Webscraping I

Peter Ganong and Maggie Shi

February 2, 2026

Table of contents I

Setup

- The scraping lectures make use the `lxml` and `requests` packages, which we forgot to include in the original `environment.yml`!
- Please `cd` to the student repo, pull the updated `environment.yml`, then run:

```
conda env update --file environment.yml --prune
```

- Then restart Python/Jupyter kernel

Skills acquired at the end of this lecture

- Be able to read basic components HTML to understand how a computer "sees" a web page
- Understand how to use `BeautifulSoup` to search through HTML
- Understand where to use AI to speed up process

A note on AI

- Pre-AI: developing a scraper used to be a *much more* time-intensive endeavor, where much of your time would be spent tweaking the exact specification to scrape what you want and not scrape what you don't want
- Now: AI can replace a lot of this human effort on pattern recognition and iterating
- Why are we still teaching this? To understand how to direct AI to code your scraper, you still need to understand underlying syntax and the scraping workflow

Introduction to Web Scraping and HTML

Roadmap:

- Intro to webscraping and HTML
- HTML structure
- Two examples: simple one and a real website
- Do-pair-share

Webscraping

- **Webscraping** uses code to systematically extract content and data from websites
- Though websites vary a lot in how they're structured and where data is located, most are constructed using a common language: **HTML**
- Each website can be converted into its underlying HTML code and then parsed with Python

Steps to building a webscraper

The steps of building a webscraper are:

1. *Manual*: inspect website's HTML to see how the info we want to extract is structured and how a computer/scraper sees the webpage
2. *Code*: download and save HTML associated with a website
3. *Code*: extract information you want to scrape from HTML

Steps to building a webscraper

The steps of building a webscraper are:

1. *Manual*: inspect website's HTML to see how the info we want to extract is structured and how a computer/scraper sees the webpage
2. *Code*: download and save HTML associated with a website
3. *Code*: extract information you want to scrape from HTML

We'll start with step 1: **learning how to inspect and read HTML**. Note you won't need to *write* HTML, just understand its structure enough so you can extract from it.

HTML structure

- **HTML**: Hypertext Markup Language
- Tells your web browser how to display the content of a web page

HTML structure

- **HTML**: Hypertext Markup Language
- Tells your web browser how to display the content of a web page
- Structure of an HTML element:

```
<name_of_tag
    attribute1 = 'value'
    attribute2 = 'othervalue'> content </name_of_tag>
```

1. **Tags**: keyword that defines what element is, such as headline text, paragraph text, link, etc.
2. **Attributes**: additional information about element (optional)
3. **Content**: text associated with that element

HTML structure

- **HTML**: Hypertext Markup Language
- Tells your web browser how to display the content of a web page
- Structure of an HTML element:

```
<name_of_tag
    attribute1 = 'value'
    attribute2 = 'othervalue'> content </name_of_tag>
```

1. **Tags**: keyword that defines what element is, such as headline text, paragraph text, link, etc.
2. **Attributes**: additional information about element (optional)
3. **Content**: text associated with that element

For scraping: we (typically) want to extract the content, and we'll use the tags and attributes as keywords to direct scraper to the content

HTML structure, cont'd

- HTML is structured hierarchically, so tags can be nested within tags

```
<tag1 attribute1 = 'value'>
    <tag2 attribute1 = 'othervalue'>
        content
    </tag2>
</tag1>
```

simple.txt example

File: simple.txt

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = ‘second'> Goodbye!</p>


</body>
</html>
```

simple.txt example

File: simple.txt

opening tag

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = ‘second'> Goodbye!</p>


</body>
</html>
```

closing tag

simple.txt example

File: simple.txt

opening tag

For the <html> tag, this is all content

closing tag

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = ‘second'> Goodbye!</p>


</body>
</html>
```

simple.txt example

File: simple.txt

Head: metadata

Body: browser content

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = ‘second'> Goodbye!</p>


</body>
</html>
```

simple.txt example

File: simple.txt

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = ‘second'> Goodbye!</p>


</body>
</html>
```

Head: metadata

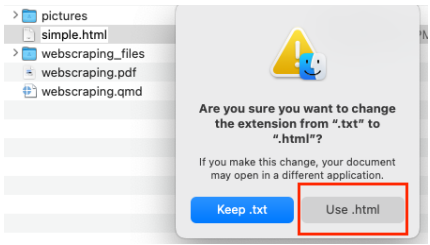
Relative to <head>, this is the content

Body: browser content

Relative to <body>, this is the content

simple.txt example

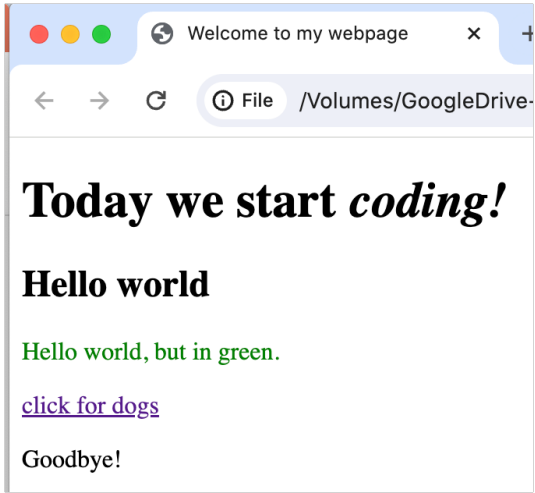
To see the HTML in action, rename the file extension from `.txt` to `.html`



Click “Use .html” when prompted

simple.txt example

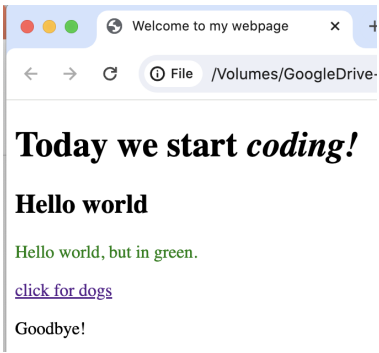
This should open as a web page in your default web browser



simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>

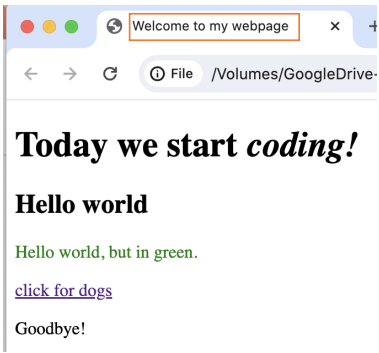



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>

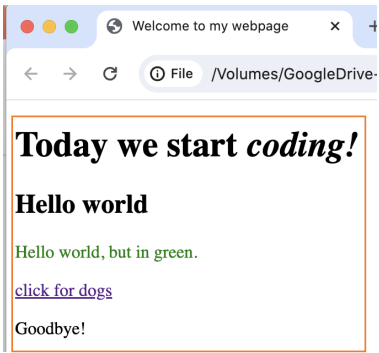



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>

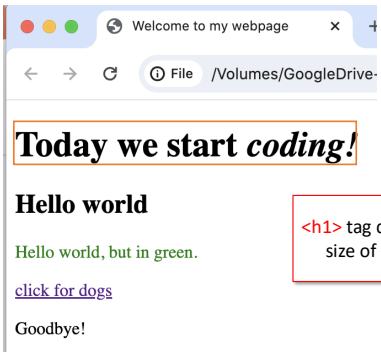



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>



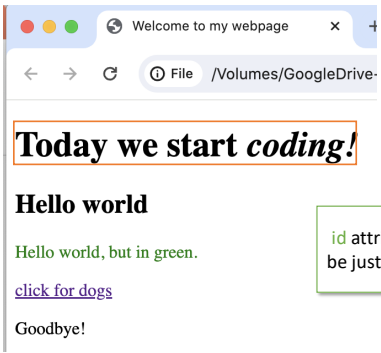
<h1> tag determines size of the text

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>



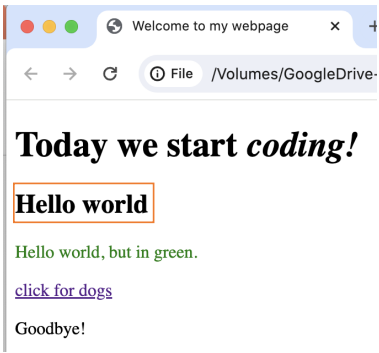

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

id attribute: should be just *one* per page

simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>

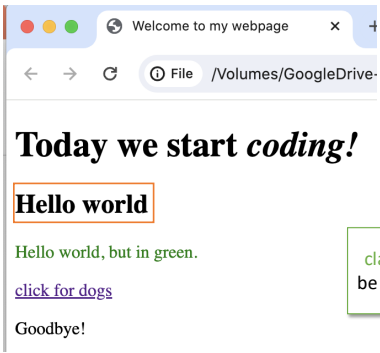



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>



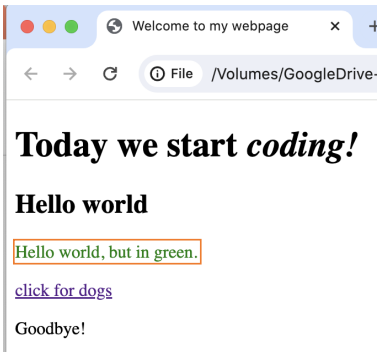

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

                          f='https://en.wikipedia.org/wiki/Dog'>click for
                          /p>
                          second'> Goodbye!</p>


</body>
</html>
```

class attribute: can be *multiple* per page

simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>

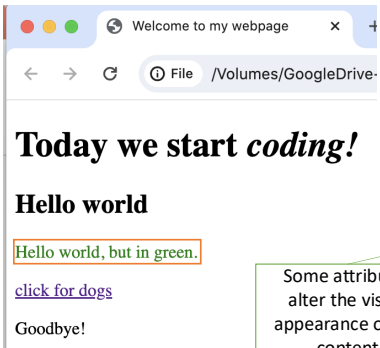



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>



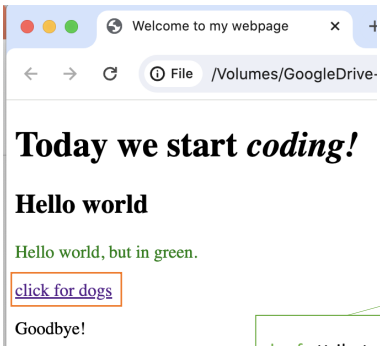

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

Some attributes alter the visual appearance of the content

simple.txt example

File: simple.txt

<name_of_tag attribute='value'>Content<\name_of_tag>



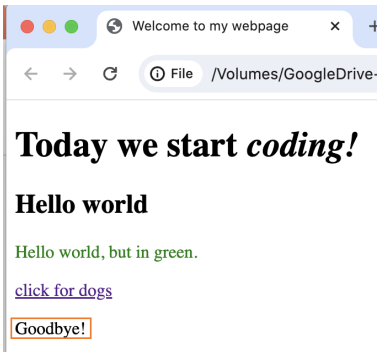

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

href attribute turns it into a hyperlink

simple.txt example

File: simple.txt

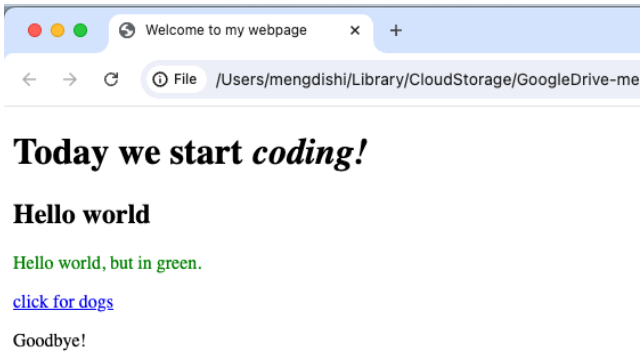
<name_of_tag attribute='value'>Content<\name_of_tag>




```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for
dogs</a></p>
<p id = 'second'> Goodbye!</p>


</body>
</html>
```

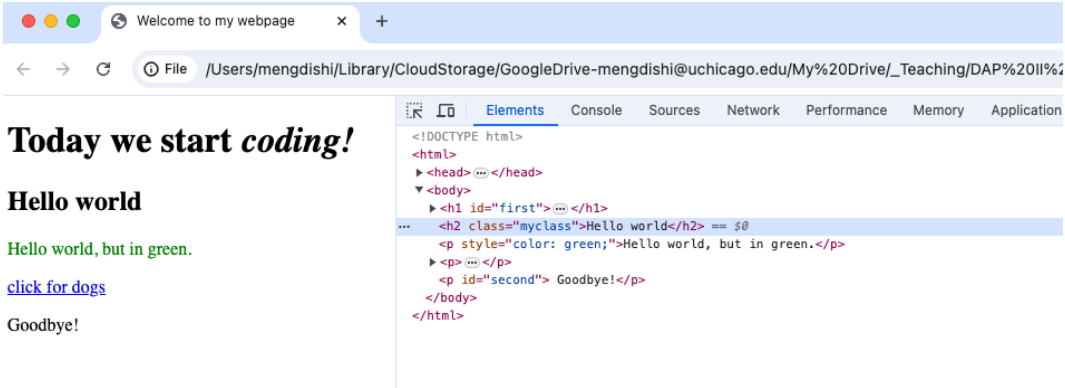
`simple.txt` example

If we open the `simple.html` file and right-click + "Inspect"...



simple.txt example

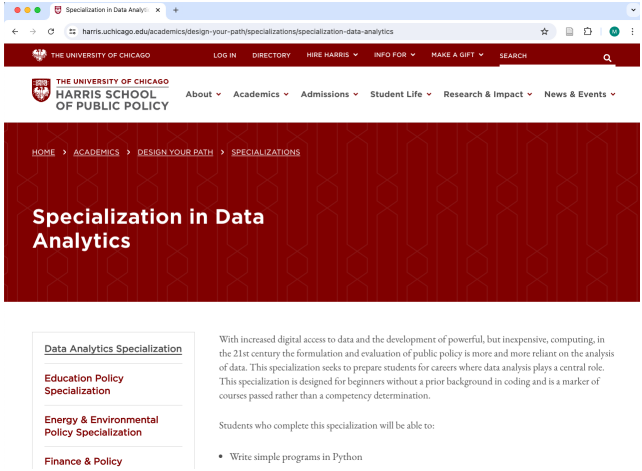
If we open the simple.html file and right-click + "Inspect"...



simple.txt example

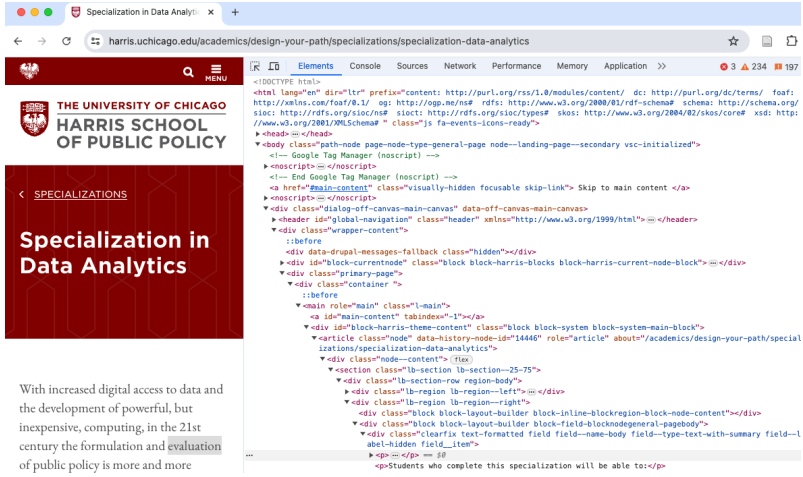
Once we expand this, we get back simple.txt!

We can “inspect” (nearly) any page on the web for its HTML



Link to demo: Harris Specialization in Data Analytics (also available as `harris_specialization.html`)

We can “inspect” (nearly) any page on the web for its HTML



Link to demo: Harris Specialization in Data Analytics (also available as `harris_specialization.html`)

Common HTML tags and attributes

- For our purposes, we don't need to know *exactly* what each tag and attribute does or means – we will just be using them as targets for scraping
- That said, there are some commonly-used HTML tags and attributes you may start to recognize as you inspect webpages

Some Common HTML Tags

HTML tags always have the structure:

```
<open> ... </close>
```

Common tags include:

- Headings: `<h1> ... </h1>`, `<h6> ... </h6>`
- Bold, italic: `<b> ... </b>`, `<i> ... </i>`
- Paragraph: `<p> ... </p>`
- Hyperlinks `<a> ... </a>`
- Images: `<img> ... </img>`

Some Common HTML Attributes

HTML attributes always have the following structure:

```
<TAG attribute = 'attributevalue'> ... </TAG>
```

Some common attributes are:

- style: `<TAG style = 'color:red;'> ... </TAG>`
- id: `<TAG id = 'idvalue'> ... </TAG>`
- class: `<TAG class = 'classname'> ... </TAG>`

Some Common HTML Tags/Attribute Combinations

- Some tags and attributes are commonly used together
- Image + source:

```
<img src = 'image.png'>... </img>
```

`img` is the tag while `src` is the attribute (source for the image file)

Some Common HTML Tags/Attribute Combinations

- Some tags and attributes are commonly used together
- Image + source:

```
<img src = 'image.png'>... </img>
```

`img` is the tag while `src` is the attribute (source for the image file)

- Links:

```
<a href = 'www.google.com'> ... </a>
```

- `a` is the tag while `href` is the attribute (URL)

Some Common HTML Tags/Attribute Combinations

One tag can also be associated with multiple attributes

```
<img src = 'image.png' width = 500 height = 600> ... </img>
```

This combines 1 tag (`img`) with 3 attributes (`src`, `width`, and `height`)

Do-pair-share

1. Inspect the HTML code on the Harris Specialization in Data Analytics (link) page
2. What is the tag associated with the text that starts with: "Students in the Master of Science in Computational Analysis and Public Policy..."?
3. What are examples of other content associated with the same tag?
4. Look at attributes for tags with links. Some of them do not appear to be "full" links. Can you explain why?

Webscraping steps

1. *Manual*: inspect website's HTML to see how the info we want to extract is structured and how a computer/scraper sees the webpage
2. *Code*: download and save HTML associated with a website
3. *Code*: extract information you want to scrape from HTML

- All web pages have a structured HTML hierarchy, which we now know how to manually parse (step 1)
- Now we can use code to do Steps 2-3!

Summary

- All websites are built in HTML
- HTML has 3 elements: tags, attributes, and content
- “Inspect” a website in your browser to view its HTML

Using `BeautifulSoup` to Parse HTML

Roadmap

- Introduce `requests` library
- Introduce `BeautifulSoup` library
- Introduce basic `BeautifulSoup` scraping syntax using `simple.txt`
- Demo how to extract URLs

Step 2: download and save HTML with `requests`

- `requests` package allows you to open webpages.
- Usually use with URLs but in this case, we'll use it to `open()` a file on disk

```
import requests
with open(r'files/simple.html', 'r') as page:
    text = page.read()
```

Preview what we saved

```
print(text)
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>

<body>
<h1 id='first'>Today we start <em>coding!</em></h1>
<h2 class='myclass'>Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>

<p><a href='https://en.wikipedia.org/wiki/Dog'>click for dogs</a></p>
<p id = 'second'> Goodbye!</p>
```

Webscraping steps

1. *Manual*: inspect website's HTML to see how the info we want to extract is structured and how a computer/scraper sees the webpage
2. *Code*: download and save HTML associated with a website
3. *Code*: extract information you want to scrape from HTML

HTML code from `simple.txt` is now in the `text` object. Step 2 - done!

Step 3: extract information you want to scrape from HTML

- We will do step 3 using the `BeautifulSoup` library: takes in HTML code and parses it in a structured way
  - *Aside: the name Beautiful Soup is a reference to poorly-structured HTML code, which is called "tag soup"*

Step 3: extract information you want to scrape from HTML

- We will do step 3 using the `BeautifulSoup` library: takes in HTML code and parses it in a structured way
  - *Aside: the name Beautiful Soup is a reference to poorly-structured HTML code, which is called "tag soup"*
- Feed `text` into `BeautifulSoup`:

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(text, 'lxml')
```

- The `soup` object is the website content, parsed by `BeautifulSoup`
- `lxml` is a parsing library to help `BeautifulSoup` parse HTML
  - You may also see `html.parser` used, which is a little slower

BeautifulSoup

```
print(soup)
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to my webpage</title>
</head>
<body>
<h1 id="first">Today we start <em>coding!</em></h1>
<h2 class="myclass">Hello world</h2>
<p style="color: green;">Hello world, but in green.</p>
<p><a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a></p>
<p id="second"> Goodbye!</p>
</body>
</html>
```

Using a `soup` object

- At first glance, `soup` object is very similar to the HTML code itself
- But it has "parsed" the code, making it **searchable** by tag and attribute

Searching a soup object: `.find_all()`

- Workhorse method of scraping is `.find_all()`: searches for and returns list of *all* instances of a particular tag
- Example: Search for each instance of the tag `p` and return it as a list

```
soup.find_all('p')
```

```
[<p style="color: green;">Hello world, but in green.</p>,
 <p><a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a></p>,
 <p id="second"> Goodbye!</p>]
```

Searching a soup object: `.find_all()`

- Workhorse method of scraping is `.find_all()`: searches for and returns list of *all* instances of a particular tag
- Example: Search for each instance of the tag `p` and return it as a list

```
soup.find_all('p')
```

```
[<p style="color: green;">Hello world, but in green.</p>,
 <p><a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a></p>,
 <p id="second"> Goodbye!</p>]
```

- A similar method: `.find()` searches for the *first* instance of an open/close tag

```
soup.find('p')
```

```
<p style="color: green;">Hello world, but in green.</p>
```

Searching a soup object: .find_all()

- We can refine our search to look for combination of tag and attribute

```
soup.find_all('p', id = "second")
```

Searching a soup object: .find_all()

- We can refine our search to look for combination of tag and attribute

```
soup.find_all('p', id = "second")
```

```
[<p id="second"> Goodbye!</p>]
```

Searching a soup object: sped up with AI

- With AI, you now don't have to write the exact search parameters yourself
- AI prompt to generate this line of code:
  "I have uploaded a `simple.html` file. Using the `BeautifulSoup` package in Python, write a `.find_all()` command that would retrieve the element that says 'Goodbye!'"

Response: (link)

```
soup.find_all('p', id='second')
```

Searching a soup object

- The output of `.find_all()` is always a *list* of objects, even if there's only one element in the list

```
soup.find_all('a')
```

```
[<a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a>]
```

- The brackets indicate that it's a list
- In contrast: `.find()` outputs just one object (the first instance)

Working with a tag object

- Specifically, `.find_all()` outputs a list of *tag* objects

```
p_tag = soup.find_all('p')[2]
p_tag
```

```
<p id="second"> Goodbye!</p>
```

Working with a tag object

- Specifically, `.find_all()` outputs a list of *tag* objects

```
p_tag = soup.find_all('p')[2]
p_tag
```

```
<p id="second"> Goodbye!</p>
```

- The `p_tag` object is aware of its tag name and attributes

```
p_tag.name
```

```
'p'
```

```
p_tag.attrs
```

```
{'id': 'second'}
```

Working with a tag object

- `h1_tag.contents` returns contents as a list. This is useful if there are tags nested inside that you want to iterate through

```
h1_tag = soup.find('h1')
h1_tag
```

```
<h1 id="first">Today we start <em>coding!</em></h1>
```

```
h1_tag.contents
```

```
['Today we start ', <em>coding!</em>]
```

Working with a tag object

- `p_tag.text` returns just the string inside, with tags removed. This is useful if you want to extract and store just the text (i.e., in a `pandas` dataframe)

```
h1_tag.text
```

```
'Today we start coding!'
```

Working with a tag objects: .get()

- Say instead of the content, we were interested in the associated *URL*, which is contained in the attribute, `href`

```
print(a_tag)
```

```
<a href="https://en.wikipedia.org/wiki/Dog">click for dogs</a>
```

- To access the URL itself, we use the `.get()` method

```
print(a_tag.get('href'))
```

```
https://en.wikipedia.org/wiki/Dog
```

- Once you've extracted the URL, you can store it and redirect the `requests` package to scrape that next.
- This forms the basis of *web crawling*, which we will discuss next lecture

Summary

- Read in HTML code from disk with `open()` from `requests`
- Then use `BeautifulSoup` parses HTML code into nested "tag" objects
- Key methods from `BeautifulSoup`:
  - `.find_all()`: return list of all tags
  - `.find()`: returns first tag
  - `.contents`, `.text` retrieve content and text
- `a_tag.get('href')` to get URLs associated with "a" tags

Example: Applying `BeautifulSoup` to a Website

Roadmap

- Apply `BeautifulSoup` methods to a real website
- Demonstrate manual and AI-assisted iterative process of tweaking your scraper

Example with Harris Website

- Now let's try pulling an actual website's HTML code and navigating it with `BeautifulSoup`
- Website of interest: Harris Specialization in Data Analytics page (link)

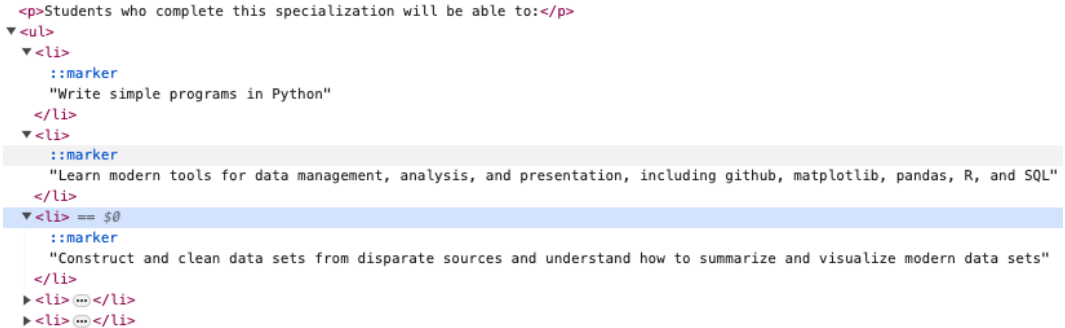
Step 1: manual inspection

Let's say we're interested in scraping all the bulleted items like the following:

- Write simple programs in Python
- Learn modern tools for data management, analysis, and presentation, including github, matplotlib, pandas, R, and SQL
- Construct and clean data sets from disparate sources and understand how to summarize and visualize modern data sets
- Use modern, computationally intensive methods to analyze data for the evaluation of policy
- The specialization's menu of electives is designed to allow students to increase their exposure to analytical methods used in the evaluation of public policy.

Step 1: manual inspection

Upon manual inspection of the HTML code, they appear to be under the `<li>` tag



Step 2: download and save HTML

- First, make a request to Harris Specialization in Data Analytics page
- Use `requests.get()` to pull HTML from a website
  - Have to specify we want `get()` from the `requests` library, not `BeautifulSoup`'s!

```
url =
 ↪  'https://harris.uchicago.edu/academics/design-your-path/specializations
response = requests.get(url)
```

Step 2: download and save HTML

- First, make a request to Harris Specialization in Data Analytics page
- Use `requests.get()` to pull HTML from a website
  - Have to specify we want `get()` from the `requests` library, not `BeautifulSoup`'s!

```
url =
 ↪  'https://harris.uchicago.edu/academics/design-your-path/specializations
response = requests.get(url)
```

- Convert into a `soup` object

```
soup = BeautifulSoup(response.text, 'lxml')
soup.text[0:50]
```

```
'\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nSpecialization in Data Analyti'
```

- Always look at what you scraped to make sure it matches what you expect

Step 3: first pass to extract info

- Use `.find_all()` and then sanity-check the output

```
tag = soup.find_all('li')
len(tag)
```

```
268
```

Step 3: first pass to extract info

- Use `.find_all()` and then sanity-check the output

```
tag = soup.find_all('li')
len(tag)
```

```
268
```

- `.find_all()` has found 266 `li` tags in the HTML code
- That is much more than the total number of bullet points we're looking for!

Step 3: first pass to extract info

- To see what's going on, we can inspect the first few elements in the `tag` object

Step 3: first pass to extract info

- To see what's going on, we can inspect the first few elements in the tag object

```
tag[0:2]
```

```
[<li class="utility-navigation__item">
 <a class="utility-navigation__item-link" data-drupal-link-system-path="us
 </li>,
 <li class="utility-navigation__item">
 <button class="js-util-nav-trigger">
         About
         <i class="fa-solid fa-chevron-right"></i>
 <i class="fa-solid fa-chevron-down"></i>
 </button>
 <div class="utility-navigation__child-container">
 <button class="js-mobile-menu-back">
 <i class="fa-solid fa-chevron-left"></i>
 <span>Home</span>
```

Step 3: first pass to extract info

- The `tag` object is picking up elements that have another tag nested in them
- But from earlier, we know the bullet points we're interested in don't have anything nested in them

Step 3: extract info, sped up with AI

> *Help me write a simple webscraper in Python using requests and BeautifulSoup. The target page is: https://harris.uchicago.edu/academics/design-your-path/specializations/specialization-data-analytics. I have uploaded the .html as well. I want to scrape the text inside the bullets. I tried soup.find_all('li') and got too many. I only want the <li> bullets where the <li> is text-only (no <a>, , etc. inside). Give me a one-line BeautifulSoup solution using `find_all()`.*

AI solution ((link)):

```
clean_bullets = soup.find_all(lambda tag: tag.name == "li" and
 ↪  tag.find(True) is None)

for li in clean_bullets:
    print(li.get_text(strip=True))
```

Step 3: extract info, sped up with AI

Explain to me the tag.find(True) is None syntax.

- AI explanation: in `tag.find(True)`, `True` is a "wildcard filter" that returns the first nested tag (of any name). If it returns None, the `<li>` is text-only.

Step 3: extract info, sped up with AI

Explain to me the tag.find(True) is None syntax.

- AI explanation: in `tag.find(True)`, `True` is a "wildcard filter" that returns the first nested tag (of any name). If it returns None, the `<li>` is text-only.

  I haven't learned get_text(strip=True)! What does that do and why do I need it?

- AI explanation: `.text` and `get_text()` are equivalent

Step 3: extract info, sped up with AI

Run and sanity-check the answer: this seems to have eliminated many of the `li` tags that we didn't want

```
len(clean_bullets)
```

16

Step 3: checking AI's work

Extract the content from this tag object into a list using `.contents`

```
clean_bullets_content = [li.contents for li in clean_bullets]
```

Step 3: checking AI's work

- Inspecting the beginning of the list ...

```
for item in clean_bullets_content[0:4]:
    print(item)
```

```
['\n                    Specialization in Data Analytics\n                ']
['Write simple programs in Python']
['Learn modern tools for data management, analysis, and presentation, inclu
['Construct and clean data sets from disparate sources and understand how
```

Step 3: checking AI's work

- Inspecting the beginning of the list ...

```
for item in clean_bullets_content[0:4]:
    print(item)
```

```
['\n                    Specialization in Data Analytics\n                ']
['Write simple programs in Python']
['Learn modern tools for data management, analysis, and presentation, inclu
['Construct and clean data sets from disparate sources and understand how
```

- and the end...

```
for item in clean_bullets_content[-2:-1]:
    print(item)
```

```
['CAPP 30300 / PPHA 30581 Data Science Clinic']
```

Step 3: cleaning up AI's work

- It seems to have largely gotten all the bullets I want
- But it also grabbed the "Specialization in Data Analytics"
- Do some final cleanup to remove the first element

```
clean_bullets_content = clean_bullets_content[1:]
for item in clean_bullets_content[0:5]:
    print(item)
```

```
['Write simple programs in Python']
['Learn modern tools for data management, analysis, and presentation, inclu
['Construct and clean data sets from disparate sources and understand how
['Use modern, computationally intensive methods to analyze data for the eva
['PPHA 30537 Data and Programming for Public Policy']
```

Example with Harris Website

Final webscraping code:

```
# Extracts and saves HTML code as a parseable object
url =
 ↪  'https://harris.uchicago.edu/academics/design-your-path/specializations/specia
response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')

# Specifies tags and attributes we want to collect
clean_bullets = soup.find_all(lambda tag: tag.name == "li" and
 ↪  tag.find(True) is None)

# "Scrapes" elements from HTML code based on step 2
clean_bullets_content = [li.contents for li in clean_bullets]

# Final cleanup
clean_bullets_content = clean_bullets_content[1:]
```

Summary: steps to building a scraper

1. *Manual*: inspect website's HTML to see how the info we want to extract is structured and how a computer/scraper sees the webpage
   - Come up with an initial guess of how to extract what you're interested in
2. *Code*: download and save HTML associated with a website
   - Use `requests.get(URL)` and then import into `BeautifulSoup`
3. *Code*: extract information you want to scrape from HTML
   - Use AI and iterate to refine your scraping code