Web scraping

2023-11-13

Agenda

- `BeautifulSoup` library
  - when to use it
  - how to use it
- robots.txt
- Ethics of web scraping

APIs vs. scraping web pages

- APIs provide structured data (usually JSON)
- The data from API requests meant for automated consumption by machines
- If an API was created, there is likely an intention to maintain (including backward compatibility)
- If you can achieve your goal with an API, **use the API**

You can't always use APIs

- Some services just don't have an API
- But they do have a website
- Sometimes there is an API, but there are other barriers:
  - Rate limit
  - Paid
  - Does not provide what **you** need
- If a webpage is served to your browser, you can scrape it

Scraping static webpages

- We still use the **requests** library
- Send a request with an **URL**
- Get an **HTML** document
- Use **BeautifulSoup** to pull necessary data out of HTML
- (With an API, we don't need BeautifulSoup because Python already understands the structure of the JSON file that gets returned)

Simlified example of HTML document

```
<!DOCTYPE html>
<html>
  <head>
    <title> Page Title </title>
  </head>
  <body>
    <h1>This is a heading </h1>
    <p> This is a paragraph </p>
  </body>
</html>
```

- **Tags** help navigate the document!
- Most information we need is inside `body`

The structure of an HTML document

- Tags are nested
- A tree-like structure with `<html>` at the root
- Common tags:
  - `<h1>`, `<h2>` ... `<h6>` denote headings
  - `<p>` is used for a paragraph
  - `<a>` is used for links
  - `<div>` and `<span>` are generic tags to group other tags
  - `<b>` for bold and `<i>` for italic
- Many others for tables, forms, buttons, etc.

The structure of an HTML document

- Tags typically need to be closed with a forward slash (/):
  - `<tag>` some content `</tag>`
- Tags can have **attributes**
  - Provide additional information about elements
  - Control the element's behavior
- They typically appear as name-value pairs:
  - `<element attribute="value">` some content `</element>`
- E.g.,
  - `<abbr id="anId" class="aClass" style="color:blue;" title="Hypertext Markup Language">HTML</abbr>`

The structure of an HTML document

- Two of the most common attributes are `<id>` and `<class>`
- `id` attribute provides a document-wide unique identifier for an element
- `class` attribute provides a way of classifying similar elements
- `href` specifies the URL of the page the link goes to
- There are many other attributes

BeautifulSoup

- BeautifulSoup represents an HTML document as a navigable tree
- Identify and navigate to specific elements using tags and attributes
- Tutorial and documentation on BeautifulSoup website
- Let's look at some parts of the tutorial

BeautifulSoup

```
html_doc = """<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
```

BeatifulSoup

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')
```

BeatifulSoup

```
print(soup.prettify())
```

```
<html>
 <head>
  <title>
   The Dormouse's story
  </title>
 </head>
 <body>
  <p class="title">
   <b>
    The Dormouse's story
   </b>
  </p>
  <p class="story">
   Once upon a time there were three little sisters; and their names were
   <a class="sister" href="http://example.com/elsie" id="link1">
    Elsie
   </a>
   ,
   <a class="sister" href="http://example.com/lacie" id="link2">
    Lacie
   </a>
   and
   <a class="sister" href="http://example.com/tillie" id="link3">
    Tillie
   </a>
   ;
and they lived at the bottom of a well.
  </p>
  <p class="story">
   ...
```

Navigating the tree

- Check the type of the soup object

```
type(soup)
```

```
bs4.BeautifulSoup
```

Navigating the tree

- Get the `title` tag
- The tag and what it contains are recognized as a Tag object

```
soup.title
```

```
<title>The Dormouse's story</title>
```

```
type(soup.title)
```

```
bs4.element.Tag
```

Navigating the tree

- Get the string (text) in the Tag object

```
soup.title.string
```

```
"The Dormouse's story"
```

- Get the name of the tag

```
soup.title.name
```

```
'title'
```

- Get the name of the parent tag

```
soup.title.parent.name
```

```
'head'
```

Navigating the tree

- If a tag occurs more than once, it'll go to first occurrence

```
print(soup.a)
print(soup.a['href'])
```

```
<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
http://example.com/elsie
```

Navigating the tree

- We can get all of them too with `find_all`
- `find_all` returns a ResultSet object where each element is a Tag
- ResultSet objects work similarly to lists (iterable, indexable/slicable)

```
rs = soup.find_all('a')
print(rs)
print(type(rs))
print(rs[0].attrs)
print(rs[0].text)
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>, <a class="sister" href="http://ex
<class 'bs4.element.ResultSet'>
{'href': 'http://example.com/elsie', 'class': ['sister'], 'id': 'link1'}
Elsie
```

Navigating the tree

- We can use attributes to navigate too
- Want neither just first element, nor all of them together?
- Remember, `id` is a document-wide unique identifier

```
soup.find(id = 'link3')
```

```
<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

Navigating the tree

- We **could** use `find_all` here and get a ResultSet
- But because we're using `id` because we know there's only one element

```
soup.find_all(id = 'link3')
```

```
[<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

- `find` will find the first element, which—in this case—is fine because there's only one element anyway

Navigating the tree

- We can `find_all` with any attribute
- `class` is a very common attribute
- But need to use `class_` argument inside `find_all`

```
soup.find_all(class_ = 'sister') # note the underscore at the end
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Navigating the tree

- `class` is used to group objects together so they can be styled together
- E.g. all objects of the same class may have the same:
  - Background color
  - Font family
  - Font size
  - Any combination of these
- Finding by class can be useful when looking for objects of the same style

Navigating the Tree

- Find by both attribute and class

```
soup.find_all('p', class_ = 'title')
```

```
[<p class="title"><b>The Dormouse's story</b></p>]
```

- Get the string in this tag

```
soup.find_all('p', class_ = 'title')[0].string
```

```
"The Dormouse's story"
```

Navigating the tree

- Use get to get the values of the attributes
- get all the hyperlinks in the page

```
for link in soup.find_all('a'):
    print(link.get('href'))
```

```
http://example.com/elsie
http://example.com/lacie
http://example.com/tillie
```

- Or, with list comprehension

```
[link.get('href') for link in soup.find_all('a')]
```

```
['http://example.com/elsie',
 'http://example.com/lacie',
 'http://example.com/tillie']
```

Navigating the tree

- Get all of the text in the page

```
print(soup.get_text())
print(type(soup.get_text()))
```

```
The Dormouse's story

The Dormouse's story
Once upon a time there were three little sisters; and their names were
Elsie,
Lacie and
Tillie;
and they lived at the bottom of a well.
...

<class 'str'>
```

More, more, and more

Go through the BeautifulSoup Documentation to learn more:

- https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Use the browser first

- Browsers like Google Chrome and Mozilla Firefox have great tools to guide your scraping
- Navigate to the web page you want to scrape
- Go to the object(s) you want to target
- Right click, then click on “Inspect”
- See which part of HTML pertains to which part of document

Use the browser first

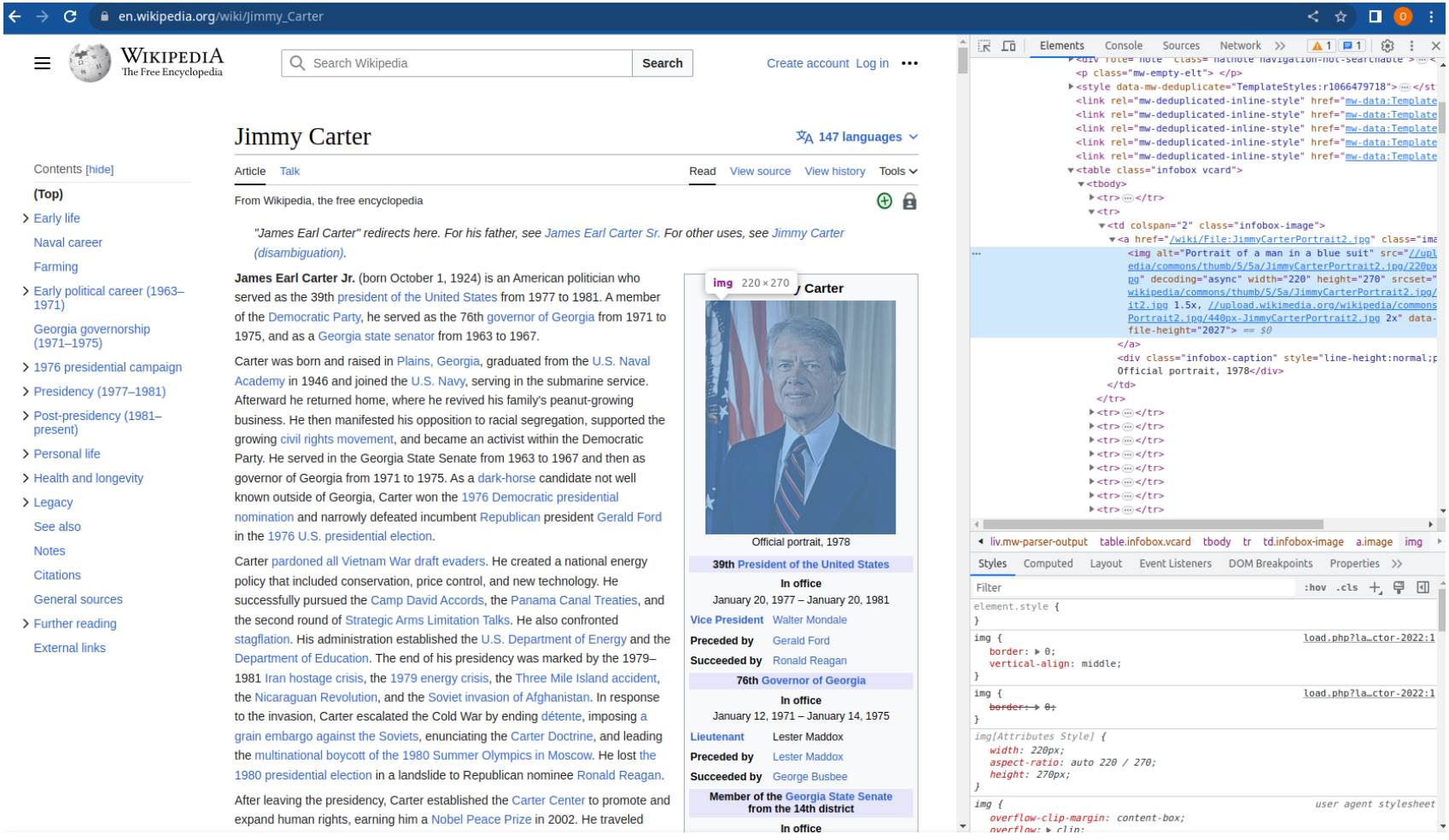

Figure 1: Google Chrome's Inspect Tool

Sleep

- To scrape a website **gently**, you can use the sleep function
- Specify how many seconds to sleep
- Place in between requests (e.g. somewhere in the for loop) to slow down
- Python will slow down before moving beyond that line
- sometimes sleeeping 1 second will be enough, sometimes 1 minute, sometimes more

```
from time import sleep
sleep(1)
```

robots.txt

- Websites communicate with scrapers using `robots.txt`
- Typically provides information on:
  - Disallowed sections of website
  - Which scrapers gets which permission
- Accessible through `URL + /robots.txt`
  - e.g. `http://www.example.com/robots.txt`

Syntax of robots.txt

- `User-agent`: it determines who is allowed or not to do the scraping
  - The `*` (read `wildcard`) means all-scrapers
- `Disallow`: denotes parts of the website that are disallowed
  - no value means everything is allowed
  - `/` means everything is disallowed
  - `/folder/` means everything in that subfolder is disallowed
- `Sitemap`: gives a list of all web pages on the site
  - it can be a list of lists (nested) that eventually branch out to all web pages

Example robots.txt

- Example from YouTube

```
# robots.txt file for YouTube
# Created in the distant future (the year 2000) after
# the robotic uprising of the mid 90's which wiped out all humans.

User-agent: Mediapartners-Google*
Disallow:

User-agent: *
Disallow: /comment
Disallow: /feeds/videos.xml
Disallow: /get_video
Disallow: /get_video_info
Disallow: /get_midroll_info
Disallow: /live_chat
Disallow: /login
Disallow: /qr
Disallow: /results
Disallow: /signup
Disallow: /t/terms
Disallow: /timedtext_video
Disallow: /verify_age
Disallow: /watch_ajax
Disallow: /watch_fragments_ajax
Disallow: /watch_popup
Disallow: /watch_queue_ajax

Sitemap: https://www.youtube.com/sitemaps/sitemap.xml
Sitemap: https://www.youtube.com/product/sitemap.xml
```

robots.txt

- It's good etiquette to respect `robots.txt`
- If not respected
  - IP address might be blocked (usually temporary)
  - Legal consequences
- LinkedIn vs. hiQ Labs