Web Scraping

Data Collection

Collecting data from webistes is a **drag** (hasn't it been a mess cleaning data for your project?!)

What if we could automate it?

- We often can with web scraping!

Accessing a website through Python

We will use the `requests` library

```
url = "https://poshmark.com/category/Women-Bags"

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64;\
    x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.\
    0.4472.124 Safari/537.36'
}

site = requests.get(url, headers=headers)
```

What's a header?

When your browser makes a request to a web server for a page you want to browse, it sends information about your browser to get the right information.

If we don't send this, we look very strange to servers, and are quickly picked up as "bots" (which we are!)

Poshmark

Used clothing and other fashion items are resold on poshmark, so it has an ever-changing catalogue of interesting things for us to look at!

Let's visit the Women's Bags page that we just requested via code.

Understanding HTML

In order to scrape a web page, we need to have a basic understanding of:

- HTML tags
- CSS formatting, (here also)
- JSON data structures

From there, we will spend LOTS of time on Google making sure we get it right for the page(s) that we care about.

Process the page

```python
from bs4 import BeautifulSoup

soup = BeautifulSoup(site.text)
```

Creates a structure of tags that we can navigate using the `BeautifulSoup` builtin methods

Explore

```
parsed.title
```

returns

```
<title>Women Bags on Poshmark</title>
```

Explore

```
parsed.title.text
```

returns

```
'Women Bags on Poshmark'
```

The `.text` attribute of each tag will work this way

Find the Bags

How can we collect information on listing in the results?

We need to identify the HTML that separates each listing from the others.

Find all

Let's find one of the "tiles" in the page (that's how the refer to them inside the page, so we can play along)

```
tiles = soup.find("div", class_="tiles_container")
```

Each tile can now be walked by examining the items in our list

Article names

If we want to find the names of each listing on a page, we can walk through our list of articles with a list comprehension:

```
listings[0].find("div", class_="title__condition__container")\
    .text.strip()
```

Note that because the listings are constantly changing, you will not get the same results as me!

Find the price

The prices are in a weird spot. It's a class called `m--t--1`, so we will extract the text from that tag to get the price:

```
listings[0].find("div", class_="m--t--1").text.strip()
```

Find the price

When we use the code above, we get back something like the following:

```
'$875'
```

That is ALMOST the price!

Regex for the win

Remember regular expression? Time for "regex" to shine:

```
import re

re.search(
    r'([$])(\d+.\d+)', # \u20AC is unicode for Euro
    listings[0].find("div", class_="m--t--1").text.strip(),
    re.UNICODE).groups()[1]
```

Wrap that all in a `float()` call and we get back something like

```
875.0
```

All the prices

We got a single item label and its price, but now we want to move on, and grab the name and price of each listed item on the page. Time for a loop!

```
data = [] # We will store information here
          # as a list of lists

for i in listings: # a is our list of article tags
    row = [] # One row per result

    row.append(i.find("div", class_="title__condition__container")\
    .text.strip()) # Add the title

    p_string = re.search(
            r'([$])(\d+(,\d+)?)',
            i.find("div", class_="m--t--1")\
            .text.strip()).groups()[1]
    p_string = float(p_string.replace(",",""))
    row.append(p_string) # Add the price
    data.append(row) # Put it into the data set
```

Frame it

```
import pandas as pd

data = pd.DataFrame(data, columns = ['Item', 'Price'])
```

Our Table (yours will differ)

	Item	Price
0	Vintage Coach Everett Crossbody Leather Bag	250.0
1	LOUIS VUITTON SAC POLOCHON 65 Monogram Canvas ...	1650.0
2	Hobo Spark Eternal Garden wristlet\n ...	58.0
...	...	...
47	Desigual 3D Floral Faux Snake Skin Leather Emb...	40.0

The next page

We need to use the characteristics of the "next page" link to consistently identify the link as we walk through each page of search results.

Look at the page. What do you see that could help you to reference the next page of results?

Maybe a button with text "Next"?? How could we access that? 🤔

Not this time...

Looking at the site, we can access the "Next" button with the following `.find` method:

```
nextPage = parsed.find('button', string="""
    Next
  """)
```

Unfortunately, it won't help us...

```
<button class="btn btn--pagination">
    Next
  </button>
```

Try again!

The website seems to use JavaScript rather than HTML to move to the next page of results. Bummer for us.

BUT! If we click the next button and copy the URL, we get the following link:

```
https://poshmark.com/category/Women-Bags?max_id=2
```

Next Page!

Now we can just do what we did before to the new page of results:

```
nextPage = "https://poshmark.com/category/Women-Bags?max_id=2"

myPage = requests.get(nextPage)

parsed = BeautifulSoup(myPage.text)
listings = [i for i in soup.find_all('div', class_="card--small")]

newData = []

for tile in listings:
  ... # see notebook for full code

newData = pd.DataFrame(newData, columns = ['listing', 'price'])
```

Combining pages of results

Once we find the next page links, we can run the code we have already written for each new page of results, and concatenate our Data Frames:

```
data = pd.concat([data, newData], axis=0).reset_index(drop=True)
```

Now we make a function

```
import requests
from bs4 import BeautifulSoup
import numpy as np
import pandas as pd
import re
import time

# A function to collect lego sets from search results on brickset.com
def poshmark(startURL, page=None):
    # keep track of what page we are on
    if page==None:
        page = 1
    # Add headers to imitate a real browser
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) \
          AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36',
        'Accept': 'text/html,application/xhtml+xml,\
          application/xml;q=0.9,image/webp,*/*;q=0.8',
        'Referer': 'https://www.google.com/'
    }
    # Retrieve starting URL
    myPage = requests.get(startURL)

    # Parse the website with Beautiful Soup
    parsed = BeautifulSoup(myPage.text)

    ... # function continued on next slide
```

```python
def poshmark(startURL, page=None):

    ... # continuation of function from last slide

    # Grab all sets from the page
    listings = [i for i in soup.find_all('div', class_="card--small")]

    # Create and empty data set
    newData = []

    # Iterate over all sets on the page
    for tile in listings:
        row = []
        try:
            row.append(tile.find('div', class_="title__condition__container").text.strip())
        except:
            row.append('')
        try:
            row.append(
                float(
                    re.search(r'(?:[$])(\d{1,3}(?:,\d{3})?)',
                           tile.find('div', class_="m--t--1").text).groups()[0].replace(",",""))
                )
            )
        except:
            row.append(np.nan)
        # Add the row of data to the dataset
        newData.append(row)

  ... # keeps going on another slide!
```

```python
def poshmark(startURL, page=None):

    ... # still finishing our function!

    newData = pd.DataFrame(newData, columns = ['listing', 'price'])

    # Until we have processed 5 pages, grab the next page of results
    if page<5:
        # Tell our program not to load new pages too fast
        #   by "sleeping" for two seconds before
        #   going to the next page
        time.sleep(2)
        # Merge current data with next page
        page += 1
        nextPage = f"https://poshmark.com/category/Women-Bags?max_id={page}"
        print(nextPage)
        return pd.concat([newData, poshmark(nextPage, page=page)], axis=0)
    # Otherwise return the current data
    else:
        return newData
```

Run the function

```
bags = poshmark("https://poshmark.com/category/Women-Bags")

bags
```

Get a DataFrame:

	listing	price
0	Black Wristlet and Wallet lot4 pieces total ch...	25.0
...	...	...
239	Fossil Snap Close Reddish Brown Crocodile Prin...	24.0

PHENOMENAL COSMIC POWERS!

Lab Time! Creating a scraping function

Now we have a script to scrape all the results from a specific category on Poshmark.

Also, our function is RECURSIVE!! 🤩🤩

Now it's your turn! Add some extra information into the `poshmark` function.