

Web Science cs532: Assignment #1

Due on Thursday, January 28, 2016

Dr. Michael.L. Nelson 4:20pm

Zetan Li

Contents

Problem 1	3
Problem 2	4
Problem 3	7

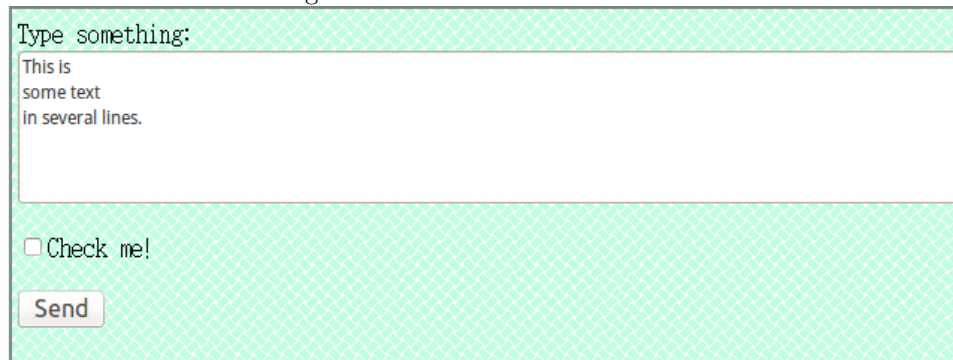
Problem 1

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

The original form is on the
<https://www.cs.tut.fi/~jkorpela/forms/testing.html>

Figure 1: The form to receive data



Type something:

This is
some text
in several lines.

☐ Check me!

Send

Using code(And save it in a bash for convenience):

```
curl -i -d "Comments=This%20is
```

```
Zetan's
```

```
test%20line" -d "box=yes" http://www.cs.tut.fi/cgi-bin/run/~jkorpela/echo.cgi
```

The option **-i** requires the header of the response, and the option **-d** pushes the data in to the form.

And the form will return a table contains all the data I submitted.

Also, the console shows the progress while waiting for response.

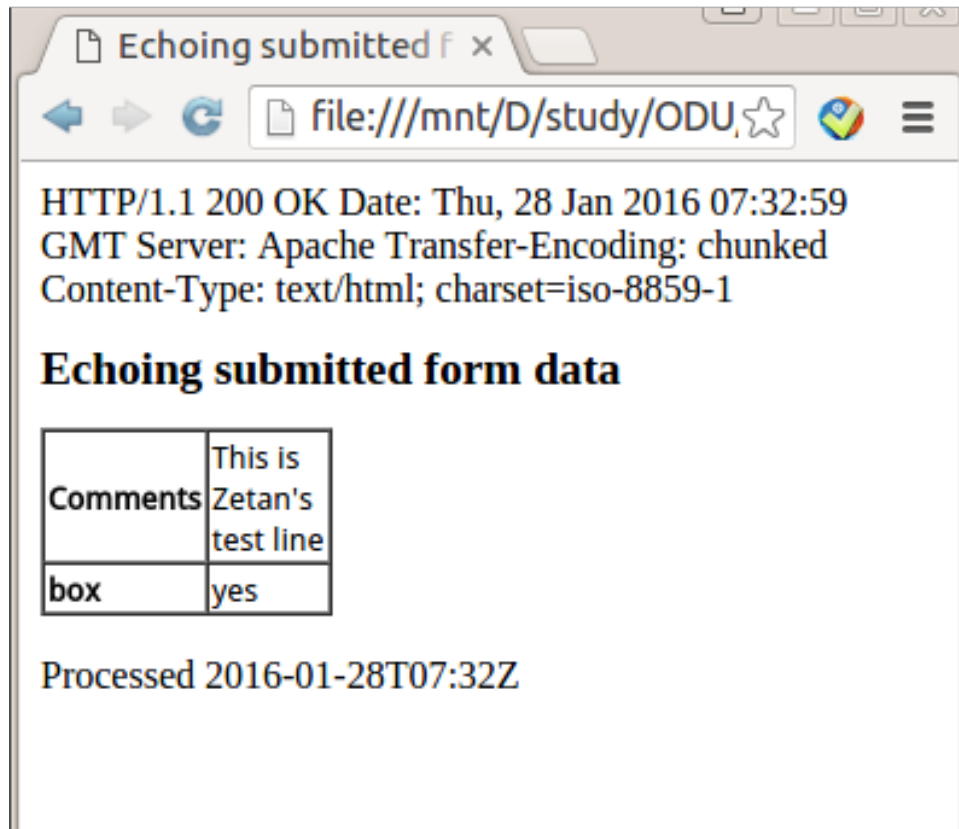
Figure 2: The console shows we are downloading response page

```

neo@TheMatrix:/mnt/D/study/ODU/web/a1$ ./a1_p1.sh
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           500    46    570   52  --:--:--  --:--:--  --:--:--  572
neo@TheMatrix:/mnt/D/study/ODU/web/a1$

```

Figure 3: The response in browser



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-s16/test/pdfs.html>

SOLUTION

First, we choose the requests package instead of urllib2. **requests** can fetch redirect link automatically, so we don't have to deal with the 300 response in code.

Then we are going to read the header of the response, get the value of “Content-type” field, if it's “Application/pdf”, then it is a pdf file.

The third step is to check whether the link to pdf is available. If so, get the “content-length” field as its size and print the result.

Listing 1: python script to list the pdf link

```
#!/usr/bin/env python
import requests
import sys
from bs4 import BeautifulSoup

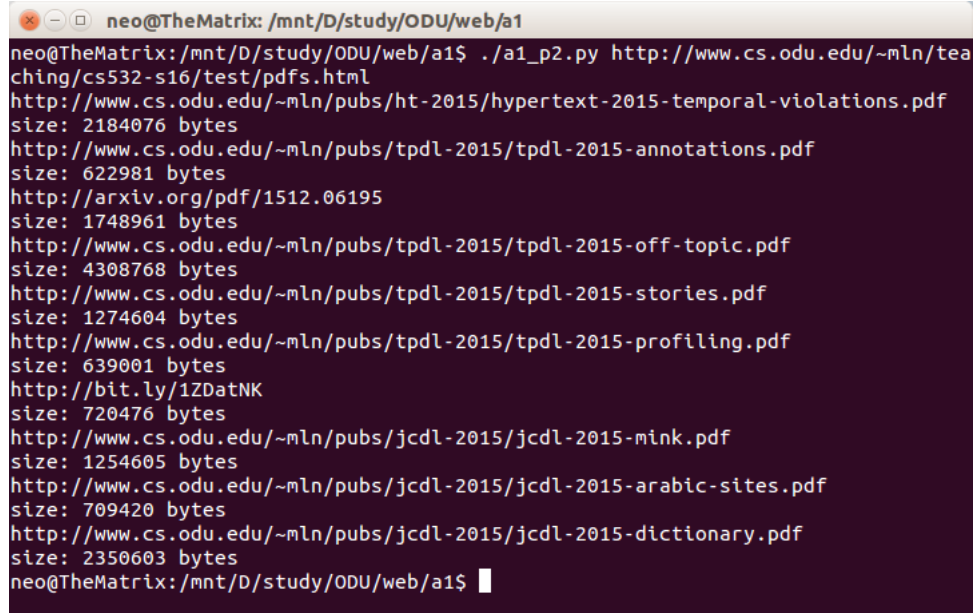
5
def getRealLocation(url):
    try:
        r=requests.get(url)
        if r.status_code==200:
10             return url,r
        elif r.status_code>=300 and r.status_code<400:
            newUrl=r.headers['location']
            return digRealLocation(newUrl)
        else:
15             return None,None
    except Exception as e:
        #print(e.message)
        return None,None

20 #script entry
if(len(sys.argv)!=2):
    print("Usage: a1_p2.py [url]")
    sys.exit(0);
url,_=getRealLocation(sys.argv[1])
25 if url is None:
    print("Invalid url")
    sys.exit(-1)

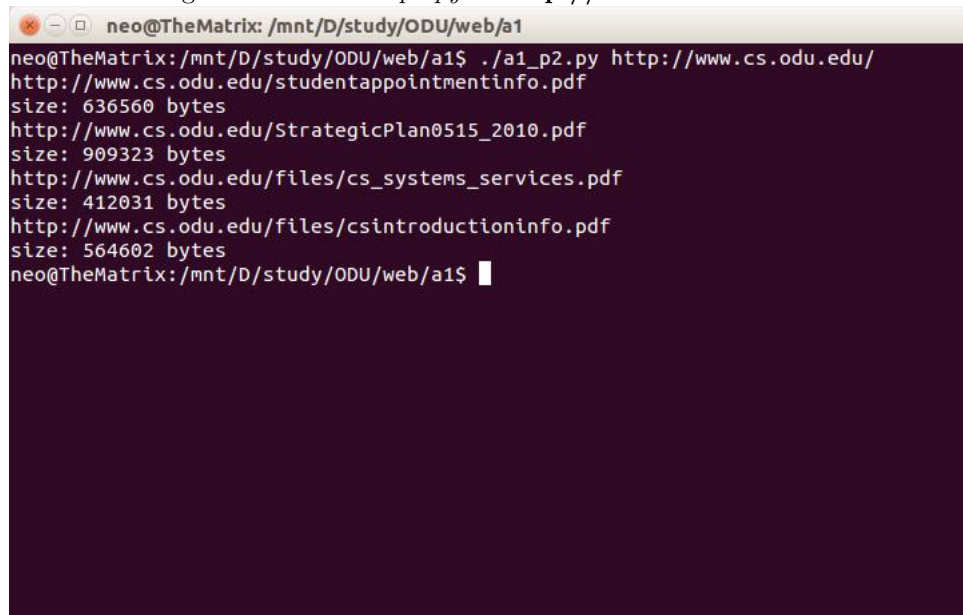
soup=BeautifulSoup(requests.get(url).text)
30 pdflinks=[]
for link in soup.find_all('a'):
    realUrl,response=getRealLocation(link.get('href'))
    if realUrl is not None:
        if response.headers['Content-Type']=='application/pdf':
35             pdflinks.append([realUrl,response.headers['Content-Length']])

if len(pdflinks)>0:
    for unit in pdflinks:
        print(unit[0])
        print("size: %s bytes"%(unit[1]))
40
else:
    print('There\'s no pdf on this page')
```

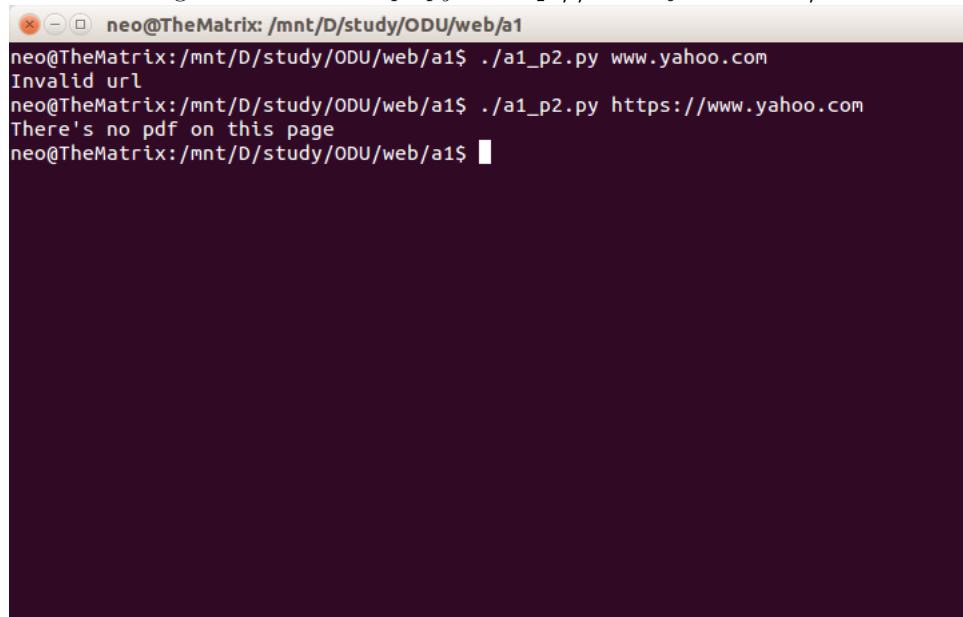
Below is the result of testing *a1_p2.py* on 3 different URIs:

Figure 4: Test of *a1_p2.py* at <http://www.cs.odu.edu/~mln/teaching/cs532-s16/test/pdfs.html>A terminal window titled 'neo@TheMatrix: /mnt/D/study/ODU/web/a1' shows the execution of the script *a1_p2.py*. The script processes a list of URLs and reports the size of each downloaded PDF file in bytes. The URLs include various academic papers and documents from ODU and arXiv. The terminal output is as follows:

```
neo@TheMatrix: /mnt/D/study/ODU/web/a1$ ./a1_p2.py http://www.cs.odu.edu/~mln/teaching/cs532-s16/test/pdfs.html
http://www.cs.odu.edu/~mln/pubs/ht-2015/hypertext-2015-temporal-violations.pdf
size: 2184076 bytes
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-annotations.pdf
size: 622981 bytes
http://arxiv.org/pdf/1512.06195
size: 1748961 bytes
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-off-topic.pdf
size: 4308768 bytes
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-stories.pdf
size: 1274604 bytes
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-profiling.pdf
size: 639001 bytes
http://bit.ly/1ZDatNK
size: 720476 bytes
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-mink.pdf
size: 1254605 bytes
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-arabic-sites.pdf
size: 709420 bytes
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-dictionary.pdf
size: 2350603 bytes
neo@TheMatrix: /mnt/D/study/ODU/web/a1$
```

Figure 5: Test of *a1_p2.py* at <http://www.cs.odu.edu>A terminal window titled 'neo@TheMatrix: /mnt/D/study/ODU/web/a1' shows the execution of the script *a1_p2.py*. The script processes a list of URLs and reports the size of each downloaded PDF file in bytes. The URLs include various academic papers and documents from ODU. The terminal output is as follows:

```
neo@TheMatrix: /mnt/D/study/ODU/web/a1$ ./a1_p2.py http://www.cs.odu.edu/
http://www.cs.odu.edu/studentappointmentinfo.pdf
size: 636560 bytes
http://www.cs.odu.edu/StrategicPlan0515_2010.pdf
size: 909323 bytes
http://www.cs.odu.edu/files/cs_systems_services.pdf
size: 412031 bytes
http://www.cs.odu.edu/files/csintroductioninfo.pdf
size: 564602 bytes
neo@TheMatrix: /mnt/D/study/ODU/web/a1$
```

Figure 6: Test of *a1_p2.py* at <http://www.yahoo.com/>

```
neo@TheMatrix: /mnt/D/study/ODU/web/a1
neo@TheMatrix:/mnt/D/study/ODU/web/a1$ ./a1_p2.py www.yahoo.com
Invalid url
neo@TheMatrix:/mnt/D/study/ODU/web/a1$ ./a1_p2.py https://www.yahoo.com
There's no pdf on this page
neo@TheMatrix:/mnt/D/study/ODU/web/a1$
```

Problem 3

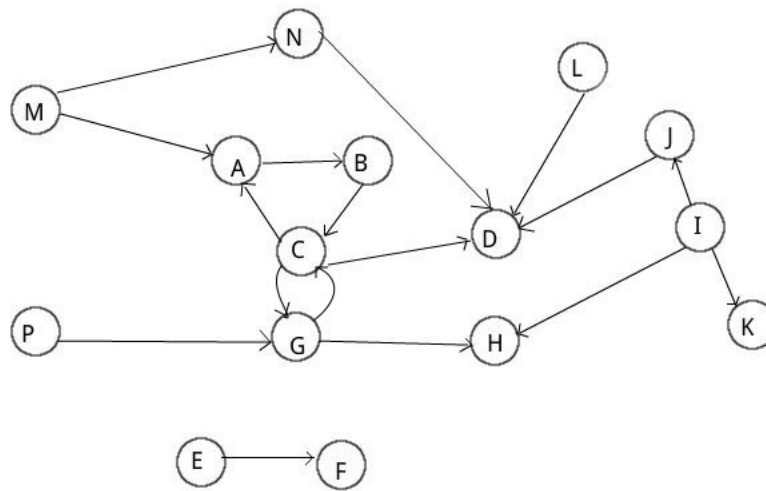
Consider the “bow-tie” graph in the Broder et al. paper (fig 9):

<http://www9.org/w9cdrom/160/160.html>

Now consider the following graph:

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

Figure 7: The bow-tie graph



For the above graph, give the values for:

IN: {M,O,P} The node that can reach the SCC, but can't reach from SCC[1]

SCC: {A,B,C,G} The core of the net, every node can reach each other[1]

OUT: {D,H} The node that can reach from the SCC, but can't link back to SCC[1]

Tendrils: {I,J,K,L} nodes that cannot reach the SCC, and cannot be reached from the SCC[1]

Tubes: {N} A portion of IN to a portion of OUT without touching SCC[1]

Disconnected: {E,F} All the other nodes that cannot reach the nodes mentioned above[1]

References

- [1] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. *Graph structure in the web*, 2000 (accessed January 27, 2016).