Web Scraping

A step-by-step guide to scraping the web.

1 – Investigation

Navigate the website to get an idea of the size of data required. Pay particular attention to URLs [1]of pages.

Use Google to investigate the site and look for alternative sources of the data. See if the website has an official API or developer hub.

Look for any sitemaps and check the robots.txt file.

Check for API calls the site is making in network traffic.[2]

Try the website without JavaScript enabled (to see if you can use requests instead of requests-html python package which is quicker).[3] Also experiment with disabling cache/cookies.

2 – Ping site

Use Jupyter Notebook in an IDE (e.g. VSCode) for quick investigation.

Make request

```python
import requests
from bs4 import BeautifulSoup

url = 'https://www.example.com/page'
header = {'user-agent' : 'Mozilla ...'} # Request headers optional
response = requests.get(url) # See defaults used: response.request.headers
```

If using javascript use requests-html instead

```python
from requests_html import HTMLSession

url = 'https://www.example.com/page'
session = HTMLSession()
response = session.get(url)
```

If required use selenium with a headless driver. See docs.

[1] Check for query (URL) parameters `?page=1&token_id=20` and for paths (URL slugs) `/directory/farm2/`

[2] Go to developer mode in browser (F12) and go to Network tab. Then filter for XHR/fetch requests. Look at response body and header.

[3] In Chrome in developer mode Ctrl+Shift+P then disable JavaScript. Sidenote: you can simulate an iPhone etc in developer mode.

Investigate response

```
response.status_code
response.content

soup = BeautifulSoup(response.text, 'lxml') # use 'lxml-xml' to parse xml in-
stead of html
print(soup.prettify())
```

You can view the response in a browser as follows:

```
with open('website.html', mode='wb') as file:
    file.write(response.content)
```

3 – Parse result

In the case of simple API that returns JSON you can use response.json for the data

Otherwise use BeautifulSoup to locate the data you need. The main object of this module are Tag objects which correspond to html tags. These come with various methods and attributes to navigate the html. It is incredibly simple

```
soup.find_all('div', {'id' : '345'})
# First argument is html tag to filter by (e.g.  a for link tags etc)
# Second argument is html tag attributes to filter by (in dict format)
```

First experiment on one page /datapoint. Once you have this down to a T then you can start the next stage.

You may need to use regular expressions library in some cases.

```
import re
re.findall('regular expression pattern', 'string to search')
```

Returns a match object that has attributes that show the position, what is found etc.

4 – Do full scan

Don't forget crawl delay (can be randomised using random module)

```
from time import sleep
```

Use built-in I/O to save results periodically and print statements to show progress.

If saving in csv format:

```
import csv
with open('data.csv', mode='a') as file:
    csv_writer = csv.writer(file)
    csv_writer.writerow(['data', 'as', 'any', 'iterable'])
```