Scrapy

Introduction to web crawling libraries in Python

Gregory Widmaier

Background

Scrapy project architecture is built around ‘spiders’, which are self-contained crawlers (as discussed in lecture 1) that are given a set of instructions. Following the spirit of other don’t repeat yourself frameworks.

Some well-known companies and products using Scrapy are: Lyst, CareerBuilder, Parse.ly, Sayone Technologies, Data.gov.uk’s World Government Data site.

What can Scrapy do?

- Built-in support for selecting and extracting data from HTML/XML sources using extended CSS selectors and XPath expressions, with helper methods to extract using regular expressions.
- An interactive shell console (IPython aware) for trying out the CSS and XPath expressions to scrape data, very useful when writing or debugging your spiders.
- Wide range of built-in extensions and middlewares for handling:
  - cookies and session handling
  - HTTP features like compression, authentication, caching
  - user-agent spoofing
  - **robots.txt**
  - crawl depth restriction

Easy Setup

1. Pip install scrapy
2. scrapy startproject <PROJECT NAME>
   a. __init__.py items.py settings.py
      __pycache__ pipelines.py spiders/
3. scrapy genspider <SCRAPER NAME> <START URL>
4. scrapy crawl <PROJECT NAME>

Example

I ran the commands in the previous slide with rowan as the project and the homepage as the URL. This .py script was created by scrapy and I just added the last two lines to output the results.

Rowan.py

```
1  # -*- coding: utf-8 -*-
2  import scrapy
3
4
5  class RowanSpider(scrapy.Spider):
6      name = 'Rowan'
7      allowed_domains = ['https://www.rowan.edu/home/']
8      start_urls = ['https://www.rowan.edu/home/']
9
10     def parse(self, response):
11         with open("rowan.html", 'wb') as file:
12             file.write(response.body)
13
```