CS 432: Homework Number One

Due on January 31, 2018 at 11:00 AM

Alexander Nwala

Tim Bruce

The LaTeX Template

It should first be noted that I did not create this template. I first used this template about a year ago for my Linear Algebra class at the University of Maine. I am using this template because it is similar to the template that you provided. I used to have a record of who made the template, but now I have lost that information. Let it be known that I am not REMOTELY this good at using LaTeX.

Problem 1

Demonstrate that you know how to use ”curl” well enough to correctly POST data to a form. Show that the HTML response that is returned is ”correct”. That is, the server should take the arguments you POSTed and build a response accordingly. Save the HTML response to a file and then view that file in a browser and take a screen shot.

Solution
I began by finding the way to post to the server and get a response. As can be seen from my first attempt which is shown below, it wasn't as simple as providing data.

Listing 1: First Curl Attempt.

```
curl -d "fname=hello&lname=world"
https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php
```

I am including this because I feel that it is important to showing how I completed the problem. I didnt́ remember how to use the man command at this point. I simply created this using a GitHub post by user subfuzion [1]. After this, I remembered how to use (and more importantly, search) manual pages in terminal. Using this, I realized that data does not actually post. That requires a command that asks for a request.

Listing 2: Correct Curl Syntax.

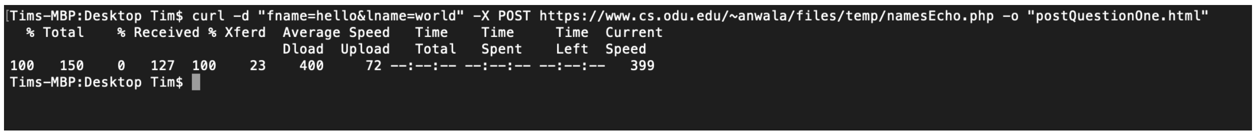
```
curl -d "fname=hello&lname=world" -X POST
https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php
```

Finally, I went over how we were supposed to provide output. While it looks like we are supposed to copy and paste the outputted HTML code into an HTML file and open that with our browser of choice, I checked the man page, and writing the output to a file is not difficult.

Listing 3: Correct Curl Syntax, with output to a file.

```
curl -d "fname=hello&lname=world" -X POST
https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php -o "postQuestionOne.html"
```

Image 1: Curl command being executed.

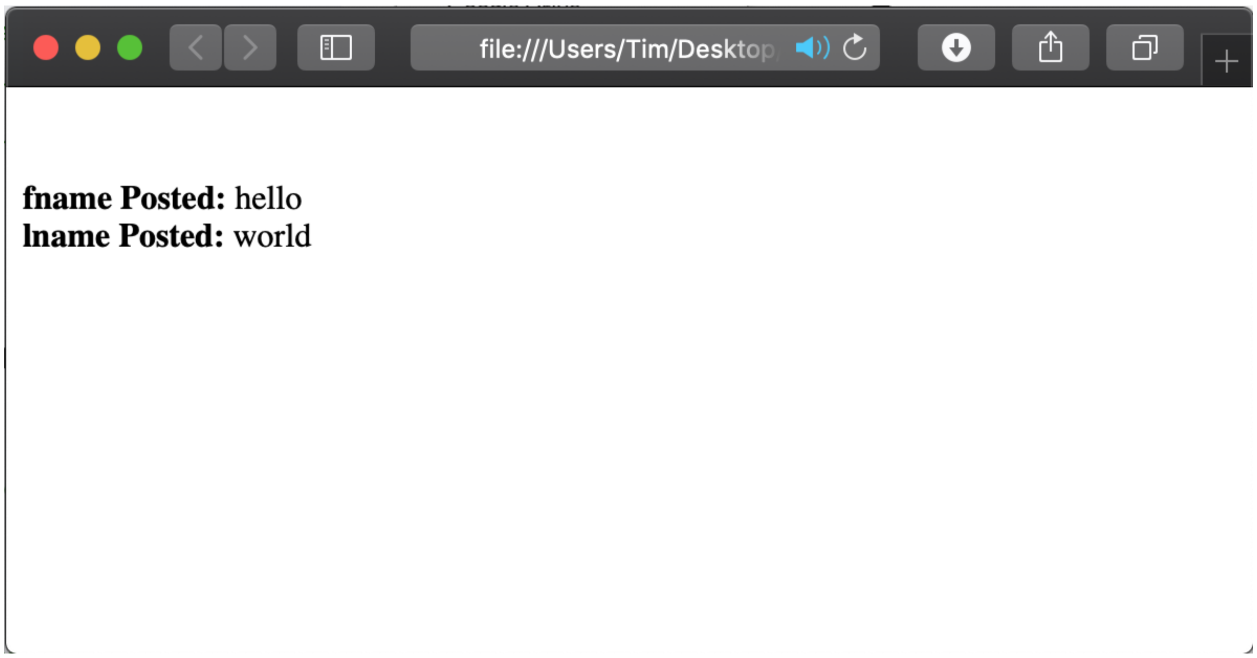


```
[Tims-MBP:Desktop Tim$ curl -d "fname=hello&lname=world" -X POST https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php -o "postQuestionOne.html"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   150    0   127  100    23    400     72 --:--:-- --:--:-- --:--:--   399
Tims-MBP:Desktop Tim$ 
```

This produced the following output when the file ”postQuestionOne.html” was opened.

Image 2: The HTML that was returned.

Problem 1 continued on next page. . .



file:///Users/Tim/Desktop

fname Posted: hello
lname Posted: world

Problem 2

Write a Python program that:

1. takes as a command line argument a web page
2. extracts all the links from the page
3. lists all the links that result in PDF files, and prints out the bytes for each of the links. (note: be sure to follow all the redirects until the link terminates with a "200 OK".)
4. show that the program works on 3 different URIs, one of which needs to be: http://www.cs.odu.edu/ mln/teaching/cs532-s17/test/pdfs.html

Solution

Part A

Take a web page as a command line argument.
This is simple to do with the sys package in Python. However, I wanted to allow for some bad inputs in order to make this function reusable on future assignments. So I implemented the following program.

Listing 4: URL User Error Correction

```
#Makes sure the address has an http or https
def ParseAddress(address, initialAddress = ""):
    if (address[0:8] != "https://" and address[0:7] != "http://"):
        if initialAddress != "":
            address = initialAddress + address
        else:
            address = "https://" + address
    return address
```

This function checks if there is an http or https tag at the front of an inputted URL. If there is not such a tag, it will add one, either by just adding one and trying, or by using an optional parameter which would

Problem 2 continued on next page. . .

give an already known link to put onto the front.

Part B

Extract all of the links from the web page at the previous part's URL.
This part actually has two sub-parts. 1. Open the web page, and 2. Extract all of the links. For the first part, urllib was used in a function called GetHTML with a parameter of the URL. This function simply returns the HTML code, and was written by using the URLLIB user documentation [2].
Next, the BeautifulSoup library for Python was used to extract all of the "<a>" blocks. Once again, this is basic functionality and was done using the BeautifulSoup documentation [3]. This left the program with the "<a>" blocks, but in order to check the links, there needs to be just the URL. This was done with some string manipulation using the standards for links in web pages.

Listing 5: URL Extraction From HTML.

```
links = []
    for i in range(len(a_components)):
        start_bound = str(a_components[i]).find('href=\"')+6
        end_bound = str(a_components[i])[start_bound:].find('\"') + start_bound
        links.append(str(a_components[i])[start_bound : end_bound])
    return links
```

In listing 5, a_components is a list of every <a>block as a string. This snippet looks for the href tag in the <a>block and determines where the URL is in the string from there.

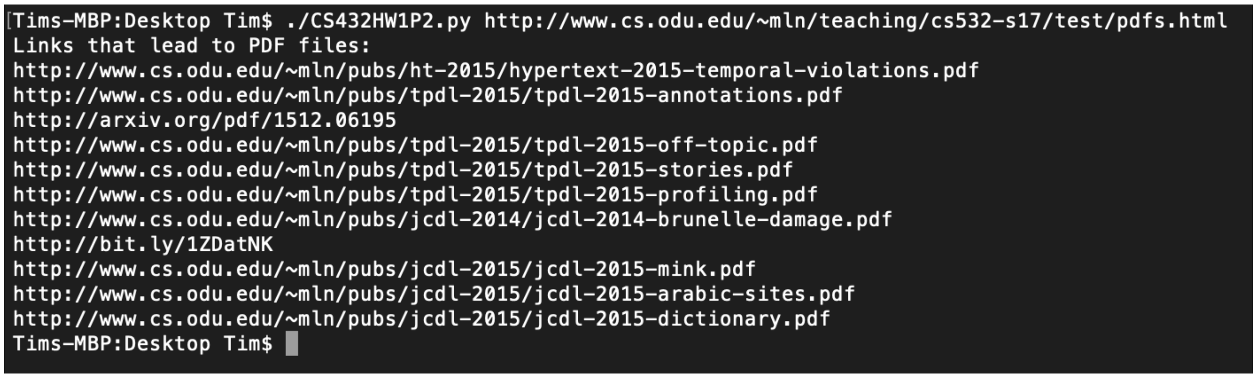
Part C

List all the links that result in PDF files, and print out the bytes for each of the links. (note: be sure to follow all the redirects until the link terminates with a "200 OK".)
In the Slack for this class, Joshua Gahan confirmed that urllib already follows redirects. How to determine if a link actually led to a PDF had me stumped for a while, until I discovered a question by user gkennos on stack overflow about this very question. The answer is that in the header of a web page there is a "content-type" tag. So I wrote a function very similar to the GetHTML function described earlier called GetURLHeader, which returns the header as a Python dictionary, once again using the urllib library.

Part D

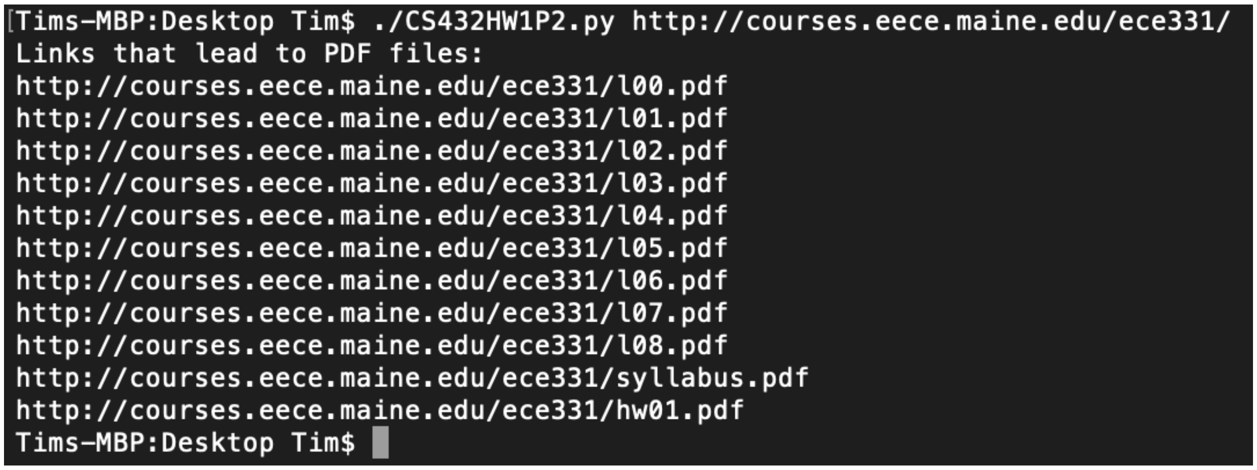
After all that, the program simply lists all of the links that it found! There is some additional code to show the status of checking the links because it takes a few seconds. The link status indicator does not show in the final output. Here is the output from three inputted URIs.

Image 3: Python output for given web page.



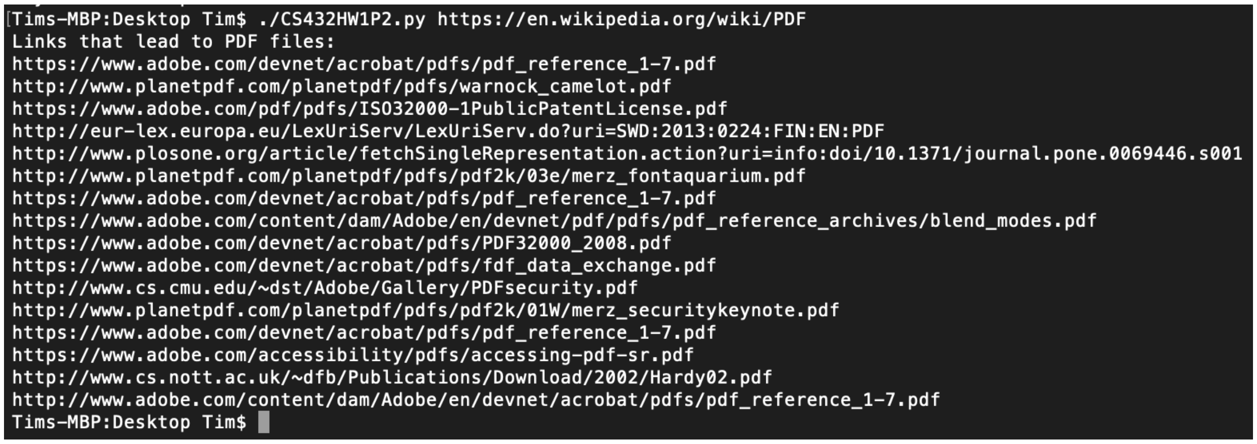
```
Tims-MBP:Desktop Tim$ ./CS432HW1P2.py http://www.cs.odu.edu/~mln/teaching/cs532-s17/test/pdfs.html
Links that lead to PDF files:
http://www.cs.odu.edu/~mln/pubs/ht-2015/hypertext-2015-temporal-violations.pdf
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-annotations.pdf
http://arxiv.org/pdf/1512.06195
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-off-topic.pdf
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-stories.pdf
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-profiling.pdf
http://www.cs.odu.edu/~mln/pubs/jcdl-2014/jcdl-2014-brunelle-damage.pdf
http://bit.ly/1ZDatNK
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-mink.pdf
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-arabic-sites.pdf
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-dictionary.pdf
Tims-MBP:Desktop Tim$
```

Image 4: Python output for a web page from a class I took a year ago.



```
[Tims-MBP:Desktop Tim$ ./CS432HW1P2.py http://courses.eece.maine.edu/ece331/
Links that lead to PDF files:
http://courses.eece.maine.edu/ece331/l00.pdf
http://courses.eece.maine.edu/ece331/l01.pdf
http://courses.eece.maine.edu/ece331/l02.pdf
http://courses.eece.maine.edu/ece331/l03.pdf
http://courses.eece.maine.edu/ece331/l04.pdf
http://courses.eece.maine.edu/ece331/l05.pdf
http://courses.eece.maine.edu/ece331/l06.pdf
http://courses.eece.maine.edu/ece331/l07.pdf
http://courses.eece.maine.edu/ece331/l08.pdf
http://courses.eece.maine.edu/ece331/syllabus.pdf
http://courses.eece.maine.edu/ece331/hw01.pdf
Tims-MBP:Desktop Tim$
```

Image 5: Python output for the Wikipedia page on the PDF format.



```
[Tims-MBP:Desktop Tim$ ./CS432HW1P2.py https://en.wikipedia.org/wiki/PDF
Links that lead to PDF files:
https://www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf
http://www.planetpdf.com/planetpdf/pdfs/warnock_camelot.pdf
https://www.adobe.com/pdf/pdfs/ISO32000-1PublicPatentLicense.pdf
http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=SWD:2013:0224:FIN:EN:PDF
http://www.plosone.org/article/fetchSingleRepresentation.action?uri=info:doi/10.1371/journal.pone.0069446.s001
http://www.planetpdf.com/planetpdf/pdfs/pdf2k/03e/merz_fontaquarium.pdf
https://www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf
https://www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/pdf_reference_archives/blend_modes.pdf
https://www.adobe.com/devnet/acrobat/pdfs/PDF32000_2008.pdf
https://www.adobe.com/devnet/acrobat/pdfs/fdf_data_exchange.pdf
http://www.cs.cmu.edu/~dst/Adobe/Gallery/PDFsecurity.pdf
http://www.planetpdf.com/planetpdf/pdfs/pdf2k/01W/merz_securitykeynote.pdf
https://www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf
https://www.adobe.com/accessibility/pdfs/accessing-pdf-sr.pdf
http://www.cs.nott.ac.uk/~dfb/Publications/Download/2002/Hardy02.pdf
http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf
Tims-MBP:Desktop Tim$
```

Problem 3

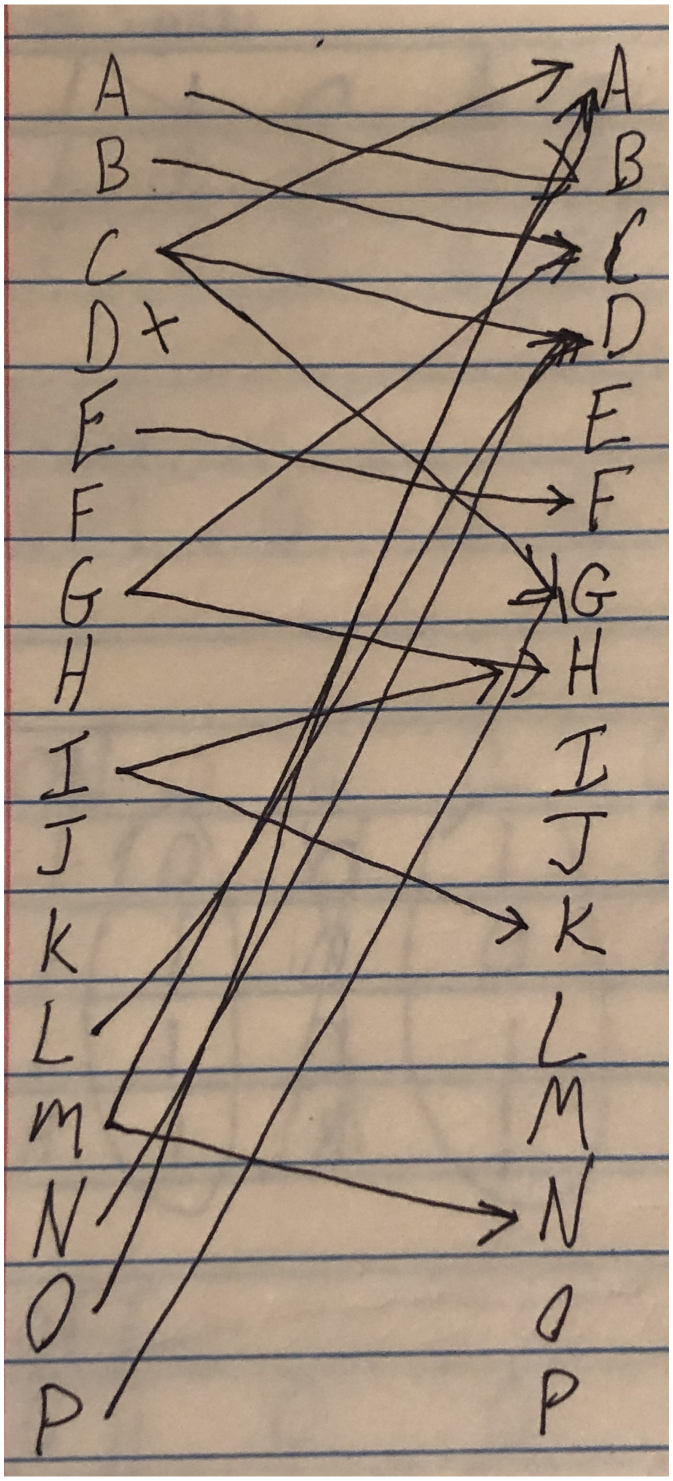
Consider the "bow-tie" graph in the Broder et al. paper:
http://snap.stanford.edu/class/cs224w-readings/broder00bowtie.pdf
Many have found this link useful: https://www.harding.edu/fmccown/classes/archive/comp475-s13/web-structure-homework.pdf Now consider the following graph:
A –>B
B –>C
C –>D
C –>A
C –>G
E –>F
G –>C
G –>H
I –>H
I –>K
L –>D
M –>A

M –>N
N –>D
O –>A
P –>G

For the above graph, give the values for:
IN, SCC, OUT, Tendrils, Tubes, and Disconnected

Solution
To solve this puzzle I started by writing each letter in order twice, with all of the connections being drawn from left to right. Note that I assume that J exists despite lack of any information about its existence.

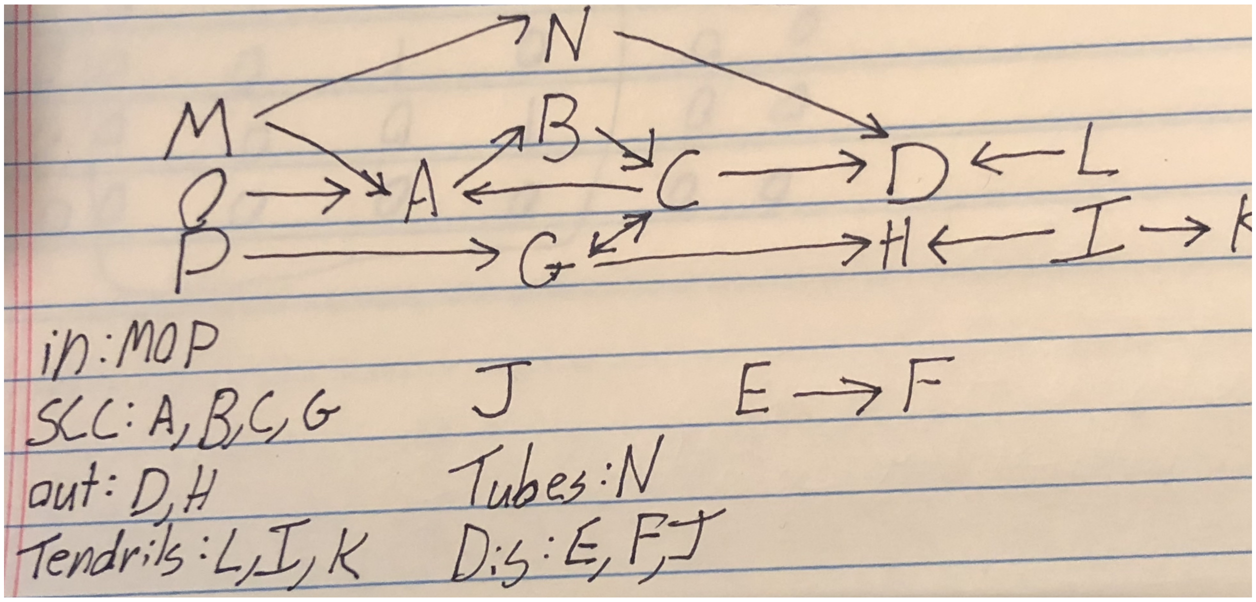
Image 6: Method used to organize the links.



Using this I was able to get a good sense of what was what. After one test-draw, I was able to determine a good clean final version of the graph. The in nodes all lead into the SCC. SCC all lead to each other. End are all links out of

Image 7: The final graph.

Problem 3 continued on next page. . .



The nodes are organized as follows:
IN: M, O, P
SCC: A, B, C, G
OUT: D, H
TENDRILS: L, I, K
TUBES: N
DISCONNECTED: E, F, J

References

[1] subfuzion, "curl post examples," Feb 2017. [Online]. Available: https://gist.github.com/subfuzion/08c5d85437d5d4f00e58

[2] "urllib.request - extensible library for opening urls." [Online]. Available: https://docs.python.org/3/library/urllib.request.htmlmodule-urllib.request

[3] "Beautiful soup documentation." [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/