Scrapy

Scrapy is a framework that you can use for crawling websites and extracting (i.e. scraping) data. It can also be used to extract data via a website's API or as a general purpose web crawler. To install Scrapy, all you need is pip:

```
pip install scrapy
```

According to Scrapy's documentation, you will also need lxml and OpenSSL installed. We are going to use Scrapy to do the same thing that we used BeautifulSoup for, which was scraping the title and link of the articles on my blog's front page. To get started, all you need to do open up a terminal and change directories to the one that you want to store our project in. Then run the following command:

```
scrapy startproject blog_scraper
```

This will create a directory named **blog_scraper** in the current directory which will contain the following items:

> - Another nested blog_scraper folder
> - scrapy.cfg

Inside of the second blog_scraper folder is where the good stuff is:

> - A spiders folder
> - __init__.py
> - items.py
> - pipelines.py
> - settings.py

We can go with the defaults for everything except **items.py**. So open up **items.py** in your favorite Python editor and add the following code:

```
import scrapy


class BlogScraperItem(scrapy.Item):
    title = scrapy.Field()
    link = scrapy.Field()
```

What we are doing here is creating a class that models what it is that we want to capture, which in this case is a series of titles and links. This is kind of like SQLAlchemy's model system in which we would create a model of a database. In Scrapy, we create a model of the data we want to scrape.

Next we need to create a spider, so change directories into the **spiders** directory and create a Python file there. Let's just call it **blog.py**. Put the following code inside of your newly created file:

```
from scrapy.spider import BaseSpider
from scrapy.selector import Selector
from ..items import BlogScraperItem


class MyBlogSpider(BaseSpider):
    name = 'mouse'
    start_urls = ['http://blog.pythonlibrary.org']

    def parse(self, response):
        selector = Selector(response)
        blog_titles = selector.xpath("//h1[@class='entry-title']")
        selections = []

        for data in blog_titles:
            selection = BlogScraperItem()
            selection['title'] = data.xpath("a/text()").extract()
            selection['link'] = data.xpath("a/@href").extract()
            selections.append(selection)

        return selections
```

Here we just import the **BaseSpider** class and a **Selector** class. We also import our **BlogScraperItem** class that we created earlier. Then we subclass BaseSpider and name our spider **mouse** since the name of my blog is The Mouse Vs the Python. We also give it a start URL. Note that this is a list which

means that you could give this spider multiple start URLs. The most important

piece is our **parse** function. It will take the responses it gets from the website and parse them.

Scrapy supports using CSS expressions or XPath for selecting certain parts of an HTML document. This basically tells Scrapy what it is that we want to scrape. XPath is a bit harder to read, but it's also the most powerful, so we'll be using it for this example. To grab the titles, we can use Google Chrome's Inspector tool to figure out that the titles are located inside an **h1** tag with a class name of **entry-title**.

The selector returns an a **SelectorList** instance that we can iterate over. This allows us to continue to make xpath queries on each item in this special list, so we can extract the title text and the link. We also create a new instance of our BlogScraperItem and insert the title and link that we extracted into that new object. Finally we append our newly scraped data into a list which we return when we're done.

To run this code, go back up to the top level folder which contained the nested blog_scraper folder and config file and run the following command:

```
scrapy crawl mouse
```

You will notice that we are telling Scrapy to crawl using the **mouse** spider that we created. This command will cause a lot of output to be printed to your screen. Fortunately, Scrapy supports exporting the data into various formats such as CSV, JSON and XML. Let's export the data we scraped using the CSV format:

```
scrapy crawl mouse -o articles.csv -t csv
```

You will still see a lot of output generated to stdout, but the title and link will be saved to disk in a file called **articles.csv**.

Most crawlers are set up to follow links and crawl the entire website or a series of websites. The crawler in this website wasn't created that way, but that would be a fun enhancement that you can add on your own.

Wrapping Up

Scraping data from the internet is challenging and fun. Python has many libraries that can make this chore quite easy. We learned about how we can use BeautifulSoup to scrape data from a blog and from Twitter. Then we learned about one of the most popular libraries for creating a web crawler / scraper in Python: Scrapy. We barely scratched the surface of what these libraries can do, so you are encouraged to spend some time reading their respective documentation for further details.