Web Scraping notes

**Real Python online guide**

**Legal matters (i.e. how to avoid getting sued)**

[(link to article)](https://benbernardblog.com/web-scraping-and-crawling-are-perfectly-legal-right/)

Violation of copyright

* Web scraping may be considered a violation of copyright even if you just copy the raw data from the websites (arrangement/coordination of how data is kept/displayed can be considered a ‘creative arrangement’)
* Check the terms of service to ensure that you don’t violate any rules or potentially, any laws
* Alternatively, you can use their API (if they provide one) instead of scraping the site

General advice for web scraping

1. Use an API if one is provided, instead of scraping data.
2. Respect the Terms of Service (ToS).
3. Respect the rules of *robots.txt*.
4. Use a reasonable [crawl rate](https://support.google.com/webmasters/answer/48620?hl=en), i.e. don't bombard the site with requests. Respect the *crawl-delay* setting provided in *robots.txt*; if there's none, use a conservative crawl rate (e.g. 1 request per 10-15 seconds).
5. Identify your web scraper or crawler with a legitimate [user agent string](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent). Create a page that explains what you're doing and why, and link back to the page in your user agent string (e.g. 'MY-BOT (+<https://yoursite.com/mybot.html>)')
6. If ToS or *robots.txt* prevent you from crawling or scraping, ask a written permission to the owner of the site, **prior** to doing anything else.
7. Don't republish your crawled or scraped data or any derivative dataset without verifying the license of the data, or without obtaining a written permission from the copyright holder.
8. If you doubt on the legality of what you're doing, don't do it. Or seek the advice of a lawyer.
9. Don't base your whole business on data scraping. The website(s) that you scrape may eventually block you, just like what happened in [Craigslist Inc. v. 3Taps Inc.](https://en.wikipedia.org/wiki/Craigslist_Inc._v._3Taps_Inc.).
10. Finally, you should be suspicious of any advice that you find on the internet (including mine), so please consult a lawyer.

**What is Web scraping?**

* (Automated) gathering of information from the web

Reasons for Web Scraping

* To automate information gathering
* For example, trying to get updated job offerings on a daily basis. It would be troublesome and time consuming to go through the same process everyday if you were to do it manually.

Challenges of Web Scraping

* Variety 🡪 Different designs and structures of data for each website. Even though some websites may follow a general structure that repeats itself, every website is unique and would require personal attention when crafting the web scraper if you want to extract information from each website.
* Durability 🡪 Websites are ever changing. Web scrapers built for a website have to be revamped once a website’s structure is updated. (i.e. the same script cannot be used once changes are made to the website’s structure.)
  + But good news is that most websites’ changes are small and incremental, most of the time only minimal adjustments would have to be made to the web scraper script for it to work again.
  + Possible long-term solutions: set up a [continuous integration](https://realpython.com/python-continuous-integration/) to run scraping tests periodically to make sure your script is working

An Alternative to Web Scraping: APIs

* Application Programming Interfaces
* Allows for users to access website’s data in a predefined manner.
* Access data directly through JSON or XML files, don’t have to parse through HTML (used to present data visually)
* More stable than web scraping in general, as they made create to be consumed by programs rather than human eyes
* More reliable source of data, front end presentation might change, but API’s structure is usually more permanent. BUT APIs also susceptible to changes.
* Consider another [API tutorial](https://realpython.com/api-integration-in-python/) to learn about data gathering and collection to enhance your knowledge

**Scraping exercise (**[**Fake Python Job Site**](https://realpython.github.io/fake-jobs/)**)**

Step 1: Inspect Your Data Source

* Explore the website
  + Click through the website to see all the different functions and pages there are
* Decipher the Information in URLs
  + 2 Main parts in example URL
    - **The base URL** represents path to search functionality of the website e.g. <https://realpython.github.io/fake-jobs/>
    - **The specific site location** that ends with .html is path to job description’s unique resource
  + A lot of information present in the URL that the programmer can use to deduce what kind of data the particular page contains
  + Some websites use query parameters to encode values that users would submit when performing a search
    - Start

**Udemy Web Scraping course**

**Introduction**

What is the best web scraping library? BS4 vs Selenium vs Scrapy

* Beautiful soup
  + Can pull data out of html and xml file, easier webscraping library and most suitable for beginners
  + Only need 2 libraries, request and beautifulsoup
  + No Javascript support
  + Inefficient, complicated to transfer code between projects
* Selenium
  + Wasn’t designed for web scraping, was made for testing websites.
  + Works with javascript, one of the best libraries with Javascript
  + Easier to learn than Scrapy
  + Slow, web scraping with selenium is slower than http requests
    - How slow?
* Scrapy
  + Written completely in python
  + Harder to learn than both beautiful soup and selenium
  + Fast, don’t have to make requests one at a time
  + Most complete framework in python, easier store data in databases
* Which is the best?
  + Beautifulsoup is great for beginners
  + Selenium is good for small Javascript projects where speed is not priority
  + Scrapy is best for big projects where speed is a priority

Python basics for web scraping

* Lists
  + 0-based index
  + Initialisation example: states = ["California", "Texas", "Florida", "New York"]
  + Access the each item with square-brackets: states[0] give you first item, states[-1] give you last item
* For loops
  + E.g. for state in states:

print(states)

* + Based on indentation, very important, state can be renamed to anything, it represents each item in the “states” list
* If conditions
  + Example: If state == “Florida”:
  + No need for brackets, and the end of the condition is represented by a colon.
    - What if you need more than 1 condition? && and brackets?
* Supporting data with python
  + First method: with open(‘filename’, ‘w’) as file1
    - File1 refers to the whole ‘open’ operation
    - Open -> function that takes in a filename and a mode
      * 3 modes
        + Write (‘w’)🡪 delete existing file content or create a new file with the name, if it does not already exist. Then, allows you to write to the file

File will be created in current directory

* + - * + Read (‘r’)🡪
        + Append (‘a’)🡪
  + Second method: Using pandas
* Handling exception errors
  + ‘Try-except’ statement
    - Example:

Try:

Print(element/2)

Except:

Print(“Element is not a number”)

* + - Try will run when the code does not encounter an error
    - Except will run when code runs into an error, so this will prevent the whole code from not working.