Python for web scraping

Worksheet 4C: Web Scraping. Group deliverable.

Data file: none

Objectives of this exercise:

1. To practice the basic tools taught in 4A and 4B.
2. To introduce you to web scraping, a mechanism for extracting data from web pages.
3. To introduce you to Python built-in functions for downloading web pages, for pausing for a fixed amount of time, and for a few string processing tasks.
4. To introduce you to the structure of csv files
5. To get your first experience of a simple coding project, from start (scoping and designing) to finish (testing and explaining).

Deliverable: A .py file, containing your code for the question. Exactly one document per group, due on Canvas by 11:55pm on Thursday 2/10. Make sure to comment your code to make it clear what the code is doing. There is only one coding task below, but with a number of suggestions. Please read the entire document before starting work.

Honor code rules: This is a team assignment, and the expectation is that all members of the team participate collaboratively in completing it. Please work patiently with your team and ensure that everyone is learning as you proceed. Seeking help from the web is allowed. Seeking a limited amount of help from other teams is also allowed, though your final deliverable must have been created by you.

Background: In the real world, data does not always come packaged in nicely formatted csv files. There are many other sources of data available. Often, companies make data available via Application Programmer Interfaces (APIs), which make it easy to get access to data via a programming language like Python. You will see many examples of APIs during next week's lecture and exercises. Sometimes, however, the data you want is stored on a publicly accessible website but there is no official API to make it easy to get access to that data. As a result, you may have at some point found yourself copying pieces of data from a website into a spreadsheet, which is not a fun way to build a dataset! **Web scraping** is a way to use a programming language like Python to automate extracting data from a website. At a high level, web scraping involves: 1) Downloading the source code of one or more web pages; 2) Searching for the data you are interested in extracting within the web page source code; and 3) Outputting the extracted data however you want to. One common use of web scraping is to collect data about competitor product availability and pricing. Motivated by this, in this worksheet you will be asked to build a web scraper to extract data from http://books.toscrape.com, a sample e-commerce site.

Part I: Checking the website

1. In this worksheet, we will scrape the website http://books.toscrape.com for all its products. Like many e-commerce sites, products are listed across multiple pages. By clicking through the main site, you can see the first 20 products are stored on http://books.toscrape.com/catalogue/page-1.html, the next 20 products are stored on http://books.toscrape.com/catalogue/page-2.html, and so on. Have a look at the website to understand what we are going to look for.

Part II: Coding task

2. Use web scraping to create a new file called "Products.csv". This file should include the item name and price for all 1,000 items on http://books.toscrape.com as two separate columns. This will be the first time you construct a comma-separated value (CSV) file for this course. As you know from opening CSV files, CSV files are spreadsheet files that are limited to a single sheet and that don't have any styling information. They are stored in a file with the data for each row as a separate line, and commas separating the columns of data for a given row (if a data element has a comma, it's wrapped in double quotes). For instance, the following is a sample Products.csv file with two products and a header:

   ```
   Title,Price
   A Light in the Attic,51.77
   Tipping the Velvet,53.74
   ```

   To confirm that this indeed creates the spreadsheet you expect, open a file products.csv with a text editor like notepad (windows) or TextEdit (mac), type in and save these three lines, and confirm that it renders correctly when opened as a spreadsheet in Excel.

   As your deliverable, upload your code that is appropriately commented. To give a sense of the scope of the task, using only techniques from 4A plus the functions I suggest you look into below, I wrote this web scraper in about 3 dozen lines of code (plus comments)

   Suggestions:

   a. One of the first things you should do is **look at the web pages you want to process**. There are a few ways to do this. One way is to navigate to one of the pages (e.g. http://books.toscrape.com/catalogue/page-1.html) and view the page source in your web browser (how exactly you do this depends on the browser, but in many web browsers right clicking and selecting "View Page Source" will do the trick).

Another way (and one that will ultimately help you complete this task) is to download the webpage using python. The following code downloads the page and saves it to a file:

```
from urllib.request import urlopen
url = 'http://books.toscrape.com/catalogue/page-1.html'
siteAsString = urlopen(url).read().decode()
with open('sampleWebsite.html', 'w') as f_out:
   f_out.write(siteAsString)
```

A few explanatory notes are in order. The first line tells python that we want to have access to the urlopen function from the urllib.request library. Python provides many libraries that help with different coding tasks. Next week you will get a lot more practice using python libraries to get data via APIs. After the third line is run, the entire source code of the website is stored in the siteAsString string. You could run print(siteAsString) to print it out to your console, though there would be a lot of output. You will probably find it easier to write to a file and open that file. You can open the sampleWebsite.html file in Spyder (recommended) or another text editor that is well behaved in opening this sort of file (notepad on Windows works well; textedit on Mac does not always display correctly).

What exactly are we looking at? The page source you are now viewing is written in Hypertext Markup Language (HTML) and contains instructions to the web browser on how to display the webpage. When performing web scraping, we read through this HTML document to find the pieces of information we want to extract.

b. Next, **look for patterns that will help you extract data**. It can be helpful to search for pieces of data that you see on the website and that you want to extract. For instance, searching for the names of the first few products in our example website shows an interesting pattern in the web page source code that is near the product name:

```
        <h3><a href="a-light-in-the-attic_1000/index.html" title="A Light in the Attic">A Light in the ...</a></h3>
         …
        <h3><a href="tipping-the-velvet_999/index.html" title="Tipping the Velvet">Tipping the Velvet</a></h3>
         …
```

This pattern seems like it could be very useful to use in extracting the product names from the website: Every product name seems to be in a line that starts with <h3> and ends with </h3>, and no other lines in the webpage seem to have this same pattern. Remember to check out how the prices are stored in the web pages, since we also want to extract this data.

c. **Before starting any coding, brainstorm with your team** to develop your algorithm. That is, on a whiteboard or piece of paper or Word or Google doc, write out in simple English what your code will look like.

d. When you start coding, **work one step at a time and test that the step works before continuing**. By a "step," I mean a few lines of code that do something relatively simple. For instance, you might first make sure you can read in the website and store it to a file. Then you might focus on opening that file, looping through its lines, and printing each line. Then you might try to modify that code to only print lines with a product name. Then you might focus on extracting the product name from each line and only printing that instead of the whole line. Then you might extract the price. Finally, you might think about how to group the two types of data into rows of your output file.

   Importantly, you should really make sure you are completely happy with your code for processing a single product page before you try to process multiple pages. As you work to update your code to work on multiple pages, make sure it works for 2 pages before trying to run on 50 pages. This is both more efficient for you as a programmer (you won't need to wait as long to see if your run worked) and also strategically sound (you're less likely to get blocked from access by a website if you download just one or two pages).

e. As you code and work with strings, **make use of the string method documentation** at https://docs.python.org/3/library/stdtypes.html#string-methods. I found the following functions useful: split (if you choose to directly process the website text instead of writing to a file and reading that file, then this function is useful for splitting up the website text into a list of lines that you can loop through by splitting on the "\n" new line character), find (for determining whether one string is contained in another, and if so, where), in (we have seen this one a few times already), and strip (for removing whitespace from the start and end of a line before processing it any further). Also, you might find the unescape function from the html package to be useful (https://docs.python.org/3/library/html.html), for example try:

```
from html import unescape
x = "Romeo &amp; Juliet"
print(unescape(x))
```

f. When you loop through multiple files for web scraping, **be respectful of toscrape.com by not requesting pages too quickly**. You should wait at least 5 seconds between each web page download request. Python makes it easy to pause for a specified number of seconds:

```
from time import sleep
sleep(5)
```

Again, here we are using a built-in python library (time) to get access to the sleep function, which pauses execution for the specified number of seconds. You will see a lot more examples of libraries next week when we use Python to access a number of different APIs.

g. **Test your code** when you think you are done. You should first test it on a single web page and then start testing it on multiple web pages. Since running your code through all 50 pages of results might take some time (since you are pausing 5 seconds between each page request), you might add a print statement to update you on the progress of the program. Think about ways you can check that the code is behaving correctly. How many products do you expect to extract from each web page? Does the csv file you outputted look correct when you open it in Excel?

h. **Document** your code, using either the # or the triple-double quotes """.

i. **Seek help** freely---as is the norm for this class. Ask me, browse on the web, talk to others, and do whatever you need to do, but make sure you understand and learn through this process.