

ASSIGNMENT 1

CS 432 Web Science

Mackenzie Kerchner

Contents

Problem 1	2
Problem 2	3
Problem 3	5

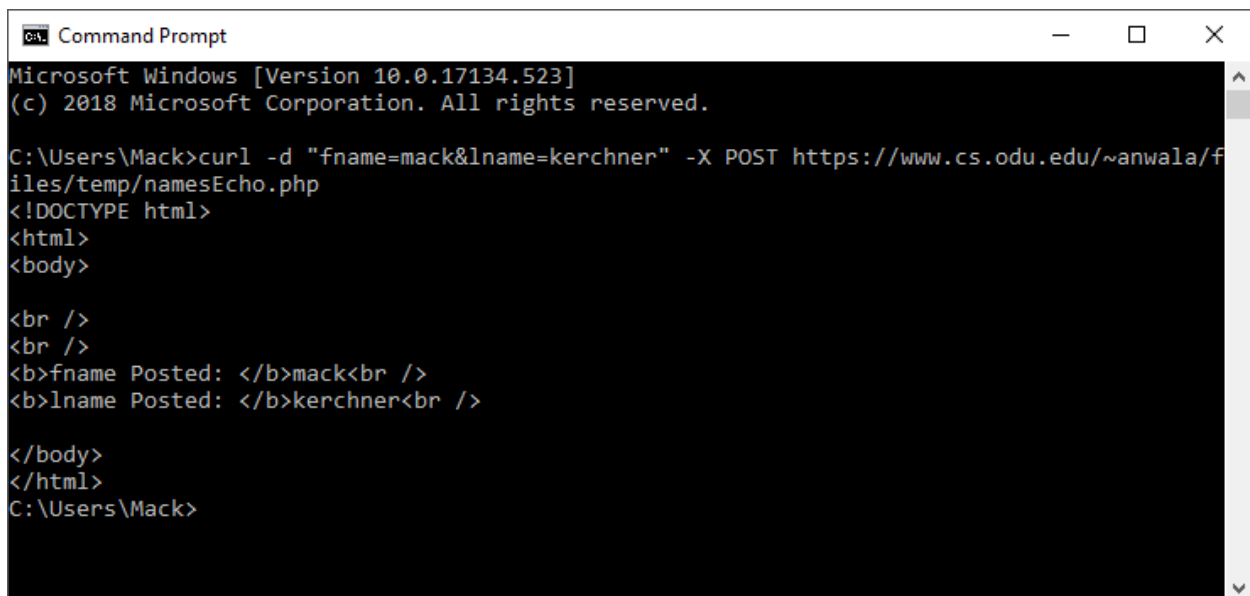
Problem 1

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.

Feel free to use my simple server for sending POST requests: <http://www.cs.odu.edu/~anwala/files/temp/namesEcho.php>
The server needs you to POST data for "fname" and "lname" fields.

Solution

I googled curl command and read up on how it worked, <https://curl.haxx.se/docs/manpage.html> and <https://gist.github.com/subfuzion/08c5d85437d5d4f00e58> were very helpful in figuring it out quickly.

A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt". The text inside shows the command prompt at "C:\Users\Mack>". The user enters the command: `curl -d "fname=mack&lname=kerchner" -X POST https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php`. The output is an HTML document. It starts with `<!DOCTYPE html>`, followed by `<html>` and `<body>`. Inside the body, there are two lines: `fname Posted: mack
` and `lname Posted: kerchner
`. The body and html tags are closed. The prompt returns to `C:\Users\Mack>`.

```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Mack>curl -d "fname=mack&lname=kerchner" -X POST https://www.cs.odu.edu/~anwala/f
iles/temp/namesEcho.php
<!DOCTYPE html>
<html>
<body>

<br />
<br />
<b>fname Posted: </b>mack<br />
<b>lname Posted: </b>kerchner<br />

</body>
</html>
C:\Users\Mack>
```

```
curl -d "fname=mack&lname=kerchner" -X POST
https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php
<!DOCTYPE html>
<html>
<body>

<br />
<br />
<b>fname Posted: </b>mack<br />
<b>lname Posted: </b>kerchner<br />

</body>
</html>
```

Problem 2

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

initial pseudocode:

input webpage through command line

webpage opens y/n

 if n, return the error code

 loop until a page opens correctly

library to parse code until found all links

go through list of links

 {

 follow redirects until end

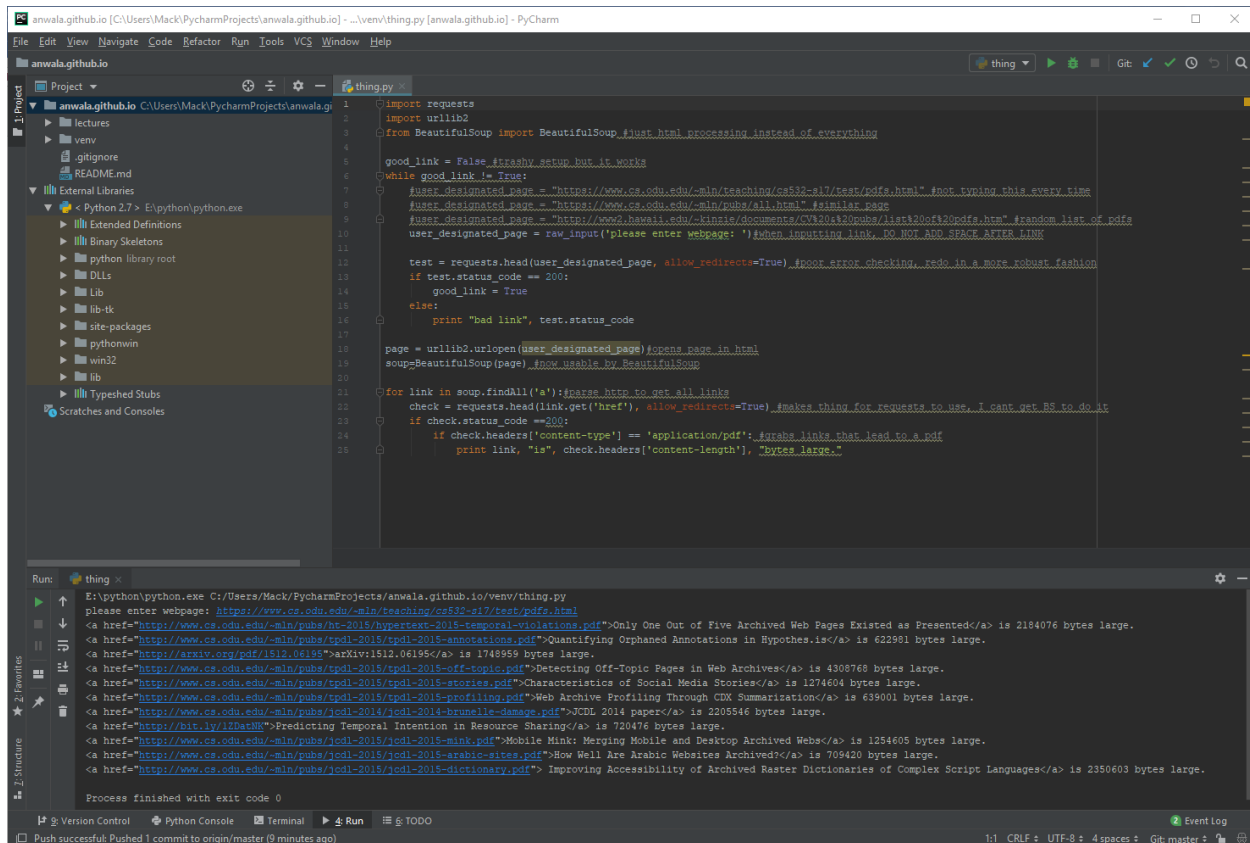
 pdf y/n

 if y, save link and number of bytes to list/tuple

 }

Print list of links and their associated bytes

This was my first time using python so the “python 101” [1] slideshow provided by Professor Nwala was invaluable in learning proper syntax. I used a combination of urllib3 [2] and BeautifulSoup [3] to handle data, the online user guides for the respective modules made it quite easy. I was getting an error with one of the links, which I found out in the lecture was being caused by redirects and solved this by setting allow_redirects=True, which is false by default, but only for HEAD. The program is not very robust and can break on some pages, I will have to put in more robust error checking so it works on a wider range of pages.

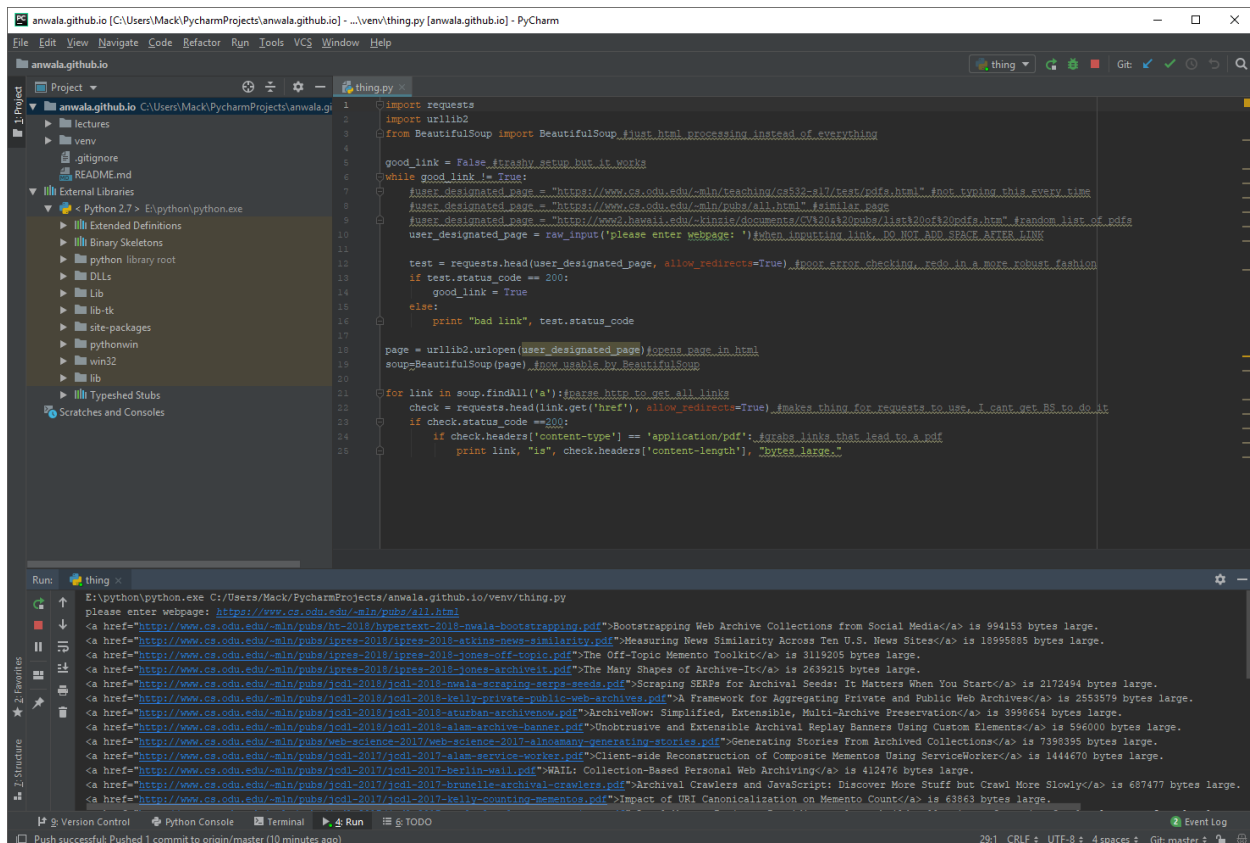


```
1 import requests
2 import urllib2
3 from BeautifulSoup import BeautifulSoup #just html processing instead of everything
4
5 good_link = False #trashy setup but it works
6 while good_link != True:
7     user_designated_page = "https://www.cs.odu.edu/~mln/teaching/cs532-s17/test/pdfs.html" #not typing this every time
8     user_designated_page = "https://www.cs.odu.edu/~mln/pubs/all.html" #similar page
9     user_designated_page = "http://www2.hawaii.edu/~kinis/documents/CV%20%20pubs/list%20of%20pdfs.htm" #random list of pdfs
10    user_designated_page = raw_input('please enter webpage: ') #when inputting link do NOT add space after link
11
12    test = requests.head(user_designated_page, allow_redirects=True) #poor error checking, redo in a more robust fashion
13    if test.status_code == 200:
14        good_link = True
15    else:
16        print "bad link", test.status_code
17
18    page = urllib2.urlopen(user_designated_page) #opens page in html
19    soup=BeautifulSoup(page) #now usable by BeautifulSoup
20
21    for link in soup.findAll('a'): #parse http to get all links
22        check = requests.head(link.get('href'), allow_redirects=True) #makes thing for requests to use, I cant get BS to do it
23        if check.status_code == 200:
24            if check.headers['content-type'] == 'application/pdf': #grabs links that lead to a pdf
25                print link, "is", check.headers['content-length'], "bytes large."
```

Run: thing

E:\python\python.exe C:\Users\Mack\PycharmProjects\anwala.github.io\venv\thing.py
please enter webpage: https://www.cs.odu.edu/~mln/pubs/all.html
Only One Out of Five Archived Web Pages Existed as Presented is 2104076 bytes large.
Quantifying Orphaned Annotations in Hypothes.is is 622981 bytes large.
arXiv:1512.06195 is 1748959 bytes large.
Detecting Off-Topic Pages in Web Archives is 4308768 bytes large.
Characteristics of Social Media Stories is 1274604 bytes large.
Web Archive Profiling Through CUX Summarization is 639001 bytes large.
tpdl-2015 papers is 2205546 bytes large.
Predicting Temporal Intention in Resource Sharing is 720476 bytes large.
Mobile Mink: Merging Mobile and Desktop Archived Webs is 1254605 bytes large.
How Well Are Arabic Websites Archived? is 709420 bytes large.
Improving Accessibility of Archived Raster Dictionaries of Complex Script Languages is 2350603 bytes large.

Process finished with exit code 0



```
1 import requests
2 import urllib2
3 from BeautifulSoup import BeautifulSoup #just html processing instead of everything
4
5 good_link = False #trashy setup but it works
6 while good_link != True:
7     user_designated_page = "https://www.cs.odu.edu/~mln/teaching/cs532-s17/test/pdfs.html" #not typing this every time
8     user_designated_page = "https://www.cs.odu.edu/~mln/pubs/all.html" #similar page
9     user_designated_page = "http://www2.hawaii.edu/~kinis/documents/CV%20%20pubs/list%20of%20pdfs.htm" #random list of pdfs
10    user_designated_page = raw_input('please enter webpage: ') #when inputting link do NOT add space after link
11
12    test = requests.head(user_designated_page, allow_redirects=True) #poor error checking, redo in a more robust fashion
13    if test.status_code == 200:
14        good_link = True
15    else:
16        print "bad link", test.status_code
17
18    page = urllib2.urlopen(user_designated_page) #opens page in html
19    soup=BeautifulSoup(page) #now usable by BeautifulSoup
20
21    for link in soup.findAll('a'): #parse http to get all links
22        check = requests.head(link.get('href'), allow_redirects=True) #makes thing for requests to use, I cant get BS to do it
23        if check.status_code == 200:
24            if check.headers['content-type'] == 'application/pdf': #grabs links that lead to a pdf
25                print link, "is", check.headers['content-length'], "bytes large."
```

Run: thing

E:\python\python.exe C:\Users\Mack\PycharmProjects\anwala.github.io\venv\thing.py
please enter webpage: https://www.cs.odu.edu/~mln/pubs/all.html
Bootstrapping Web Archive Collections from Social Media is 994153 bytes large.
Measuring News Similarity Across Ten U.S. News Sites is 1899588 bytes large.
The Off-Topic Memento Toolkit is 3119205 bytes large.
The Many Shapes of Archive-It is 2639215 bytes large.
Scraping SEPs for Archival Seeds: Is Browsers When You Start is 2172494 bytes large.
A Framework for Aggregating Private and Public Web Archives is 2553579 bytes large.
ArchivalNow: Simplified, Extensible, Multi-Archive Preservation is 3998654 bytes large.
Unobtrusive and Extensible Archival Replay Banners Using Custom Elements is 596000 bytes large.
Generating Stories From Archived Collections is 7398395 bytes large.
Client-side Reconstruction of Composite Mementos Using ServiceWorker is 1444670 bytes large.
WAIL: Collection-Based Personal Web Archiving is 412476 bytes large.
Archival Crawlers and JavaScript: Discover More Stuff but Crawl More Slowly is 687477 bytes large.
Impact of URI Canonicalization on Memento Count is 63863 bytes large.

Push successful: Pushed 1 commit to origin/master (10 minutes ago)

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:

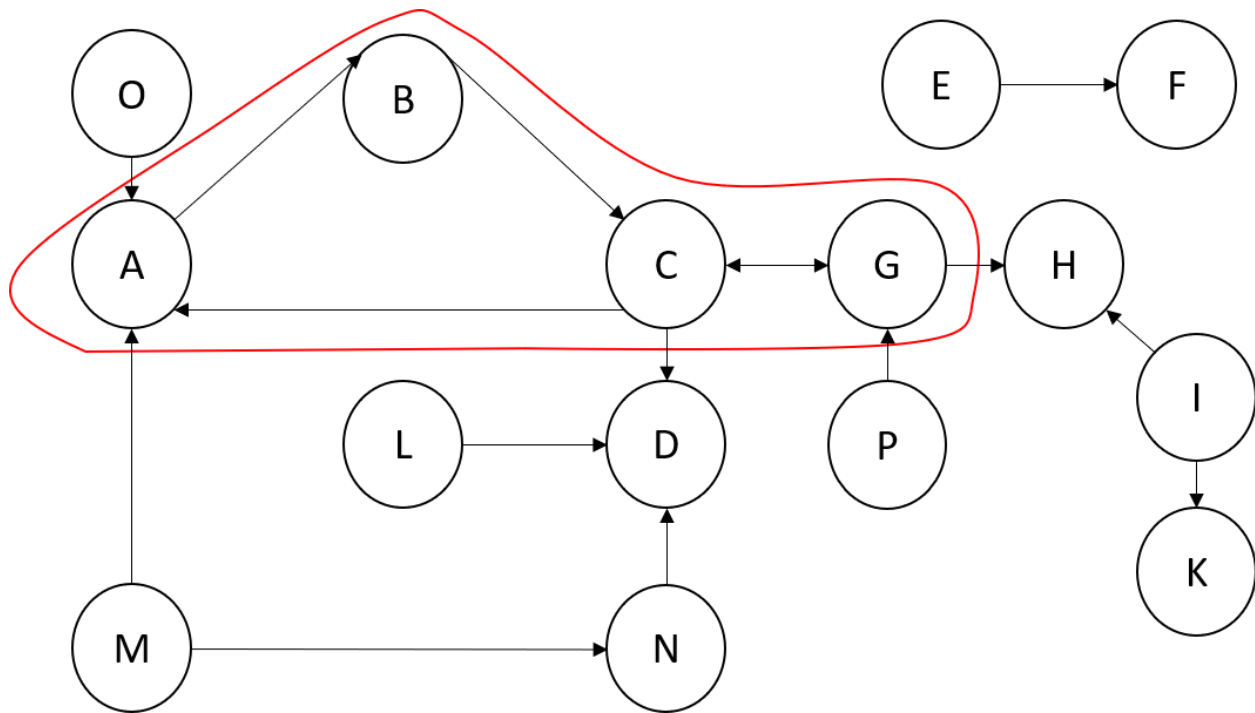
```
A --> B
B --> C
C --> D
C --> A
C --> G
E --> F
G --> C
G --> H
I --> H
I --> K
L --> D
M --> A
M --> N
N --> D
O --> A
P --> G
```

For the above graph, give the values for:

```
IN:
SCC:
OUT:
Tendrils:
Tubes:
Disconnected:
```

Solution

The solution to the problem is to create a graph of the connections, and then fill out the value sets with the criteria given on <https://www.harding.edu/fmccown/classes/archive/comp475-s13/web-structure-homework.pdf>. I did it SCC first, then IN and OUT, then tendrils, tubes, and disconnected pages.



SCC: A, B, C, G

In: O, M, P

Out: D, H, K

Tendrils: I, L

Tubes: N

Disconnected: E, F

A, B, C, and D are all strongly connected, as they can reach and be reached by any other page within the group. O, M, and P have no in-links and point to either SCC or a tube. D, H and K have no out-links and have in-links from either SCC or a tendril. I and L both have no in-links and point only to OUT. N has only in-links from IN and out-links to OUT. E and F are totally disconnected from the SCC.

References

- [1] Python slideshow https://drive.google.com/file/d/16b7VpKcdaacDxcVlaLv_eV41xULVimH8/view.
Accessed 1-30-2019
- [2] urllib3 user guide <https://urllib3.readthedocs.io/en/latest/user-guide.html> Accessed 1-31-2019
- [3] BeautifulSoup documentation <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> Accessed
1-31-2019