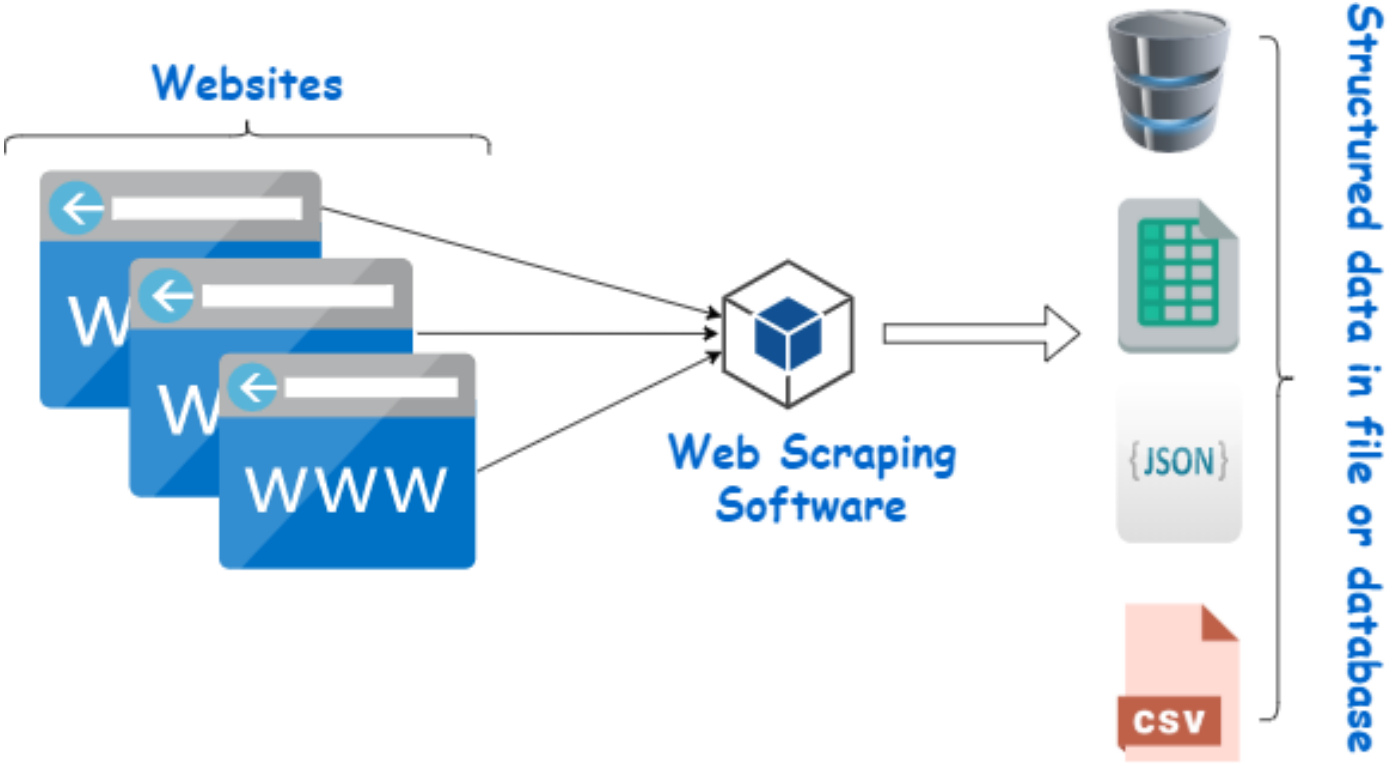
ADVANCED PYTHON PROGRAMMING

Web Scrapping

Peter Lo

Overview

- **Web Scraping** is a technique employed to extract large amounts of data from websites whereby the data is extracted and saved to a local file in your computer or to a database in table format.



Web Scraping

- Data displayed by most websites can only be viewed using a web browser.
- They do not offer the functionality to save a copy of this data for personal use.
- Web scraping is about downloading structured data from the web, selecting some of that data, and passing along what you selected to another process
- It’s a technique of extracting information from the websites.
- It extracts a large amount of data/information from the websites and saves it to a local file or table in a database or we can say that it transforms the unstructured data on to web to structured data.

Web Scraping Procedure

1. Send an HTTP request to the URL of the webpage you want to access. The server responds to the request by returning the HTML content of the webpage.
2. Once we have accessed the HTML content, we are left with the task of parsing the data. Since most of the HTML data is nested, we cannot extract data simply through string processing. One needs a parser which can create a nested/tree structure of the HTML data.
3. Now, all we need to do is navigating and searching the parse tree that we created, i.e. tree traversal.

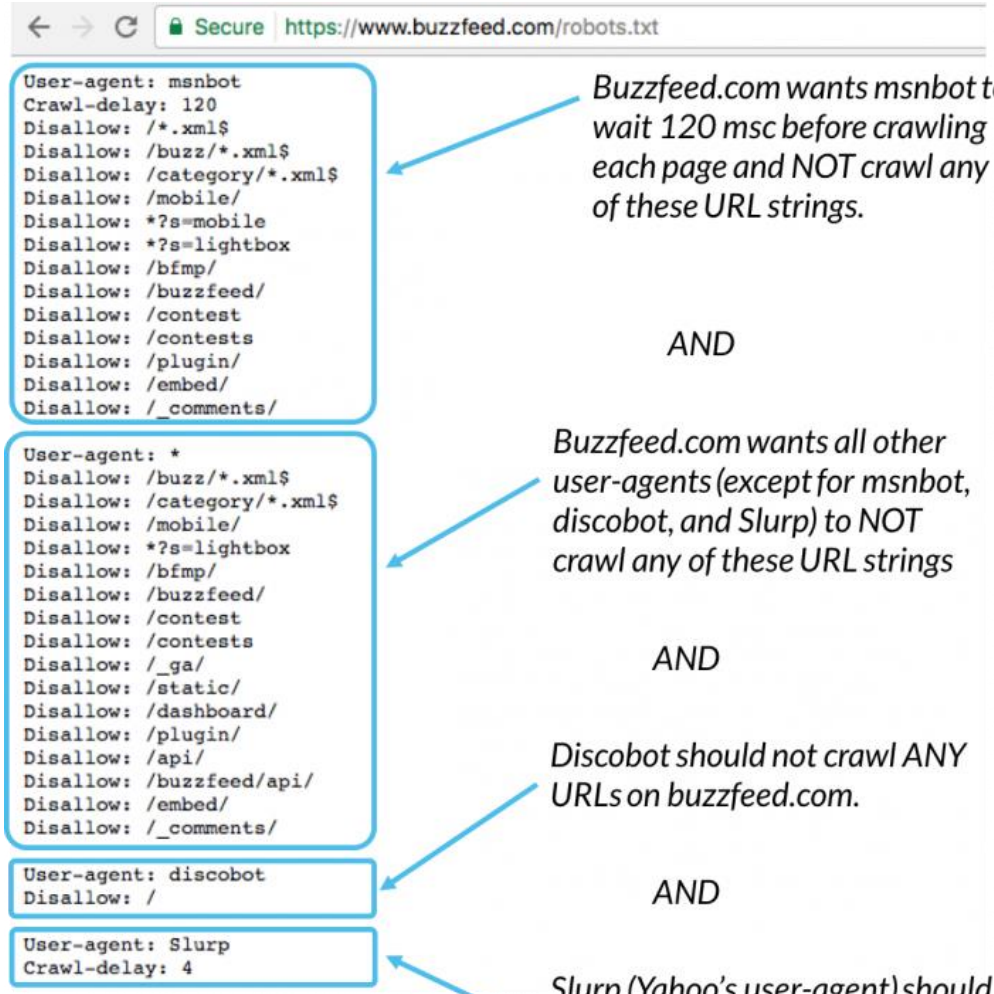
Is Web Scraping Legal?

- To better understand if a website allows scraping, your best bet is to read the ***robots.txt*** file.
- They are created by webmasters and instruct the search engine bots about how to crawl pages on the website. This includes indicators for whether specific web-crawling software is allowed or disallowed to crawl any section of the website.

Introduction to robots.txt

- Robots.txt is a text file webmasters create to instruct web robots how to crawl pages on their website.
- The robots.txt file is part of the Robots Exclusion Protocol (REP), a group of web standards that regulate how robots crawl the web, access and index content, and serve that content up to users.
- The REP also includes directives like meta robots, as well as page-, subdirectory-, or site-wide instructions for how search engines should treat links (such as “follow” or “nofollow”).
- In practice, robots.txt files indicate whether certain user agents (web-crawling software) can or cannot crawl parts of a website.
- These crawl instructions are specified by “disallowing” or “allowing” the behavior of certain (or all) user agents.

Example


Secure https://www.buzzfeed.com/robots.txt
User-agent: msnbot
Crawl-delay: 120
Disallow: /*.xml$
Disallow: /buzz/*.xml$
Disallow: /category/*.xml$
Disallow: /mobile/
Disallow: *?s=mobile
Disallow: *?s=lightbox
Disallow: /bfmp/
Disallow: /buzzfeed/
Disallow: /contest
Disallow: /contests
Disallow: /plugin/
Disallow: /embed/
Disallow: /_comments/
User-agent: *
Disallow: /buzz/*.xml$
Disallow: /category/*.xml$
Disallow: /mobile/
Disallow: *?s=lightbox
Disallow: /bfmp/
Disallow: /buzzfeed/
Disallow: /contest
Disallow: /contests
Disallow: /_ga/
Disallow: /static/
Disallow: /dashboard/
Disallow: /plugin/
Disallow: /api/
Disallow: /buzzfeed/api/
Disallow: /embed/
Disallow: /_comments/
User-agent: discobot
Disallow: /
User-agent: Slurp
Crawl-delay: 4
Buzzfeed.com wants msnbot to wait 120 msc before crawling each page and NOT crawl any of these URL strings.
AND
Buzzfeed.com wants all other user-agents (except for msnbot, discobot, and Slurp) to NOT crawl any of these URL strings
AND
Discobot should not crawl ANY URLs on buzzfeed.com.
AND
Slurp (Yahoo's user-agent) should wait 4 msc before crawling each page, but crawl all URLs on buzzfeeed.com.

Access Permission

- Allow full access - This states that all pages are crawlable by bots.

```
User-agent: *
Allow: /folder1/
```

- Block all access - This states that no part of the website can be accessed by an automated web crawler.

```
User-agent: *
Disallow: /folder2/
```

- Partial access - This declares specific sections or files on the site that are accessible and the ones that are not.

```
User-agent: *
Allow: /folder1/
Disallow: /folder2/
```

Example: Get the robots.txt

- In order to get the robots.txt, you can use the GET method from request module.

```
import requests

# Get the robots.txt
response = requests.get("https://en.wikipedia.org/robots.txt")

# Print the content
print(response.text)
```

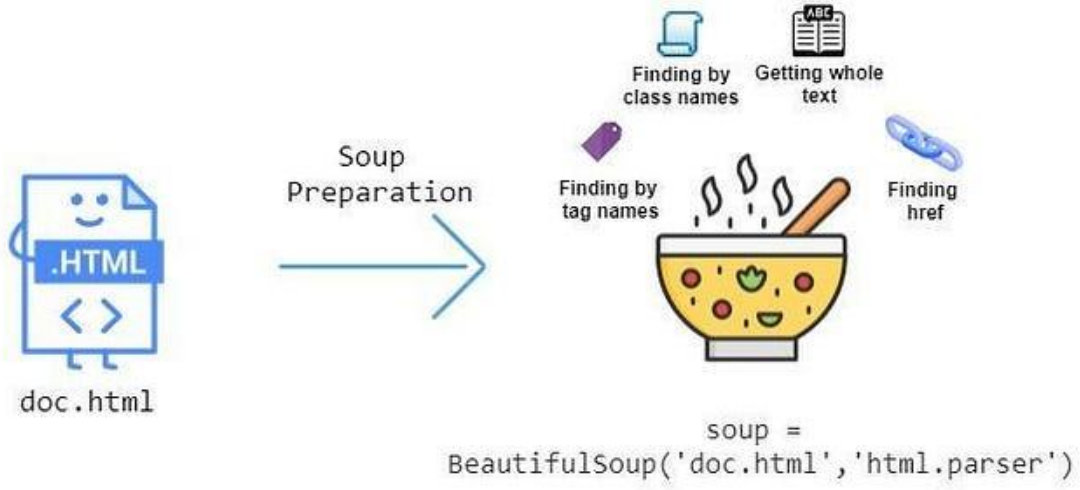
```
# robots.txt for http://www.wikipedia.org/ and friends
#
# Please note: There are a lot of pages on this site, and there are
# some misbehaved spiders out there that go _way_ too fast. If you're
# irresponsible, your access to the site may be blocked.
#

# Observed spamming large amounts of https://en.wikipedia.org/?curid=NNNNNN
# and ignoring 429 ratelimit responses, claims to respect robots:
# http://mj12bot.com/
User-agent: MJ12bot
Disallow: /
```

Popular Web Scrapping Library

Library	Ease of Use	Performance	Key Features
BeautifulSoup	Easy	Moderate	HTML/XML parsing, Easy navigation, Modifying parse tree
Scrapy	Moderate	High	Web crawling, Selectors, Built-in support for exporting data
Selenium	Easy	Moderate	Web browser automation, Supports major browsers, Record and replay actions
Requests	Easy	High	Custom headers, Session objects, SSL/TLS verification
LXML	Moderate	High	High-performance XML processing, XPath and XSLT support, Full Unicode support
MechanicalSoup	Easy	Moderate	Beautiful Soup integration

What is Beautiful Soup?

- Beautiful Soup is a Python library for pulling data out of HTML and XML files.
- It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.



Importing the Libraries

- We will import both Requests and Beautiful Soup with the import statement.
- For Beautiful Soup, we'll be importing it from bs4, the package in which Beautiful Soup 4 is found.

```
import requests
from bs4 import BeautifulSoup
```

Collecting a Web Page

- The next step we will need to do is collect the URL of the first web page with Requests.
- We'll assign the URL for the first page to the variable page by using the ***get()*** method.

```
# Get the page from URL
page = requests.get("https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ1.htm")
```

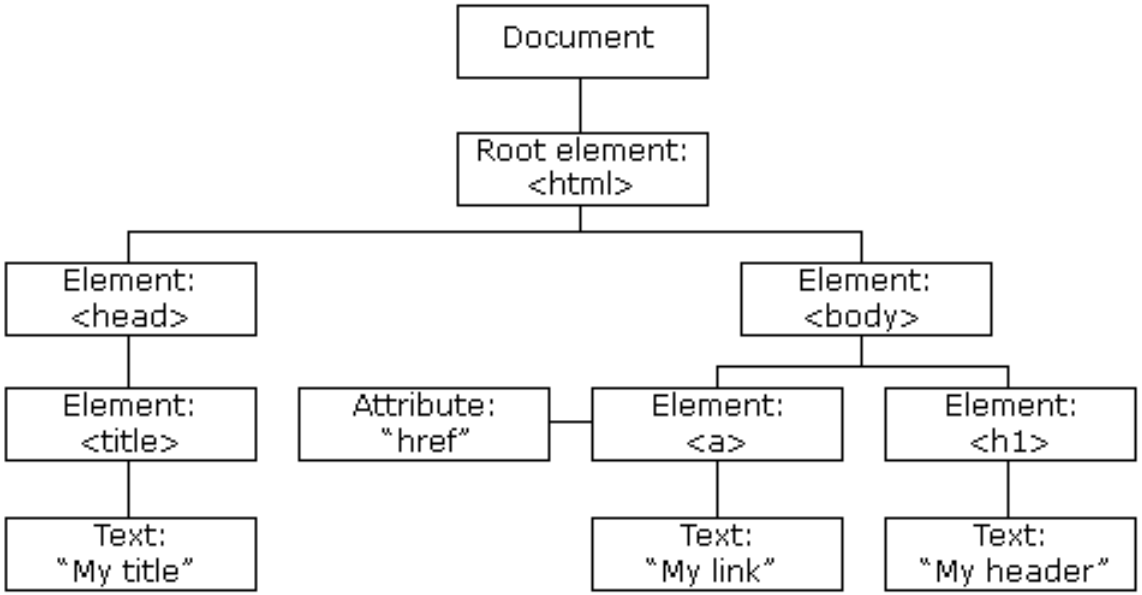
Parsing a Web Page

- We'll now create a BeautifulSoup object, or a parse tree. This object takes as its arguments the page.text document from Requests (the content of the server's response) and then parses it from Python's built-in ***parser*** method.

```
# Parse the text
soup = BeautifulSoup(page.text, "html.parser")
```

HTML Tree Traversal

- The HTML tree represents a hierarchical information view.
- The root node is the <html> tag, which can have parents, children and siblings.
- The head tag and body tag follow the HTML tag.
- The head tag contains the metadata and the title, and the body tag contains the divs, paragraphs, heading, etc.



Pulling Text From a Web Page

- We'll use Beautiful Soup's ***find()*** and ***find_all()*** methods in order to pull the text of the artists' names from the BodyText .
- Then we'll print the result with the ***prettify()*** method in order to turn the Beautiful Soup parse tree into a nicely formatted Unicode string.

```
# Parse the Artist Name List
artist_name_list = soup.find(class_="BodyText")
artist_name_list_items = artist_name_list.find_all("a")

# Pull out the <a> tag's children
for artist_name in artist_name_list_items:
    names = artist_name.contents[0]
    print(names)
```

Removing Superfluous Data

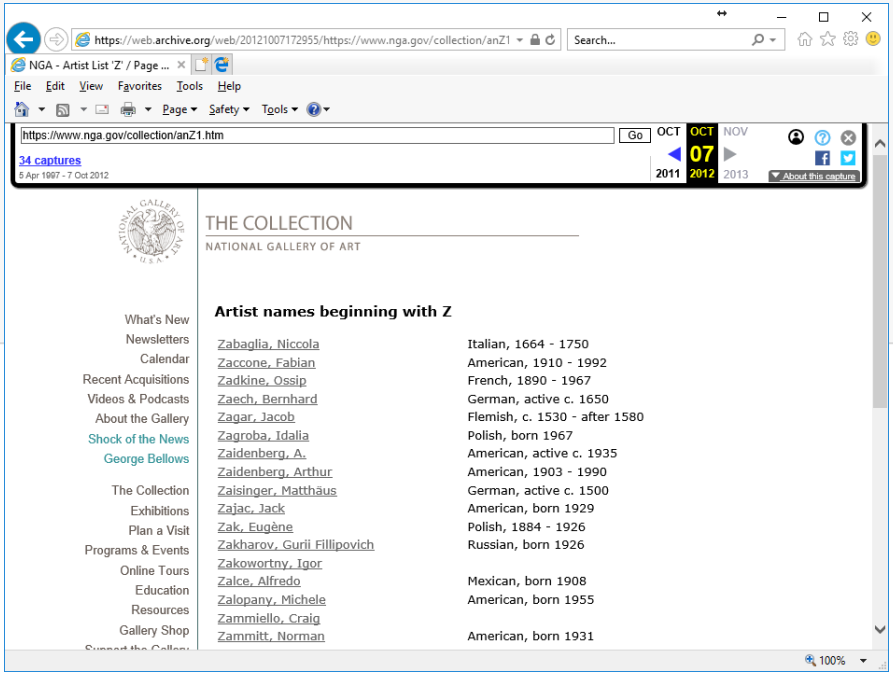
- In order to remove the bottom links of the page, we need to inspect the DOM.
- We can therefore use Beautiful Soup to find the class and use the ***decompose()*** method to remove a tag from the parse tree and then destroy it along with its contents.

```
# Remove the [Next Page] button
last_links = soup.find(class_='AlphaNav')
last_links.decompose()
```

Example

```
import requests
from bs4 import BeautifulSoup

# Get the page from URL
page = requests.get("https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ1.htm")

# Parse the text
soup = BeautifulSoup(page.text, "html.parser")

# Remove the [Next Page] button
last_links = soup.find(class_='AlphaNav')
last_links.decompose()

# Parse the Artist Name List
artist_name_list = soup.find(class_="BodyText")
artist_name_list_items = artist_name_list.find_all("a")

# Pull out the <a> tag's children
for artist_name in artist_name_list_items:
    names = artist_name.contents[0]
    print(names)
```

```
Zabaglia, Niccola
Zaccone, Fabian
Zadkine, Ossip
Zaech, Bernhard
Zagar, Jacob
Zagroba, Idalia
Zaidenberg, A.
Zaidenberg, Arthur
Zaisinger, Matthäus
Zajac, Jack
Zak, Eugène
```



MechanicalSoup

- MechanicalSoup is a python package that automatically stores and sends cookies, follows redirects, and also can follow hyperlinks and forms in a webpage.
- This Python browser automation library allows you to simulate user actions on a browser like the following:
  - Filling out forms
  - Submitting data
  - Clicking buttons
  - Navigating through pages