sm-4 col-md-3 col-lg-3">
<article class="product_pod">
<div class="image_container">
<a href="catalogue/a-light-in-the-attic_1000/index.html"><img alt="A Light in the Attic" class="thumbnail" src="media/cache/2
c/da/2cdad67c44b002e7ead0cc35693c0e8b.jpg"/></a>
</div>
<p class="star-rating Three">
<i class="icon-star"></i>
<i class="icon-star"></i>
<i class="icon-star"></i>
<i class="icon-star"></i>
<i class="icon-star"></i>
</p>
<h3><a href="catalogue/a-light-in-the-attic_1000/index.html" title="A Light in the Attic">A Light in the ...</a></h3>
<div class="product_price">
<p class="price_color">£51.77</p>
<p class="instock availability">
```

Now that you have the master container with all of the books of interest, you can then search within this smaller block to extract the relevant information. If you take a look at the preview above, you should see that each of the books is referenced twice: first as a simple link via an `a` tag and then again nested within an `h3` tag. You could, therefore, select all of the `a` tags and simply extract every other block of code, although this could be brittle and prone to error. A more reliable method would be to select only the `img` tags or only the `h3` tags. As you are starting to see, web scraping is a back and forth process of investigating a page and generalizing its structure.

Generally, this is best done with a little trial and error: make a selection, preview it, and continue slicing down until you have what you're after.

In [6]:
```
titles = book_container.findAll('h3') # Make a selection
titles[0] # Preview the first entry it
```

Out[6]:
```
<h3><a href="catalogue/a-light-in-the-attic_1000/index.html" title="A Light in the Attic">A Light in the ...</a></h3>
```

Looks like you need to further slice into these `h3` tags:

In [7]:
```
titles[0].find('a')
```

Out[7]:
```
<a href="catalogue/a-light-in-the-attic_1000/index.html" title="A Light in the Attic">A Light in the ...</a>
```

Closer. Once you make it down to a single tag that's not nested, you can use the `.attrs` attribute to pull up a dictionary of the tag's attributes. In this case, you're looking for the title:

In [8]:
```
titles[0].find('a').attrs['title']
```

Out[8]:
```
'A Light in the Attic'
```

Great! Now that you've done some exploration to find what you were after, you can formalize the process and put it all together.

In [9]:
```
final_titles = [h3.find('a').attrs['title'] for h3 in book_container.findAll('h3')]
print(len(final_titles), final_titles[:5])
```

```
20 ['A Light in the Attic', 'Tipping the Velvet', 'Soumission', 'Sharp Objects', 'Sapiens: A Brief History of Humankind']
```

Passing Regular Expressions

Another useful feature is passing a regular expression (regex) into a Find statement. A regex is a sequence of characters that is used to search and match specific patterns of text. Think about the find feature of a web browser or text editor. Regex syntax is a bit complicated and you will learn all about it later. For now, try to follow along with the example below keeping in mind that the regex is matching a specific pattern of text.

Going back to our book example, you may have noticed that the star ratings for each of the books are encapsulated within a `p` tag whose class reads "star-rating ...". Let's take a look at how you could extract these features.

In [10]:
```
import re
```

In [11]:
```
regex = re.compile("star-rating (.*)")
book_container.findAll('p', {"class" : regex}) # Initial Trial in developing the script
```

Out[11]:
```
[<p class="star-rating Three">
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 </p>, <p class="star-rating One">
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 </p>, <p class="star-rating One">
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 <i class="icon-star"></i>
 </p>, <p class="star-rating Four">
```

As you can see, as before, you need to navigate a little further in order to remove the extraneous information.

In [12]:
```
star_ratings = []
for p in book_container.findAll('p', {"class" : regex}):
    star_ratings.append(p.attrs['class'][-1])
star_ratings
```

Out[12]:
```
['Three',
 'One',
 'One',
 'Four',
 'Five',
 'One',
 'Four',
 'Three',
 'Four',
 'One',
 'Two',
 'Four',
 'Five',
 'Five',
 'Five',
 'Three',
 'One',
 'One',
 'Two',
 'Two']
```

As you can see, even here we have strings whereas integers would probably be a more useful representation so you may still have to do some further data transformations.

In [13]:
```
star_dict = {'One': 1, 'Two': 2, 'Three':3, 'Four': 4, 'Five':5} # Manually create a dictionary to translate to numeric
star_ratings = [star_dict[s] for s in star_ratings]
star_ratings
```

Out[13]:
```
[3, 1, 1, 4, 5, 1, 4, 3, 4, 1, 2, 4, 5, 5, 5, 3, 1, 1, 2, 2]
```

Further Practice

You're definitely making some progress here! Let's take a look at extracting two more pieces of information: the price and availability.

In [14]:
```
book_container.findAll('p', class_="price_color") # First preview
```

Out[14]:
```
[<p class="price_color">£51.77</p>,
 <p class="price_color">£53.74</p>,
 <p class="price_color">£50.10</p>,
 <p class="price_color">£47.82</p>,
 <p class="price_color">£54.23</p>,
 <p class="price_color">£22.65</p>,
 <p class="price_color">£33.34</p>,
 <p class="price_color">£17.93</p>,
 <p class="price_color">£22.60</p>,
 <p class="price_color">£52.15</p>,
 <p class="price_color">£13.99</p>,
 <p class="price_color">£20.66</p>,
 <p class="price_color">£17.46</p>,
 <p class="price_color">£52.29</p>,
 <p class="price_color">£35.02</p>,
 <p class="price_color">£57.25</p>,
 <p class="price_color">£23.88</p>,
 <p class="price_color">£37.59</p>,
 <p class="price_color">£51.33</p>,
 <p class="price_color">£45.17</p>]
```

In [15]:
```
prices = [p.text for p in book_container.findAll('p', class_="price_color")] # Keep cleaning it up
print(len(prices), prices[:5])
```

```
20 ['£51.77', '£53.74', '£50.10', '£47.82', '£54.23']
```

In [16]:
```
prices = [float(p[1:]) for p in prices] # Removing the pound sign and converting to float
print(len(prices), prices[:5])
```

```
20 [51.77, 53.74, 50.1, 47.82, 54.23]
```

Hopefully, the process is starting to feel a bit smoother.

In [17]:
```
avails = book_container.findAll('p', class_="instock availability")
avails[:5] # Preview our selection
```

Out[17]:
```
[<p class="instock availability">
 <i class="icon-ok"></i>

         In stock

 </p>, <p class="instock availability">
 <i class="icon-ok"></i>

         In stock

 </p>, <p class="instock availability">
 <i class="icon-ok"></i>

         In stock

 </p>, <p class="instock availability">
 <i class="icon-ok"></i>

         In stock

 </p>, <p class="instock availability">
 <i class="icon-ok"></i>

         In stock

 </p>]
```

In [18]:
```
avails[0].text # Dig a little deeper into the structure
```

Out[18]:
```
'\n\n    \n        In stock\n    \n'
```

In [19]:
```
avails = [a.text.strip() for a in book_container.findAll('p', class_="instock availability")] # Finalize the selection
print(len(avails), avails[:5])
```

```
20 ['In stock', 'In stock', 'In stock', 'In stock', 'In stock']
```

Putting it All Together

Now that you have the relevant information, it's time to put it all together into a dataset!

In [20]:
```
import pandas as pd
```

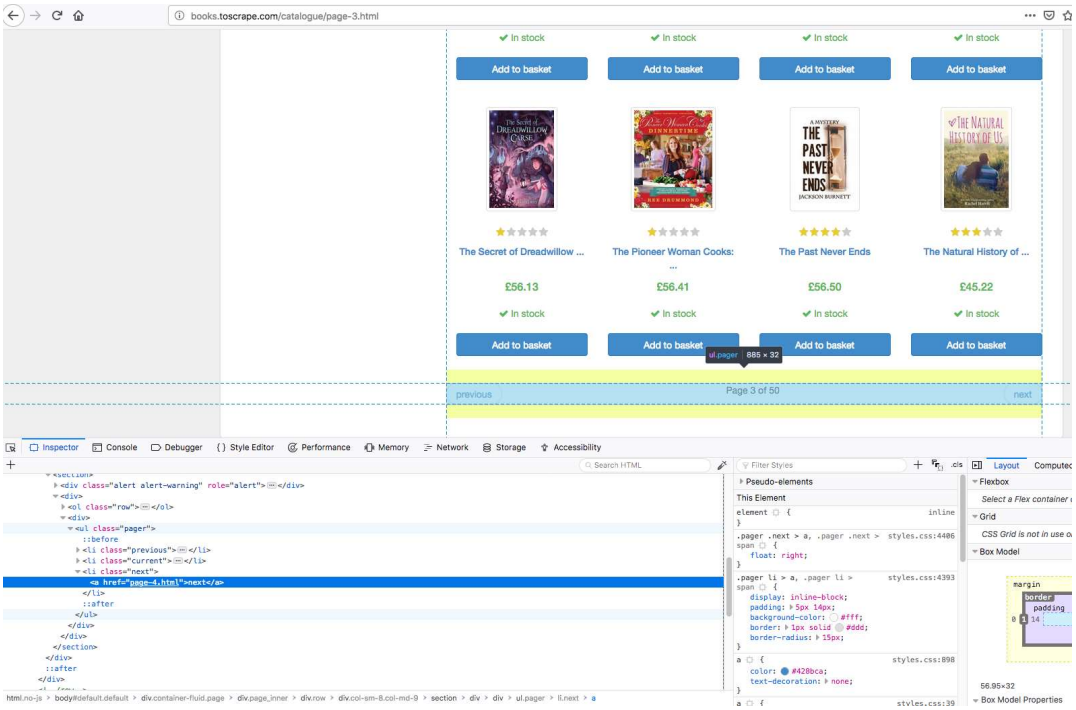
In [21]:
```
df = pd.DataFrame([final_titles, star_ratings, prices, avails]).transpose()
df.columns = ['Title', 'Star_Rating', 'Price_(pounds)', 'Availability']
df
```

Out[21]:

	Title	Star_Rating	Price_(pounds)	Availability
0	A Light in the Attic	3	51.77	In stock
1	Tipping the Velvet	1	53.74	In stock
2	Soumission	1	50.1	In stock
3	Sharp Objects	4	47.82	In stock
4	Sapiens: A Brief History of Humankind	5	54.23	In stock
5	The Requiem Red	1	22.65	In stock
6	The Dirty Little Secrets of Getting Your Dream...	4	33.34	In stock
7	The Coming Woman: A Novel Based on the Life of...	3	17.93	In stock
8	The Boys in the Boat: Nine Americans and Their...	4	22.6	In stock
9	The Black Maria	1	52.15	In stock
10	Starving Hearts (Triangular Trade Trilogy, #1)	2	13.99	In stock
11	Shakespeare's Sonnets	4	20.66	In stock
12	Set Me Free	5	17.46	In stock
13	Scott Pilgrim's Precious Little Life (Scott Pi...	5	52.29	In stock
14	Rip it Up and Start Again	5	35.02	In stock
15	Our Band Could Be Your Life: Scenes from the A...	3	57.25	In stock
16	Olio	1	23.88	In stock
17	Mesaerion: The Best Science Fiction Stories 18...	1	37.59	In stock
18	Libertarianism for Beginners	2	51.33	In stock
19	It's Only the Himalayas	2	45.17	In stock

Pagination and URL Hacking

Now that you have successfully scraped one page of books, the next logical step is to extrapolate this to successive pages. In general, the two most common approaches are to search for a button that will take you to the next page or to investigate the structure of the page URLs. For example, at the bottom of the page you should see a button like this:



As you can see, this portion contains a link to the next page of the book listings. What's more, is that you can also see that the next pages are easy to anticipate the URL for. They're simply:

- http://books.toscrape.com/catalogue/page-2.html (http://books.toscrape.com/catalogue/page-2.html)
- http://books.toscrape.com/catalogue/page-3.html (http://books.toscrape.com/catalogue/page-3.html)
- http://books.toscrape.com/catalogue/page-4.html (http://books.toscrape.com/catalogue/page-4.html)

- etc.

In more complex examples, you would simply have to use selections such as those for the title, price, star rating and availability to retrieve the URL of the next page. However, in simple cases like this, it is possible to simply hardwire the page URLs in a `for` loop. In the upcoming lab, you'll formalize this knowledge by writing a script to scrape all 50 pages from the site. The pseudo-code will look something like this:

```
df = pd.DataFrame()
for i in range(2,51):
    url = "http://books.toscrape.com/catalogue/page-{}.html".format(i)
    html_page = requests.get(url)
    soup = BeautifulSoup(html_page.content, 'html.parser')
    warning = soup.find('div', class_="alert alert-warning")
    book_container = warning.nextSibling.nextSibling
    new_titles = retrieve_titles(book_container)
    new_star_ratings = retrieve_ratings(book_container)
    new_prices = retrieve_prices(book_container)
    new_avails = retrieve_avails(book_container)
    ...
```

Summary

Well done! In this lesson, you took a look at some methods for traversing and dissecting a web page with Beautiful Soup! You also got some practice selecting specific elements from HTML and scraping multiple pages. In the upcoming lab, you'll continue to formalize this, turning the current script into modularized functions which you can then use to scrape all of the information from all 50 pages of the book listings.