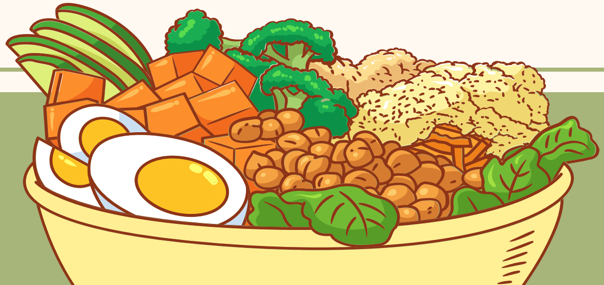
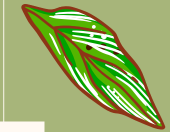
Recipe Generator

Aviva Munshi
Nizan Howard
Saumya Nauni





Background

Topic: Recipe Generator

Goal: Build a Streamlined Web App that generates a recipe from collection of recipes

- Input: List of ingredients you have at home.
- Output: A random recipe that uses only the ingredients listed.

Application:

Tool: Generate a recipe from ingredients provided upon input

Use Case

Use Case:

Have a collection of recipes to generate a recipe upon output

Overall Objective:

User receives a recipe as output that includes the ingredients they provide upon input .

Python Library needs:

A library package that can scrape text from web pages

Purpose:

Web scrape recipes from the internet to create a collection of recipes to generate

Python Package Choices 1 & 2

BeautifulSoup

Author: Leonard Richardson

Brief Summary: A python parser library for data collection. Utilizes web scraping for XML HTML, and other markup files. Outgoing information is automated as UTF-8, while the incoming data is transitioned into Unicode.

Scrapy

Author: Zyte

Brief Summary: Open-source tool that enables people to perform data collection, web crawling, data mining, performing testing automation, as well as other web-based tasks. Scrapy is Application Programming Interface (API)-based and can be used as a framework for building tailored web spiders.

BeautifulSoup

Pros:

- Beginner friendly framework and documentation
- Suitable for extracting specific elements from target web pages
- Automatic encoding conversions, provides extraction from badly written webpages

Cons:

- Requires many dependencies, can’t work on its own
- Can be slow with other dependencies used, not suited for large frameworks
- Minimal amount of proxy support , can get blocked when extracting large amounts of data from the same server.

Scrapy

Pros:

- Scrapy is a full-suite framework for extracting data.
- Streamlines the error-handling process
- Executes multiple requests simultaneously.
- Allows you to post-process any data.

Cons:

- Can’t handle JavaScript.
- Complicated installation process.
- Light documentation for beginners.

Summary of Comparison

Beautiful Soup is a 'parser' whereas Scrapy is a 'crawler'

- With Beautiful Soup, you need to provide a specific url, and Beautiful Soup will help you get the data from that page. You can give Scrapy a start url, and it will go on, *crawling* and extracting data, without having to explicitly give it every single URL.

Package Choice Chosen Reasoning

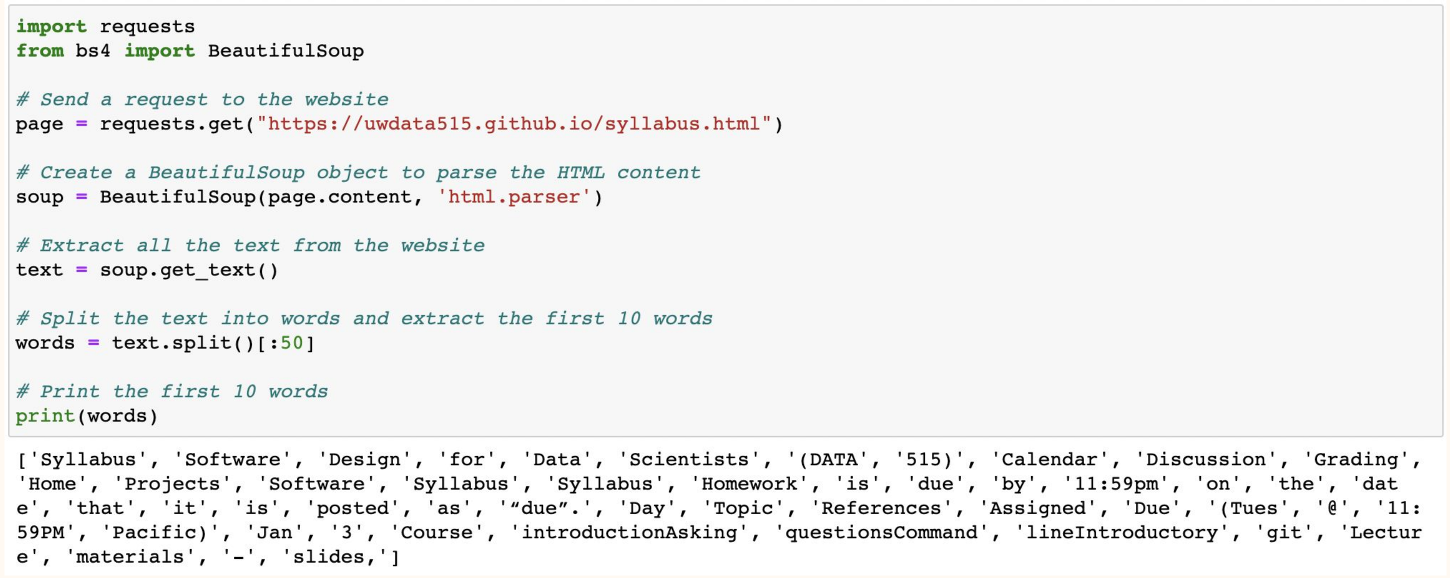
- Beautiful Soup is a Python library that provides simple methods for navigating and searching HTML and XML documents.
- Lightweight package that is easy to install and use, even for those who are new to web scraping.
- Can handle most HTML and XML documents and can extract specific elements from target web pages with ease.
- Can automatically convert encodings, making it easy to extract data from web pages with different character sets.
- Popular choice for web scraping because of its beginner-friendly interface and wide range of use cases.

Drawbacks/Remaining Concerns

- Beautiful Soup is not designed to handle complex web scraping projects that require advanced features, such as JavaScript rendering or complex request handling.
- While it can extract data from poorly written web pages, it may struggle with complex or highly dynamic web pages that require more advanced scraping techniques.
- Relies on several dependencies and may not work on its own, which can make it more complicated to set up and use.
- When scraping large amounts of data from the same server, it may encounter issues with IP blocking or server throttling, and it provides minimal support for proxy rotation.
- Can be slower than other packages when working with large frameworks or dependencies, which can impact performance in some use cases.

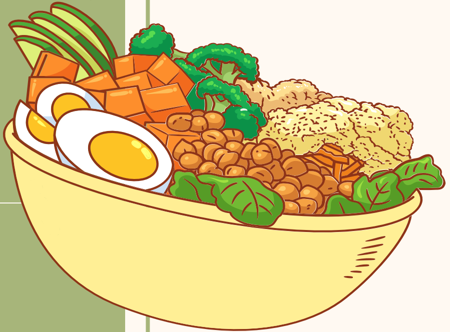
Demo of Package

Using BeautifulSoup to Extract Text from our Syllabus Web Page in Python



```python
import requests
from bs4 import BeautifulSoup

# Send a request to the website
page = requests.get("https://uwdata515.github.io/syllabus.html")

# Create a BeautifulSoup object to parse the HTML content
soup = BeautifulSoup(page.content, 'html.parser')

# Extract all the text from the website
text = soup.get_text()

# Split the text into words and extract the first 10 words
words = text.split()[:50]

# Print the first 10 words
print(words)
```

```
['Syllabus', 'Software', 'Design', 'for', 'Data', 'Scientists', '(DATA', '515)', 'Calendar', 'Discussion', 'Grading',
'Home', 'Projects', 'Software', 'Syllabus', 'Syllabus', 'Homework', 'is', 'due', 'by', '11:59pm', 'on', 'the', 'dat
e', 'that', 'it', 'is', 'posted', 'as', '“due”.', 'Day', 'Topic', 'References', 'Assigned', 'Due', '(Tues', '@', '11:
59PM', 'Pacific)', 'Jan', '3', 'Course', 'introductionAsking', 'questionsCommand', 'lineIntroductory', 'git', 'Lectur
e', 'materials', '-', 'slides,']
```



Thanks!

Do you have any questions?