#DLFteach • #DLFTeach Toolkit: Lesson Plans for Digital Library Instruction

Collecting Data with Web Scraping

Zach Coble

Published on: Sep 19, 2019

DOI: [10.21428/65a6243c.4c324a94](https://doi.org/10.21428/65a6243c.4c324a94)

Session Specifics

One-shot, workshop, semester-long class, faculty collaboration, online learning, or other;

1-2 hour workshop, ideally 90 minutes.

Instructional Partners

Can be offered by one instructor, although technical helpers are always welcome. It is a technical workshop that requires some setup, so depending on the instructor's comfort with troubleshooting on the fly, additional partners may be appreciated.

Audience

Librarians, graduate students, and faculty are the target audience. Undergraduate students and other interested parties are always welcome, but in my experience they often don't have as many places to apply the skills learned here.

Curricular Context

This workshop can be offered as a standalone, although it is more effective as part of a larger suite of instructional offerings. Since it builds upon existing technical skills --e.g. knowledge of command line, understanding basic programming concepts and web page architecture-- it is helpful if participants have had opportunities to acquire these skills. Indeed, it is worth communicating to participants that, though not required, a knowledge of HTML, CSS, and basic Python (or object oriented programming) are useful prerequisites.

Learning Outcomes

1. Strengthen computational thinking skills
2. Learn concepts and basic approaches to web scraping
3. Understand how web scraping relates to academic research and collection development
4. Understand broader legal and ethical context of web scraping

Preparation & Materials

While the opening discussion of concepts can be expanded to accommodate beginners, it is helpful to inform participants that some knowledge of the command line, programming, and web page architecture is helpful.

Another option is to require participants to bring their own laptop. Since they will be installing software, they will need admin privileges on their laptop.

There are two approaches to offering this workshop, depending on the availability of a computer lab with specialized software pre-installed and the instructor's comfort with troubleshooting on the fly.

1. **Approach 1: Computer lab:** If a computer lab is available, this option provides a unified programming environment for all participants and can streamline the technical aspects of the workshop and free up time for examples and discussion. If a computer lab is available, you will need to have a text editor (e.g. TextWrangler or Sublime Text), Python 3, with the Beautiful Soup module preinstalled on all workstations. If possible, installing Jupyter Notebooks will further streamline the process and provide an even experience for all users. Naturally, instructors will want to walk through the examples in advance to identify any problems and find workarounds.
2. **Approach 2: Participants bring laptops:** While this option is more time intensive, it does have the additional benefit of allowing participants to encounter the technical obstacles involved in this work and to learn practical troubleshooting techniques they will use after the workshop. It is wise to contact participants ahead of time and ask them to install the required software (a text editor, Python 3, and Pip. See Section 6: Technical Setup in lesson plan for more details). This will help save time during the workshop, but it's usually a guarantee that not everyone will have the programs installed or they might have tried to but encountered an error message. Therefore, you should build in time to help participants install the software and troubleshoot errors. This can be time consuming and requires the instructor to be comfortable with troubleshooting any number of complications that can arise.

Session Outline

1. **Introduction**
 - Short ice breaker exercise to introduce instructor and participants
2. **Workshop Agenda**
3. **Why use web scraping?**
 - What are the legal and ethical implications?
 - Technical introduction to web scraping and setup

- Hands-on web scraping exercises

4. Why Web Scraping?

- More and more primary and secondary source material is appearing on/as websites, and so it is increasingly important for scholars and librarians to learn how to collect this material
- Initiatives such as [Collections as Data](#) underscore our shifting approach toward this area of work. Scholars are increasingly requesting data from the internet. As stewards of research material, libraries have an obligation to decide how best to collect, preserve, and provide access to this info.
- More practically, web scraping can be a valuable skill in your digital toolbox, and connects with other technical skills.
- For example, data gathered from web scraping is often messy and might need to be cleaned in [Open Refine](#). Or, web scraping might be just one step in text analysis project, and you might want to use something like the [Named Entity Recognition](#) (NER) library to extract names of people or place from your data
- Additionally, web scraping can be used to, for example, extract the latest data from the U.S. census website, which is usually displayed in tables. Coding examples 2 and 3 will work with this material.
- Another common example is newspaper articles. While it is technically possible to scrape data from a newspaper website, this is usually prohibited because it can threaten the newspaper's business model (see Section 4: Legal and Ethical Implications). Additionally, newspapers articles are often available via other means, such as library databases, or the APIs (e.g. [New York Times API](#)).
- Develop computational thinking skills. You won't leave the workshop as an expert web scraper, but you will gain a deeper understanding of the logic of computers. This understanding is perhaps more important, as it helps you see how various pieces of digital projects fit together. Developing a practical understanding of what is involved in and what resources are required for a project can be helpful in your own projects and in working with collaborators.

5. Legal and Ethical Implications?

- Just because you can scrape a website, it doesn't mean you should.
- Most sites have a terms of use policy and sometimes specifically ban or set limitations on systematic scraping. Before beginning a scraping project, you should check these terms (and follow them!) and contact the site managers if you have any questions.
- This is even more critical if you're approaching this through the lens of collection development because in this case, scraping is not just for personal research, but to provide access to a larger population.
- Again, just because you can scrape a website, it doesn't mean you should.
- Thinking through the ethical implications of a project before beginning is just as important as any technical consideration.

- Are you potentially putting others at risk if you collect or share this information? This is a particularly pertinent concern for materials dealing with controversial subject matters.
- The [Documenting the Now](#) project is an excellent example of examining the ramifications of a scraping project

6. Technical Intro to Web Scraping

- [Beautiful Soup](#) is a Python library for getting data out of web pages. This could be dates, addresses, or other information relevant to your research or project. Beautiful Soup helps you target specific data within a page, extract the data, and remove the HTML markup.
- This is where we need our computational thinking skills. We're looking at data from a webpage that's meant to be machine readable - i.e. HTML. And we need to write a program (in machine readable form) to extract this data in a more human readable form. This requires that we "see" as our computers "see" in order to understand that if, for example, we want the text of an article, that we need to write a program that extracts the data between the `<p></p>` tags.
- Once you understand the underlying rules for how webpages are created (i.e. HTML and CSS), you can start to see the patterns in how people decide to present different types of information on pages. And that's the secret to web scraping: spotting these patterns so you can efficiently extract the data that you need.

7. Technical Setup (NB: Skip this section if teaching in a computer lab)

- Make sure everyone has the correct software installed.
- Help troubleshoot if anyone needs assistance.
- Make sure everyone has Python 3 installed.
- You can check by running the command: `$ python --version`.
- If it returns something, you have it installed.
- If not, go to <https://www.python.org/downloads/>
- Make sure everyone has [Pip](#) installed
- You can check by running the command: `$ which pip`.
- If it returns something, you have it installed.
- It doesn't return anything, follow the instructions [here](#) or [here](#).
- Make sure everyone has [Virtualenv](#) installed
- You can check by running the command: `$ which virtualenv`.
- If it returns something, you have it installed.
- If not, run the command: `$ pip install virtualenv`.
- NB: You might need to add `sudo` at the beginning of the installation commands, e.g. `sudo pip install virtualenv`.
- NB: Users need admin access on their laptop to perform the 'sudo' command. If they don't have admin access, it is simplest to ask that participant to pair up with someone who does.

8. Next, create a folder somewhere on your computer (e.g. Desktop, Documents, etc.) that we will use as our working directory.

- In the command line, navigate to that folder.
- For example, to get to my folder "BS_Workshop" on the Desktop:
- `$ cd ~/Desktop/BS_Workshop`

9. Next, we'll create a virtual Python environment in this folder.

- Virtualenv is virtual environment for installing/managing Python libraries.
- As you do more programming work, you'll find that sometimes libraries conflict, and you might accidentally do things during the installation process that mess up your other projects; so, it's best to create a unique environment for each project.

10. Run the command `$ virtualenv env`.

- 'env' is an arbitrary name of the folder
- Run the command `$ source env/bin/activate`.
- This 'activates' the environment.
- Next, we will install [Beautiful Soup](#).
- Run the command: `$ python -m pip install bs4`.
- NB: You might need to run as sudo; `$ sudo python -m pip install bs4`.
- Now we are ready to begin writing our first scraping script!

11. Hands-on Web Scraping Exercise

- We're going to start with a simple exercise — grabbing each title from a Craigslist [page for barbers](#). Let's first look at the HTML for our page. You can view this by using the View Source option in your browser.

```

1489      <a href="https://newyork.craigslist.org/lgi/bar/d/1995-chevrolet-lt-tahoe/6707394545.html" class="result-image gallery"
1490      data-ids="1:00Q0Q_59cUFWcGQnd">
1491          <span class="result-price">$2500</span>
1492      </a>
1493      <p class="result-info">
1494          <span class="icon icon-star" role="button">
1495              <span class="screen-reader-text">favorite this post</span>
1496          </span>
1497          <time class="result-date" datetime="2018-10-03 15:54" title="Wed 03 Oct 03:54:00 PM">Oct 3</time>
1498      </p>
1499      <a href="https://newyork.craigslist.org/lgi/bar/d/1995-chevrolet-lt-tahoe/6707394545.html" data-id="6707394545"
1500      class="result-title hdrlnk">1995 Chevrolet LT Tahoe</a>
1501
1502      <span class="result-meta">
1503          <span class="result-price">$2500</span>
1504
1505          <span class="result-tags">
1506              pic
1507              <span class="maptag" data-pid="6707394545">map</span>
1508          </span>
1509          <span class="banish icon icon-trash" role="button">
1510              <span class="screen-reader-text">hide this posting</span>
1511          </span>
1512          <span class="unbanish icon icon-trash red" role="button" aria-hidden="true"></span>
1513          <a href="#" class="restore-link">
1514              <span class="restore-narrow-text">restore</span>
1515              <span class="restore-wide-text">restore this posting</span>
1516          </a>
1517      </span>
1518  </p>
1519 </li>
1520

```

- You can begin to see how the data is structured. We want to find the part of the code that uniquely marks the title of barter posts. If we look closely at one of the titles, we see:
- `<a href="https://newyork.craigslist.org/lgi/bar/d/1995-chevrolet-lt-tahoe/6705555544.html" data-id="6705555544" class="result-title hdrlnk">1995 Chevrolet LT Tahoe</a>`

CL new york > all new york > for sale > barter

barter << search barter

☐ search titles only
☐ has image
☐ posted today
☐ bundle duplicates
☐ include nearby areas

MILES FROM ZIP
 miles from zip

PRICE
 min max

☐ cryptocurrency ok

reset update search

\$2500

★ Oct 3 **1995 Chevrolet LT Tahoe** \$2500

12. Let's break this into parts:

- a href="https://newyork.craigslist.org/lgi/bar/d/1995-chevrolet-lt-tahoe/6705555544.html"
- This is the link to the full post. We don't want this because it's not the title.
- data-id="6705555544"
- This looks better, but it's a unique ID for a post. If we search for it, it will only return one title.
- class="result-title hdrlnk"
- This looks much better. But there are actually two tags here — "result-title" and "hdrlnk" — which one do we want to use? If you search the page, you'll find that "result-title" appears 120 times on this page. There are only 20 posts displayed on my page, so I don't want that. If I search for "hdrlnk," there are 20 results. Bingo! I would do a quick check of other posts on the page to confirm that "hdrlnk" is the unique string that will return the post's title.

13. Now, let's write a program to put this into action.

- NB: The code examples below are also available on GitHub at <https://github.com/coblezc/web scraping-workshop>
- Open a new document in your text editor and save it to your working directory. You can call it "craigslist.py." The ".py" extension is important as that indicates it should be executed as a Python

program. Below is the code with comments on what each line is doing, followed by the condensed code:

```
// First we need to call the BeautifulSoup library

from bs4 import BeautifulSoup

// Next we need to call the urllib library

// urllib is used to open the webpage and read the data in it

import urllib.request

// Create a variable called start_url and define it as the page we're scraping

start_url = 'https://newyork.craigslist.org/search/bar'

// Create a variable called html and ask urllib to get the source code from our page

html = urllib.request.urlopen(start_url).read()

// HTML code is machine readable now our program has 'read' it and we can work with it

// urllib doesn't understand HTML structure, so we'll create a variable called soup and ask
BeautifulSoup to parse our page into individual elements

soup = BeautifulSoup(html, 'html.parser')

// Get the specific element we want, in this case the post titles

titles = soup.select('.hdrlnk')

// Create a for loop that displays each of the titles

for title in titles:

// The indentation is very important!

    print (title.text)

// This will output the list of titles, and we can copy and paste this into a text file to use later
```

And here is the abbreviated code:

```
from bs4 import BeautifulSoup
```

```
import urllib.request

start_url = 'https://newyork.craigslist.org/search/bar'

html = urllib.request.urlopen(start_url).read()

soup = BeautifulSoup(html, 'html.parser')

titles = soup.select('.hdrlnk')

for title in titles:

    print (title.text)
```

Alternate Example

Also available at <https://github.com/coblezc/web scraping-workshop>

Scrape a table and export to CSV

```
from bs4 import BeautifulSoup

import urllib.request

import csv

start_url = 'https://www.census.gov/quickfacts/ks'

html = urllib.request.urlopen(start_url).read()

soup = BeautifulSoup(html, 'html.parser')

table = soup.find('tbody', attrs={'data-topic': 'Population'})

f = csv.writer(open("ks-data.csv", "w"))

rows = table.find_all('tr')

for row in rows:

    cols = row.find_all('td')

    cols = [ele.text.strip() for ele in cols]

    if len(cols) > 0:
```

```
f.writerow([cols[0], cols[1]])
```

Second Alternate Example

Also available at <https://github.com/coblezc/web scraping-workshop>

Same as above, but grab all data, introduce nested for loops

```
from bs4 import BeautifulSoup

import urllib.request

import csv

start_url = 'https://www.census.gov/quickfacts/ks'

html = urllib.request.urlopen(start_url).read()

soup = BeautifulSoup(html, 'html.parser')

tables = soup.find_all('tbody')

f = csv.writer(open("ks-data-tables.csv", "w"))

for table in tables:

    heading = table.find('th')

    heading = heading.text

    f.writerow([heading])

    rows = table.find_all('tr')

    for row in rows:

        cols = row.find_all('td')

        cols = [ele.text.encode('utf-8').strip() for ele in cols]

        if len(cols) > 0:

            f.writerow([cols[0], cols[1]])

    f.writerow(["", ""])
```

Assessment

Distribute post-workshop survey to participants. You can use your institution's preferred assessment tool if available, but you should be sure to gather feedback on the following:

- Was the workshop helpful?
- How was the pacing of the workshop?
- Did the technical material make sense?
- What parts worked well and what could be improved?
- How might you use web scraping in your work or research?
- Other comments or feedback?