UNIT- IV

Python to access Web Data : cont'd.

- Regular Expressions
- Extracting data
- Sockets
- Using the Developer Console to Explore HTTP and Retrieving Web Page
- Parsing Web Pages

Prof. R. MADANA MOHANA
Professor, Artificial Intelligence & Data Science
http://rmadanamohana.com/

Reference

https://www.coursera.org/learn/python-network-data

https://github.com/I-am-Harsh/Using-Python-to-Access-Web-Data

https://www.py4e.com/code3/

https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_Languages/Book%3A_Python_for_Everybody_(Severance)

http://docs.python.org/library/socket.html

https://youtu.be/T0rYSFPAR0A

Parsing Web Pages / Parsing HTML (also known as Web Scraping)

What is Web Scraping?

- When a program or script pretends to be a browser and retrieves web pages, looks at those web pages, extracts information, and then looks at more web pages.
- Search engines scrape web pages - we call this “spidering the web” or “web crawling” or “web indexing”.
- web crawling is the process of automatically browsing and discovering web pages on the internet. It involves systematically following hyperlinks from one web page to another, recursively traversing through websites to collect data and build an index of the web.

What is Web Scraping?

- Web scraping refers to the automated extraction of data from websites.
- It involves writing code in a programming language, such as Python, to retrieve and parse HTML or XML content from web pages.
- Web scraping enables us to extract specific information, such as text, images, tables, or links, from websites and save it for further analysis, processing, or storage.

Why Scrape?

- Pull data - particularly social data - who links to who?
- Get our own data back out of some system that has no "export capability"
- Monitor a site for new information
- Spider the web to make a database for a search engine

Scraping Web Pages

- There is some controversy about web page scraping and some sites are a bit snippy about it.
- Republishing copyrighted information is not allowed
- Violating terms of service is not allowed
- Python is a popular language for web scraping due to its simplicity, rich ecosystem of libraries, and excellent tools for parsing HTML and XML

Scraping Web Pages

Key steps involved in a typical web scraping process:

- Sending HTTP requests: Use a library like `requests` or `urllib` to send an HTTP GET or POST request to the target website's URL.
- Retrieving the HTML content: Once the request is sent, we receive the HTML content of the web page as the response. We can access this content using the `response.text` property or similar methods.
- Parsing the HTML content: To extract data from the HTML, we need to parse it using a library like `BeautifulSoup` or `lxml`. These libraries provide functions and methods to traverse and search the HTML structure, extract specific elements, and retrieve their attributes or contents.
- `lxml` is one of the fastest and feature-rich libraries for processing XML and HTML in Python. This library is essentially a wrapper over C libraries libxml2 and libxslt (XSLT - Extensible Stylesheet Language Transformations). This combines the speed of the native C library and the simplicity of Python.

Scraping Web Pages

Key steps involved in a typical web scraping process: *cont'd.*

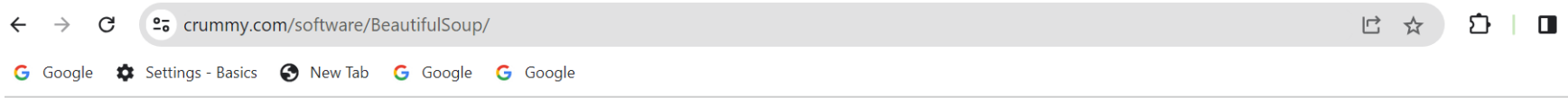
- Extracting data: Using the parsed HTML, we can apply various techniques to locate the desired data. This may involve searching for specific HTML tags, CSS (Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML) classes, or attributes. We can use methods like `find_all()` or `Xpath` (XML path language) expressions to extract the required data.
- Storing or processing the data: Once we have extracted the data, we can save it to a file, a database, or process it further according to our needs.

Scraping Web Pages

- Web scraping can be a powerful tool for gathering data from the web, automating repetitive tasks, conducting research, and building data-driven applications.
- However, it's crucial to use it responsibly and ethically, respecting the legal and ethical boundaries of web scraping.

Scraping Web Pages using BeautifulSoup

Free software library called BeautifulSoup from https://www.crummy.com/software/BeautifulSoup/



[Download | Documentation | Hall of Fame | For enterprise | Source | Changelog | Discussion group | Zine]

Beautiful Soup

You didn't write that awful page. You're just trying to get some data out of it. Beautiful Soup is here to help. Since 2004, it's been saving programmers hours or days of work on quick-turnaround screen scraping projects.

Beautiful Soup is a Python library designed for quick turnaround projects like screen-scraping. Three features make it powerful:

1. Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. It doesn't take much code to write an application
2. Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8. You don't have to think about encodings, unless the document doesn't specify an encoding and Beautiful Soup can't detect one. Then you just have to specify the original encoding.
3. Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility.

Beautiful Soup parses anything you give it, and does the tree traversal stuff for you. You can tell it "Find all the links", or "Find all the links of class `externalLink`", or "Find all the links whose urls match "foo.com", or "Find the table heading that's got bold text, then give me that text."

Valuable data that was once locked up in poorly-designed websites is now within your reach. Projects that would have taken hours take only minutes with Beautiful Soup.

BeautifulSoup Installation

- Beautiful Soup is a Python library for pulling data out of HTML and XML files.
- It works with our favorite parser to provide native ways of navigating, searching, and modifying the parse tree.
- It commonly saves programmers hours or days of work.
- The latest Version of Beautifulsoup is v4.9.3 as of now.

BeautifulSoup Installation

- To run this, we can install BeautifulSoup

```
pip install beautifulsoup4
```

- Import bs4 from BeautifulSoup

```
from bs4 import BeautifulSoup
```

BeautifulSoup - Example

webscrap.py - C:/Users/rmmna/AppData/Local/Programs/Python/Python310/webscrap.py (3.10.0)

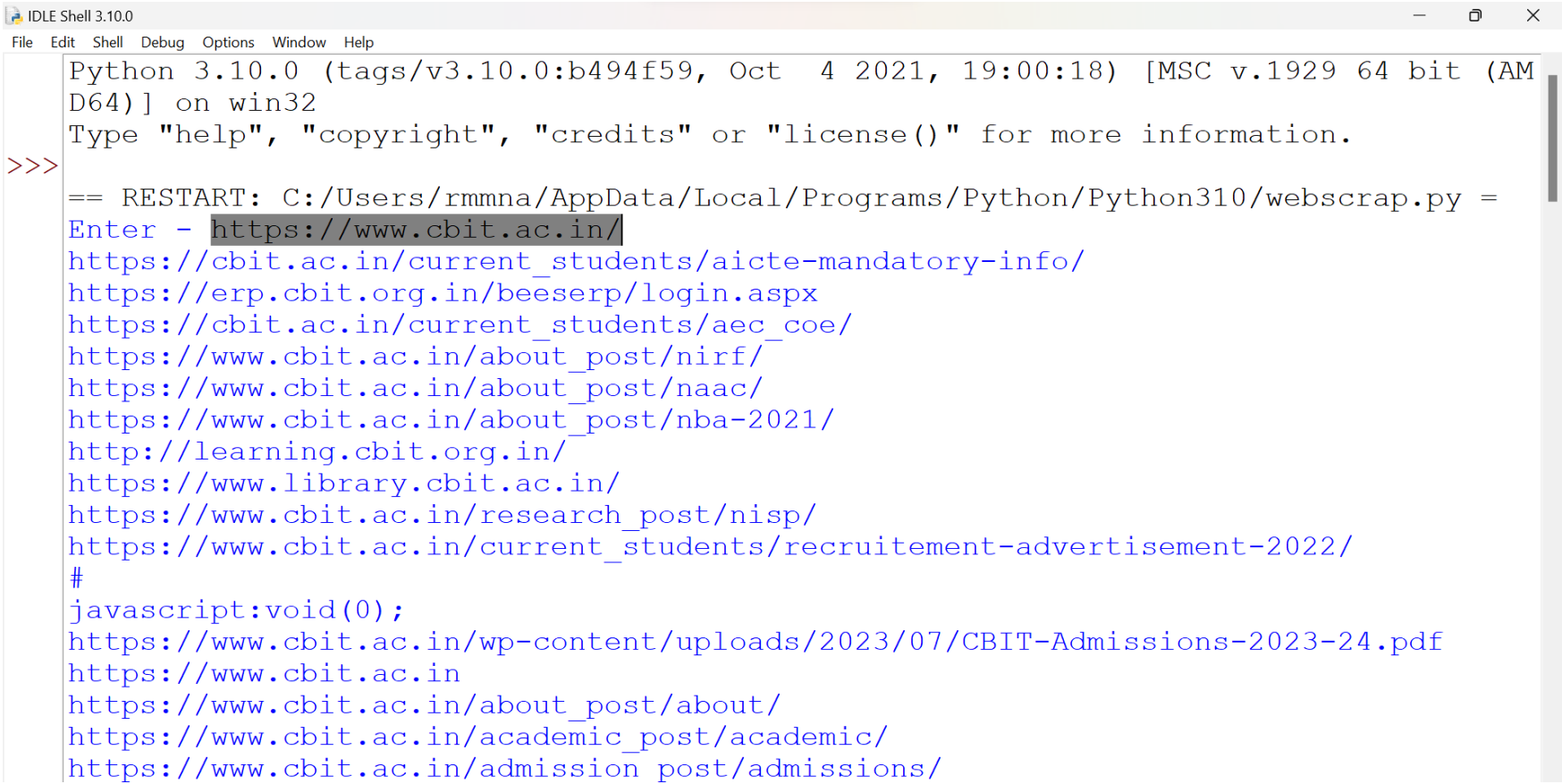
File Edit Format Run Options Window Help

```
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup

url = input('Enter - ')
html = urllib.request.urlopen(url).read()
soup = BeautifulSoup(html, 'html.parser')

# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

BeautifulSoup - Example



Lab Experiment

Create a Python project to get the citation from Google scholar using title and year of publication and volume and pages of journal

google_citations.py - C:\Users\rmmna\AppData\Local\Programs\Python\Python310\google_citations.py (3.10.0)

File Edit Format Run Options Window Help

```
import requests
from bs4 import BeautifulSoup
def get_citation(title, year, volume, pages):
    # Format the search query
    query = f"{title} {year} {volume}:{pages}"
    query = query.replace(' ', '+')
    url = f"https://scholar.google.com/scholar?q={query}"
    # Send a GET request to Google Scholar
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    # Find the first search result
    result = soup.find('div', {'class': 'gs_ri'})
    if result:
        # Extract the citation text
        citation = result.find('div', {'class': 'gs_a'}).text
        return citation
    else:
        return "Citation not found"
# Example usage
article_title = "CCBIR: A cloud based implementation of content based image retrieval"
article_year = "2015"
article_volume = "14"
article_pages = "333-346"
citation = get_citation(article_title, article_year, article_volume, article_pages)
print(citation)
```

Lab Experiment

Create a Python project to get the citation from Google scholar using title and year of publication and volume and pages of journal