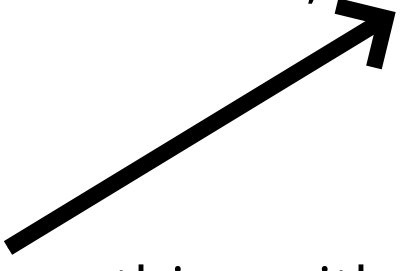
WEB SCRAPING

Objectives

- Define what web scraping is and the issues surrounding it
- Use the requests and BeautifulSoup modules to parse HTML
- Explain some common problems with web scraping
- Explore other tools that can interact with web pages

Introduction to Web Scraping

- Web scraping involves programmatically grabbing data from a web page
- Three steps: Download, extract data, ~~PROFIT!~~



Okay...more like, do something with data

Why Scrape?

- There's data on a site that you want to store or analyze
- You can't get by other means (e.g. an API)
- You want to programmatically grab the data (instead of lots of manual copying/pasting)

Is it...ok?

- Some websites don't want people scraping them
- Best practice: consult the robots.txt file
- If making many requests, time them out
- If you're too aggressive, your IP can be blocked

Introduction to Beautiful Soup

Getting started with Beautiful Soup

- To extract data from HTML, we'll use Beautiful Soup
- Install it with pip
- Beautiful Soup lets us navigate through HTML with Python
- Beautiful Soup does NOT download HTML - for this, we need the requests module!

Parsing and Navigating HTML

- `BeautifulSoup(html_string, "html.parser")` - parse HTML
- Once parsed, There are several ways to navigate:
  - By Tag Name
  - Using `find` - returns one matching tag
  - Using `find_all` - returns a list of matching tags

Navigating with CSS Selectors

`select` - returns a list of elements matching a CSS selector

Selector Cheatsheet

- Select by id of foo: `#foo`
- Select by class of bar: `.bar`
- Select children: `div > p`
- Select descendents: `div p`

Selecting Elements by Attribute

```
# find an element with an id of foo
soup.find(id="foo")
soup.select("#foo")[0]

# find all elements with a class of bar
# careful! "class" is a reserved word in Python
soup.find_all(class_="bar")
soup.select(".bar")

# find all elements with a data
# attribute of "baz"
# using the general attrs kwarg
soup.find_all(attrs={"data-baz": True})
soup.select("[data-baz]")
```

Accessing Data in Elements

- `get_text` - access the inner text in an element
- `name` - tag name
- `attrs` - dictionary of attributes
- You can also access attribute values using brackets!

Navigating with Beautiful Soup

Via Tags

- `parent / parents`
- `contents`
- `next_sibling / next_siblings`
- `previous_sibling / previous_siblings`

Via Searching

- `find_parent / find_parents`
- `find_next_sibling / find_next_siblings`
- `find_previous_sibling / find_previous_siblings`

Web Scraping Example with Beautiful Soup

Requests + Beautiful Soup Example

- Let's scrape data into a CSV!
- Goal: Grab all links from Rithm School blog
- Data: store URL, anchor tag text, and date

Common Issues with Web Scraping

- Gnarly HTML
- Code tightly coupled to UI
- Sanitizing data after grabbing it
- Data that isn't part of HTML, but is loaded later!

Other Tools for Web Scraping

Other Tools

- Scrapy: https://scrapy.org/
- Selenium: http://www.seleniumhq.org/

Scrapy

- A more streamlined way to build web *crawlers*, which can programmatically navigate across multiple pages
- Can export to many different file formats from the command line

Selenium

- Allows you to open up a browser window from your code!
- Often used with testing
- Requires a *driver* for your browser of choice
- Doesn't navigate through the page until all contents have loaded

Recap

- Web scraping is the process of downloading, extracting, and storing data from a web page
- It's helpful when there's no other way to grab data you want
- Be sure you're allowed to scrape before you do so
- BeautifulSoup + requests allow you to scrape websites in Python
- Building scrapers can take time up front, but should save you time in the long term
- Other helpful tools include Scrapy and Selenium

YOUR TURN