Lab 11: Web Scraping

CSE/IT 107

NMT Computer Science

"The limits of my language mean the limits of my world."

— Ludwig Wittgenstein

"Any sufficiently advanced technology is indistinguishable from magic."

— Arthur Clarke

"Imagination is more important than knowledge."

— Albert Einstein

---

1 Introduction

In this lab, you will be extracting information from web pages and learning how to plot some of it.

You may be familiar with the Wikipedia game of clicking on the first link and figuring out how long it takes you to arrive at the Philosophy page. You will be writing a Python script that starts at a Wikipedia page and does this to figure out how many clicks it takes.

2 HTML

As you may be familiar, HTML is the main formatting language of the Internet. HTML stands for HyperText Markup Language. As a language, it defines where content appears on a website, and how it looks. For an example, open up `Google Chrome` and navigate to `http://www.nmt.edu`. Right click on the webpage, and select "View Source". What follows is the source code for `nmt.edu`!

HTML is actually fairly easy to read. All content exists between various kinds of "tags". Each tag has a special kind of meaning. Here is a simple example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello!</title>
  </head>
  <body>
    <p> This is a paragraph in the body of the page </p>
    <p> This text is <b> BOLD </b></p>
  </body>
</html>
```

The tags `html`, `head`, and `body` are required for a web page and just give some structure. `head` contains information about the page, such as the title. `body` contains the actual content of the web page. The special `<!DOCTYPE html>` also has to be there.

To learn about all the various tags, see

`http://www.w3schools.com/`.

3 Requesting a web page

The following is an example code snippet to get the source code of a particular web page:

```
from urllib import request

def get_page_source(webpage):
  req = request.Request(webpage, headers={'User-Agent' : 'Python Browser'})
  page = request.urlopen(req)
  return page.read()

print(get_page_source('https://cs.nmt.edu/~ckoch/cse107/beautifulsoup.html'))
```

This will give you the HTML source of the web address you give it.

Below is an example of a slightly more complex program using urllib. This program gets a random Wikipedia article and prints out its URL. This is identical to what would happen if you click on the Random Article button on the Wikipedia homepage.

```
from urllib import request

base = 'https://en.wikipedia.org'

def get_random():
  req = request.Request(base + '/wiki/Special:Random',
    headers={'User-Agent' : 'Python Browser'})
  page = request.urlopen(req)
  url = page.geturl()
  url = url[len(base):]
  return url

def main():
  rand_url = get_random()
  print('Found random page {}.'.format(rand_url))

if __name__ == '__main__':
  main()
```

All of the logic for this program is inside the `get_random` function. First, it creates a `Request` object. This is used to tell urllib what page we want to load. First, we construct our URL using the `base` variable (since all of our URLs are going to start the same way), then we pass a special headers argument. Don't worry too much about what headers are – in this case, we are only specifying a `User-Agent` string in order to identify ourselves to the remote server. We need to do this because the default urllib `User-Agent` string is blocked by Wikipedia.

Once we have our `Request` object created, we simply pass it to `request.urlopen` in order to get an object representing the page we've loaded. Creating the `Request` object and calling `request.urlopen` will be done no matter what page we are loading.

Once we have the page object, we simply check its URL (since the Wikipedia random article button redirects you to a new page, this will not be the same URL that we requested!) and strip off the beginning section, leaving us with only /wiki/Article.

4 BeautifulSoup

BeautifulSoup is a library that will let you “scrape” web pages for certain information. Instead of having to deal with the HTML source of a web page directly, you can use BeautifulSoup to find information easily. If, for example, I want to find all the a elements (link elements) on a web page, I could do the following:

```python
import bs4

# Using previous function
data = get_page_source('https://cs.nmt.edu/~ckoch/cse107/beautifulsoup.html')
soup = bs4.BeautifulSoup(data)

print(soup.prettify())

elements = soup.findAll('a')
print(elements)
```

Listing 1: BeautifulSoup

The output of `.prettify()` looks like this:

```html
<!DOCTYPE html>
<html>
 <head>
  <title>
   CSE107 BeautifulSoup Test
  </title>
 </head>
 <body>
  <p>
   Haha
  </p>
  <p>
   <a href="https://cs.nmt.edu/~ckoch/">
    Chris's site
   </a>
   <a href="https://arctem.com">
    Russell's site
   </a>
  </p>
 </body>
</html>
```

On this page, we have two a elements, and thus `.findAll()` returns the following list:

```
[<a href="https://cs.nmt.edu/~ckoch/">Chris's site</a>,
<a href="https://arctem.com">Russell's site</a>]
```

Now, to find just the first element, we use the `.find()` function. Also note that the above list is not a list of strings; the type of the list elements is a beautiful soup element tag. These tags have a few properties, as you can see in the following example:

```python
import bs4
# calling beautiful soup with my own data
```

```
soup = bs4.BeautifulSoup('<html><body><p>Test</p><p>T</p></body></html>')

paragraph = soup.find('p') # finds <p>Test</p>
print(paragraph)
print(paragraph.string) # Test
print(paragraph.contents) # ['Test']
print(paragraph.name) # prints p

body = soup.find('body') # finds <body><p>Test</p></body>
print(body.contents) # [<p>Test</p>, <p>T</p>]
print(body.string) # None, because there is more than one thing
```

Listing 2: BeautifulSoup tag usage

4.1 Summary

Tables 1 and 2 show the BeautifulSoup methods and the tag attributes we will probably need in this lab. If you want to know more about BeautifulSoup, take a look at its documentation:

`http://www.crummy.com/software/BeautifulSoup/bs4/doc/`

Method	**What it does**
`s.find(name)`	Finds the first tag named `name` in document `s`
`s.findAll(name)`	Finds all tags named `name` in document `s`
`s.prettify()`	Returns a string of the HTML formatted prettily

Table 1: BeautifulSoup methods, where s is a BeautifulSoup object. See Listings 1 and 2 for example usage.

Attribute	**What it gives**
`.contents`	List of all the things a tag contains
`.name`	Name of the tag (e.g. `p` or `title`)
`.string`	If `.contents` only contains one thing, return a string of that. Otherwise, `None` since it will not know which one to choose

Table 2: Tag attributes. See Listing 2 for example usage.

5 Matplotlib

Matplotlib is a neat library for plotting in Python. It works similarly to MATLAB plotting so that people with experience in that can easily switch over; however, we will give you our own introduction to it.

Let's plot the following *x* and *y* coordinates:

x	0	2	4	6	8	10	12	14	16	18
y	0	4	8	12	16	20	24	28	32	36

Table 3: Values for Figure 1

```python
import matplotlib.pylab as pl

x = range(0, 20, 2)
y = range(0, 40, 4)
pl.plot(x, y, '.')
pl.xlabel('X values')
pl.ylabel('Y values')
pl.title('Random plot of X vs Y')
pl.show()
```

Listing 3: Code to produce Figure 1 with values from Table 3

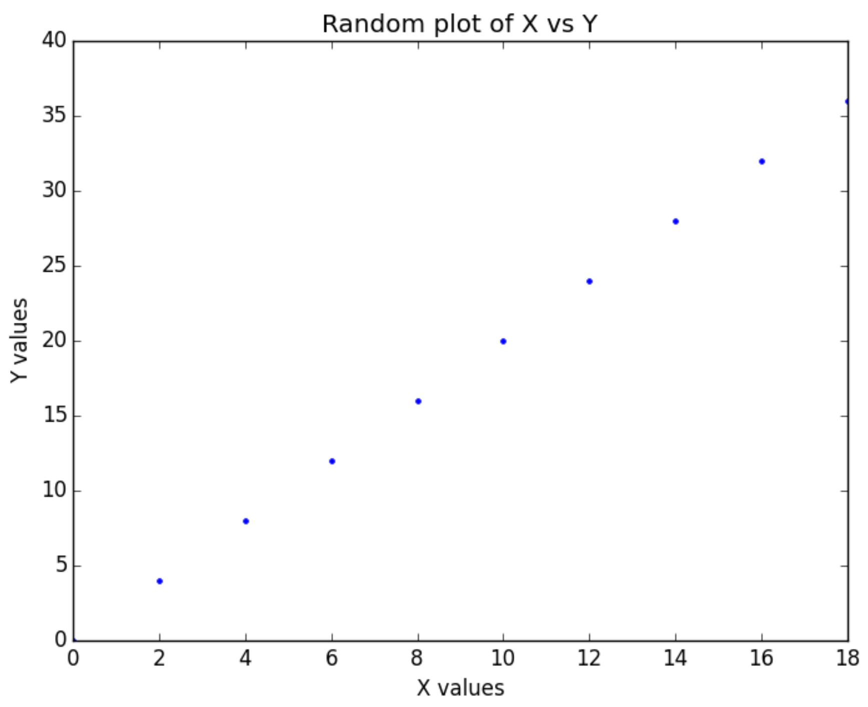

Figure 1: Graph produced by Listing 3

The `pl.plot()` function takes as arguments a sequence for x, a sequence for y, and optionally a formatting specifier. The important ones are "-" for a solid line and "." for point markers. You can add more options, such as colors or labels. You can find more docs on these specifiers on the Matploblib website:

`http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot`

You have to call `pl.show()` for the graph to actually show. There are ways to print the graphs to image files, but we are not covering those for now. You can look them up in the documentation if you want.

Bar plots work in a similar way. The `pl.bar()` takes a sequence of numbers to label the left side of the bars with and a sequence of heights of the bars.

```python
import matplotlib.pylab as pl

x = range(1, 11, 2)
y = range(1, 21, 4)
pl.bar(x, y)
pl.show()
```

Listing 4: Code to produce Figure 2

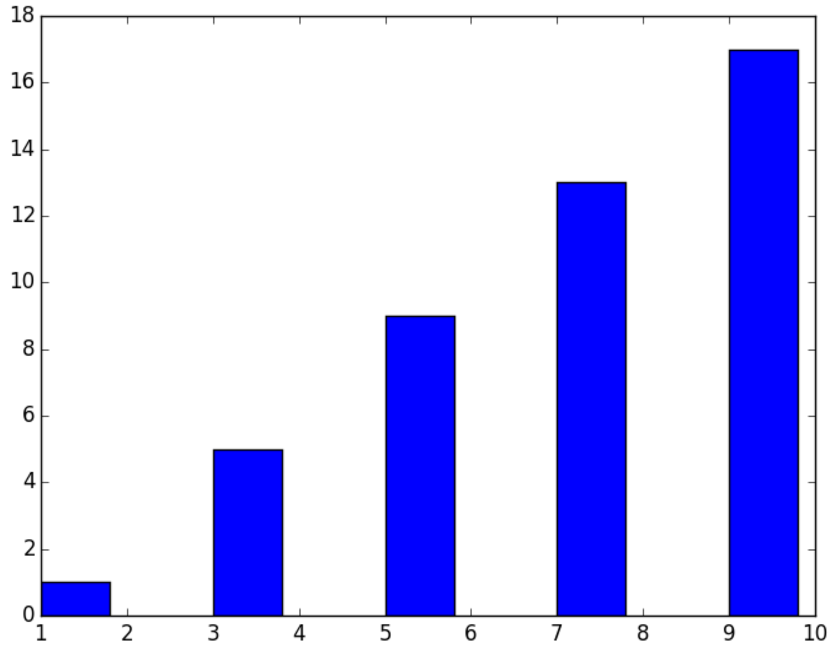

Figure 2: Graph produced by Listing 4

These are the two main functions you will need for plotting in this lab. If you want to customize more, please see the matplotlib pylab documentation at

`http://matplotlib.org/api/pyplot_summary.html.`

6 Exercises

Boilerplate

Remember that this lab *must* use the boilerplate syntax introduced in Lab 5.

simple_plot.py Using Matplotlib, make a simple plot of the function $y = x^2$ to make sure that you are able to use matplotlib.

beautiful_title.py Like the last exercise, just to make sure the libraries are working properly, use `get_page_source` and `BeautifulSoup` to get the title of a user-entered Wikipedia page.

wiki_stats.py This comprises the majority of the project. Your task takes advantage of the "philosophy property" of most Wikipedia pages. The user will provide a list of Wikipedia article names. Your program will take this list, and return a list of the "first-link-distance" to Philosophy. That is, the number of articles you must visit in order to reach Philosophy, if for each article you simply follow the first link in that article. If the given article does not converge to Philosophy, display the string "inft". It the given article does not exist, display the string "N/A".

Take a look at the provided wikiclicker.py for a function that gets you the first link on a Wikipedia page.

For example, see this sample input (the numbers here are just made up, don't worry)...

```
Article List: NASA, Space, Grapefruit, BadArticle
Philosophy Distance: [3, 6, inft, N/A]
```

So this means that three clicks from "NASA", you can find "Philosophy", but "Grapefruit" never reaches "Philosophy". "Bad Article" is not found.

If the user list was empty, however, generate a random list of 5 articles. For example (again, made up data)...

```
Article List:
Choosing random articles...
Articles: Dog, Cat, Roger Ebert, Star Wars, Philosophy
Philosophy Distance: [2,1,4,5,4]
```

Finally, your program must **plot a bar graph** of the distance of each article, with labeled axes, identifying each bar. If the distance was "N/A" or "inft", then do not plot that article.

`http://en.wikipedia.org/wiki/Wikipedia:Getting_to_Philosophy`

7 Submitting

Files to submit:

- simple_plot.py (Section 6)
- beautiful_title.py (Section 6)
- wiki_stats.py (Section 6)

You may submit your code as either a tarball (instructions below) or as a .zip file. Either one should contain all files used in the exercises for this lab. The submitted file should be named either `cse107_firstname_lastname_lab11.zip` or `cse107_firstname_lastname_lab11.tar.gz` depending on which method you used.

For Windows, use a tool you like to create a `.zip` file. The TCC computers should have `7z` installed. For Linux, look at lab 1 for instructions on how to create a tarball or use the "Archive Manager" graphical tool.

Upload your tarball or .zip file to Canvas.