Final Presentations - Friday!

4-5 minutes, with 2 minutes after for questions

Web Scraping

Data Collection

Collecting data from webistes is a **drag**

What if we could automate it?

- We can do exactly that with web scraping!

Accessing a website through Python

We will use the `requests` library

```
import requests
myPage = requests.get("https://brickset.com/sets/year-2020")
```

Brickset

Great website for learning web scraping!

- There is a CSV export tool built into the website
- We can compare our results against a CSV of the *correct* results
- Also, it's about Legos!

Let's visit the page

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

```
from bs4 import BeautifulSoup

parsed = BeautifulSoup(myPage.text)
```

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

Explore

```
parsed.title
```

returns

```
<title>2020  | Brickset: LEGO set guide and database</title>
```

Explore

```
parsed.title.text
```

returns

```
'2020  | Brickset: LEGO set guide and database'
```

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

Find all

Let's find all of the `<article>` tags in the page

```
a = [i for i in parsed.find_all('article')]
```

Each `<article>` tag can now be walked by examining the items contained in our list

Article names

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

```
[i.h1.text for i in a]
```

Find the price

We can see on the page that the price is preceded by the text "RRP". Let's inspect that text, and then try to find the tags that will identify that text block.

Find the price

"RRP" sits inside a `<dt>` tag. We can find those tags using the following:

```
a[0].find('dt', text="RRP")
```

We get back

```
<dt>RRP</dt>
```

Find the price

From here, we can "walk" from one tag to the next, because we can see (upon *inspection*) that the price is always in the next tag after "RRP"

```
a[0].find('dt', text="RRP").find_next_sibling().text
```

Now we get

```
'$179.99, 159.99€ | More'
```

That is ALMOST the price!

Regex for the win

Remember regular expression? Here is where it will become VERY valuable:

```
import re

re.search(
    r'(\d+.\d+)(\u20AC)',
    a[0].find('dt', text="RRP").find_next_sibling().text,
    re.UNICODE).groups()[0]
```

And we get back

```
'159.99'
```

All the prices

We got a single 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

for i in a:
    row = [] # One row per result
    row.append(i.h1.text) # Add the title
    try: # Unless there is an error
        row.append(
            re.search(
                r'(\d+.\d+)(\u20AC)',
                i.find('dt', text="RRP").find_next_sibling().text,
                re.UNICODE).groups()[0]) # Add the price
    except: # If there is an error
        row.append('') # Leave the entry blank
    data.append(row) # Put it into the data set
```

Frame it

```
import pandas as pd

data = pd.DataFrame(data, columns = ['Set', 'Price_Euro'])
```

	Set	Price_Euro
0	Basic Building Set with Storage Case	
1	Bookshop	155.96
2	Fiat 500	77.97
...	...	...

The next page

```
nextPage = parsed.find('li', class_="next").a['href']
```

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.

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)
```

Creating a scraping function

Now that we have all of the needed elements, we can create a script to scrape all the results from a specific search on Brickset:

```
#Import statements
import requests
from bs4 import BeautifulSoup
import numpy as np
import pandas as pd
import re
```

```
# A function to collect lego sets from search results on brickset.com
def collectLegoSets(startURL):
    # Retrieve starting URL
    myPage = requests.get(startURL)

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

    # Grab all sets from the page
    a = [i for i in parsed.find_all('article')]

    # Create and empty data set
    newData = []
```

The code on following slides is also part of the function

```
    # Iterate over all sets on the page
    for i in a:
        row = []
        # Add the set name to the row of data
        row.append(i.h1.text)
        try:
            # Extract price and translate to a floating point number from string,
            #   append to row IF PRICE EXISTS
            row.append(
                float(
                    re.search(
                        r'(\d+.\d+)(\u20AC)',
                        i.find('dt', text="RRP").find_next_sibling().text,
                        re.UNICODE).groups()[0]))
        except:
            # Missing value for sets with no price, append to row
            #   IF NO PRICE EXISTS
            row.append(np.nan)

        # Add the row of data to the dataset
        newData.append(row)
```

```
    newData = pd.DataFrame(newData, columns = ['Set', 'Price_Euro'])

    # Check if there are more results on the "next" page
    try:
        nextPage = parsed.find('li', class_="next").a['href']
    except:
        nextPage = None

    # If there is another page of results, grab it and combine
    if nextPage:
        return pd.concat([newData, collectLegoSets(nextPage)], axis=0)
    # Otherwise return the current data
    else:
        return newData
```

This is the end of the function!

Running our function

```
lego2020 = collectLegoSets("https://brickset.com/sets/year-2020")
```

Lab Time!