Python: Web Scraping

Diptesh Kanojia

e-Yantra Team
ERTS Lab, IIT Bombay

IIT Bombay
May 3, 2021

Agenda for Discussion

1. Web Scraping: The 'what' and the 'why'
   - The What!
   - The Why?
2. Web Scraping: Python's role
   - The How?
   - urllib3 Library
   - BeautifulSoup Library

What is "Web Scraping"

Web scraping is a process of automating the extraction of data in an efficient and fast way. With the help of web scraping, you can extract data from any website, no matter how large is the data, on your computer. Moreover, websites may have data that you cannot copy and paste. Web scraping can help you extract any kind of data that you want.

Motivation

- Web scraping, or web content extraction, can serve an unlimited number of purposes.
- Fortune 500 firms derive their strategies and insights from data!
- It is at the core of market research and business strategies.
- 'Data' is the new currency!
- **Question:** *Would you like to go to individual web-pages and copy-paste data?*
- In some ways, it is, metaphorically speaking, 'a way to earn' - you learn this 'well', and you may not need to look for other skills.

'How' do we start?

- The **urllib** module in Python allows you access websites via your python program.
- This opens up as many doors for your programs as the Internet opens up for you.
- This library has different versions in Python.
- Python 2.x has urllib2 & Python 3.x has urllib3
- **We will focus on urllib3 as we are focussing on Python 3.x in this course.**

The path ahead

- Through urllib, you can access websites, download data, parse data, modify your headers, and do any GET and POST requests you might need to do.
- You can change user-agents and make your program look like a browser.

Using the modules in urllib3

These are the collection of modules that you can use for working with URLs:

- `urllib.request`
- `urllib.parse`
- `urllib.error`
- `urllib.rebotparser`

urllib.request: Algorithm

- Module primarily used for opening and fetching URLs (Uniform Resource Locators)
- Sample algorithm to how your code should do:
  - import the module.
  - open a URL.
  - Now we have an HTTPResponse object .
  - Use *geturl* method to return the URL of the resource.
  - This is useful for finding out if we followed a redirect.
  - Use info to return metadata about the page, such as headers. Because of this, we assign that result to our headers variable and then call its as_string method. This prints out the header we received from your URL.
  - You can also get the HTTP response code by calling getcode, when 200, it means getcode worked successfully.
- If you'd like to see the HTML of the page, you can call the read method on the 'url' variable. Note that the request object defaults to a GET request unless you specify the data parameter.

urllib.request: Sample Code

```
#code used to make requests

import urllib.request
x = urllib.request.urlopen("https://www.google.com/")
print(x.read())

# to find out on your own: How does urllib.request.urlretrieve work?
# the "urlretrieve" method will copy a network object to a local file.
```

Protected Page 'woes'? Try this out!

```
import webbrowser
import requests

login_url = "https://www.cse.iitb.ac.in/roundcube"
payload = {"_user": "kd", "_pass": "foobar"}

s = requests.session()
response = s.post(login_url, data = payload)

print(response.url)
print(response)
webbrowser.open(response.url)
```

urllib.parse: What is it?

- Interface for breaking up URL strings and combining them back together. You can use it to convert a relative URL to an absolute URL.
- Import the urlparse function and pass it an URL that contains a search query to the duckduckgo website. My query was to look up articles on "python".
- As you can see, it returned a ParseResult object that you can use to learn more about the URL. For example, you can get the port information (None in this case), the network location, path and much more.

urllib.parse: Example

```
from urllib.parse import urlparse
result = urlparse("https://duckduckgo.com/?q=python")
print(result.netloc)
print(result.geturl())
print(result.port)
```

urllib.parse.urlencode: What if?

- Let us say we want to submit a query to duckduckgo ourselves using Python instead of a browser.
- To do that, we need to construct our query string using urlencode. Then we put that together to create a fully qualified URL.
- Then use urllib.request to submit the form. We then grab the result and save it to disk.

urllib.parse.urlencode: Example

```
import urllib.request
import urllib.parse

data = urllib.parse.urlencode({'q': "Python"})
print(data)

url = "http://duckduckgo.com/html"
full_url = url + '?' + data

response = urllib.request.urlopen(full_url)
with open("put file path and file name here", 'wb') as f:
    f.write(response.read())
```

What is BeautifulSoup?

- BeautifulSoup is a Python library to extract data from HTML files
- It is the same as urllib3 but you can 'parse' HTML a little better and extract individual components as well.
- You can even extract data from XML files using this library.
- Read up more on BeautifulSoup here.
- You can easily install library this using 'pip'
- On your command-line, execute **pip3 install beautifulsoup4**

How to use it?

- Beautiful Soup parses the given HTML document into a tree of Python objects.
- There are four main Python objects that you need to know about: Tag, NavigableString, BeautifulSoup, and Comment.
- A sample code snippet is provided below.

```
import requests
from bs4 import BeautifulSoup

req = requests.get("http://www.cse.iitb.ac.in")
soup = BeautifulSoup(req.text, 'html.parser')
print(soup.title)
```

Try on your own - *soup.find_all('a')* - and see what you get!

Navigating using BS.. bs4

You can navigate through various page elements in an HTML page using the following functions.

- Going down:
  - .contents and .children
  - .descendants
  - .string
  - .strings and stripped_strings
- Going up:
  - .parent and .parents
- Going sideways:
  - .next_sibling and .previous_sibling
  - .next_siblings and .previous_siblings
- Going back and forth:
  - .next_element and .previous_element
  - .next_elements and .previous_elements

Navigation Example Snippets

Try this:

```
head_tag = soup.head
head_tag.contents
for sstrings in head_tag.stripped_strings:
    print(sstrings)
```

Similarly, please try all the tags and try to scrape a (read: any!) web page down to just it's text.
HINT: you can also try - print(soup.get_text())

References

- Python: Basics
- W3Schools: Python
- e-Yantra Homepage

Thank You!

Author: Diptesh Kanojia
Contributor: Prashant Sharma

Post your queries at: resources@e-yantra.org