# Assignment 1 CS532-s16: Web Sciences

CS532-s16: Web Sciences
Spring 2016
John Berlin
Generated on January 28, 2016

## Question 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.

#### Answer

To post data to form using curl we must first consult the man pages to see what options are available.

From the man pages of curl:

```
-d, --data <data>
```

(puHTTP) Sends the specified data in a *POST request to the HTTP server*, in the same way that a browser does when a user has filled in an HTML form and presses the submit button. This will cause curl to pass the data to the server using the content-type application/x-www-form-urlencoded. Compare to -F, --form.

The man pages of curl state it can be used to post data to a form using this syntax:

```
curl -d "formValX=x&formValY=y" http://www.someform.com
```

From my recent work with the WS-DL tool mink I believe we can use curl send a uri to WebCite for archival using this command:

```
curl -I -d "url=http://docs.python-requests.org/en/latest/&email=jberlin@cs.odu.edu" http://webcitation.org/archive
```

The curl commands posts and we receive the following response:

```
HTTP/1.1 200 OK
Date: Thu, 28 Jan 2016 01:40:12 GMT
Server: Apache/2.0.63 (Unix) mod_ssl/2.0.63 OpenSSL/0.9.8e-fips-
rhel5 mod_auth_passthrough/2.1 mod_bwlimited/1.4 PHP/5.2.9
X-Powered-By: PHP/5.2.9
Content-Length: 5993
Content-Type: text/html
```

The form that was used is shown in Figure 1

WebCite archive p WebCite arch									
www.webcitation.org/archive							E C 0 4	Q Search	
■ READPAPERS ▼	<b>≡</b> schoolSites ▼	Most Visited ▼	Getting Started	JSHint Documentation	n	<b>■</b> conspiracy <del>▼</del>	<b>≡</b> streams ▼	<b>™</b> NO CLEAN SINGING <b>™</b>	Chegg Tu
		HOME FAQ NEWS APPLY MEMBERS SEARCH COMB ARCHIVE BOOKMARKLET							
		WebCite <sup>®</sup> archi	ve form						
		This page allows you to submit a single URL for instant archiving with WebCitle®, a member of the International Internet Preservation Consortium. Archiving in WebCitle® allows anybody (p. stable version of a Web page (including Blogs, Wikl, PDF file, and other webdocuments), making it "citable" in an academic context. It also provides the cited author and the academic communication of a Web page (including Blogs, Wikl, PDF file, and other webdocuments), making it "citable" in an academic context. It also provides the cited author and the academic communication of a Web page (including Blogs, Wikl, PDF file, and other webdocuments), making it "citable" in an academic context. It also provides the cited author and the academic communication of a Web page (including Blogs, Wikl, PDF file, and other webdocuments), making it "citable" in an academic context. It also provides the cited author and the academic context.							
		The content of the page requested below will be immediately archived, including any inline images and / or media (up to a maximum size). WebCite® automatically determines if the webpage is already archived copy. As part of the archiving process, an e-mall will be sent to the address of the citing author below, containing the unique URL that can be used to access the archived content, whus your e-mall address for any other purposes than sending a confirmation or failure email.							
		URL to Archive [url]:							
		Your (citing author) E-m	ail Address [email]:						
		Language of archived f	ile:	Default by	vebsite ‡				
					n which language you want the web e language is used when preferred			ive the webpage in that language.	
		Metadata (optional	1)						
		These are Dublin Core	e elements. Entering the	ese will help you to correctly o	you to correctly cite the URL. Any information here will take precedence over metadata extracted from the cited webpage.				
		Title [title]:							
		Author(s) [author]:		Lastname, Fir	stname; Separate multiple authors	s with ";". Example:	Doe, John; Smith,	R	
		Cited Author's E-mail [a	uthoremail]:						
		Publisher [source]:							
		Date [date]:							
		()		Date of public	ation, if known: yyyy-mm-dd or yy	yyy-mm or yyyy			
		Subject Keywords [subj	ect]:	Konnwords (o.s	. MeSH keywords). Separate mul	tiple konwords with !	":" Evample: inform	natios: hoalth	
				Submit	. moor keywords). Separate mui	upic neywords Willi	, . Example. illioni	nanos, nealti	
		Instructions for webaut	thors (cited authors) wh	o want to link directly to this fo	rm				

Figure 1: WebCite Archival Form

The -I option from the man pages for curl:

#### -I, --head

(HTTP/FTP/FILE) Fetch the HTTP-header only!
HTTP-servers feature the command HEAD which this uses to get nothing but the header of a document. When used on an FTP or FILE file, curl displays the file size and last modification time only.

As expected only the headers were returned to in the response. To recive back the html the -I option is removed and curl is executed with these arguments:

```
curl -d "url=http://docs.python-requests.org/en/latest/&email=
    jberlin@cs.odu.edu" http://webcitation.org/archive
```

The rendered html returned by this curl command is shown in Figure 2

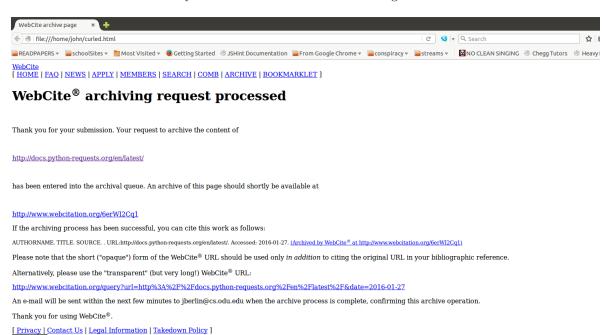


Figure 2: The output of the curl command, rendered in Mozilla Firefox

Except for archived content, this site is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 2.5 License.

### Question 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".)
- - http://www.cs.odu.edu/~mln/teaching/cs532-s16/test/pdfs.html

#### Answer

```
#!/usr/bin/env python3
            import argparse
            import re
            from collections import deque
            import requests
  6
            from bs4 import BeautifulSoup
              \begin{array}{lll} -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ & -9 \\ &
10
            # my standard url regex found a while ago
13
14
             url_re = re.compile(reg_s , re.IGNORECASE)
16
17
             def print_headers(r):
                           print ("printing headers for url=", r.url)
18
 19
                           for headerk, headerv in r.headers.items():
                                         print(headerk + ":" + headerv)
20
21
                           print("++++++++++++++++++++++++")
22
23
             def ispdf(rq):
24
                           if rq.headers['Content-type'].lower() in 'application/pdf':
25
                                        return True
26
                           else:
27
                                         return False
28
29
30
             def strip_href(text, que, saw):
31
32
                           try:
                                       s = BeautifulSoup(text, 'html5lib')
33
34
                           except:
                                        # just because if this fails there are problems
35
                                         s = BeautifulSoup(text)
36
                           all_a = s.find_all('a', href=True)
37
                           for link in map(lambda a: a['href'], all_a):
```

```
if link not in saw:
39
                if url_re.match(link):
40
                    que.append(link)
41
                else:
42
                     print ("The input uri %s failed to pass my regex " %
43
                          link, reg_s)
44
        return s
45
46
   if __name__ == '__main__':
47
        parser = argparse.ArgumentParser()
48
        parser.add_argument("uri", type=str, help="the url to extract
49
        parser.add_argument("-v", help="verbose", action="store_true")
50
        parser.add_argument("-ph", help="print all headers", action="
            store_true")
        args = parser.parse_args()
53
54
       # your on Ubuntu and you will like it
        useragent = 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0)
            Gecko/20100101 Firefox/44.01'
        session = requests. Session()
57
        session.headers.update({ 'User-Agent': useragent})
58
        seen = set()
59
       q = deque()
60
       q.append(args.uri) # ("http://tinyurl.com/gv8jnso")
61
        uri = q.popleft()
62
63
        if re.match(url_re, uri):
64
65
            if args.v:
                print("it matches")
66
        else:
67
            print ("The input uri %s failed to pass my regex " % uri,
68
                reg_s)
69
        request = session.get(uri)
70
        """:type : requests.Response """
71
        if ispdf(request):
72
            print ("The link %s was a link to an actual pdf file " % uri
73
                  ("but it was misleading %s its length is " % request.
74
                       url)
                   if uri != request.url else "its length is",
                   request.headers['Content-Length']
76
77
            if request.is_redirect or request.is_permanent_redirect:
78
                print("It was dirty reditect ;)")
79
            exit()
80
        soup = strip_href(request.text, q, seen)
81
82
        if args.ph:
            print_headers(request)
83
84
        while True:
85
86
            try:
                uri = q.popleft()
87
88
            except IndexError:
```

```
break
89
             if re.match(url_re, uri):
90
                 if args.v:
91
                     print ("the uri %s was valid" % uri)
92
             else:
93
                 print ("Bad url %s" % uri)
94
                 continue
             seen.add(uri)
96
97
98
             request = session.get(uri)
             """:type: requests.Response """
99
100
             if args.ph:
                 print_headers(request)
             if ispdf(request):
104
                 if request.ok:
                     print ("The link %s was a link to an actual pdf file
106
                            \% uri,
                            ("but it was misleading %s its length is " %
                                request.url)
                            if uri != request.url else "its length is",
108
                            request.headers['Content-Length']
110
                     continue
             if args.v:
                 print ("so far I have processed these uris: ", seen)
114
                 print ("I still have these to go", q)
117
                 print("\nHey were are ok! %i" % request.status_code,
118
                     Done going down the rabit hole for %s\n" % uri)
             {\tt elif \ request.is\_permanent\_redirect \ or \ request.is\_redirect:}
119
                 soup = strip_href(request.text, q, seen)
120
        session.close()
```

Listing 1: Python program for extracting links from a web page to look for pdf files

The code is shown in Listing 1 starting on page 4:

- Takes an argument uri specifying the web page to extract links from (line 49)
- Takes an optional argument to print extra data (line 50)
- Takes an optional argument to print the headers of all received requests (line 51)
- Open a session for (line 57), Set our user agent (line 58), Download starting URI (line 70), check to see if won on first uri(line 72), otherwise get all hrefs(line 81), and consume each one in web page informing user if pdf was found(lines 87-120)

The core libraries used are

- argparse: I do not like sys.args and closest thing to Commons CLI
- re: for regular expression support. Do not want some arbitrary href=file.js or malformed uri
- requests: better urllib, it will handle sneaky redirects automatically
- bs4: get the web soup even tho I like my soup hot and sour

strip\_href (lines 31 to 44) takes the html text and retrieves all a tags and extracts the uri content and adds it the queue. Return the hot soup after extraction.

Since requests library handles the redirects for us a check to see if there was some trickery the statement uri != resquest.url checks if the uri found in the extraction is not the same as was the final link. If its not a redirect happened for sure.

Executing: python3 follow Links.py http://www.cs.odu.edu/<br/> mln/teaching/cs532-s16/test/pdfs.html gives the output

The link http://www.cs.odu.edu/~mln/pubs/ht-2015/hypertext-2015-temporal-violations.pdf was a link to an actual pdf file its length is 2184076

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-annotations.pdf was a link to an actual pdf file its length is 622981

The link http://arxiv.org/pdf/1512.06195 was a link to an actual pdf file but it was misleading http://arxiv.org/pdf/1512.06195.pdf its length is 1748961

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-off-topic.pdf was a link to an actual pdf file its length is 4308768

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-stories.pdf was a link to an actual pdf file its length is 1274604

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-profiling.pdf was a link to an actual pdf file its length is 639001

The link http://bit.ly/1ZDatNK was a link to an actual pdf file but it was misleading http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-temporal-intention.pdf its length is 720476

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-mink.pdf was a link to an actual pdf file its length is 1254605

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-arabic-sites.pdf was a link to an actual pdf file its length is 709420

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-dictionary.pdf was a link to an actual pdf file its length is 2350603

Now when executing it with a link straight to a pdf such as http://arxiv.org/pdf/1512.06195.pdf

The link http://arxiv.org/pdf/1512.06195.pdf was a link to an actual pdf file its length is 1748961 No other links are in a pdf have nice day

For the third and final link I saw one in the full output of the program when running on the required link that looked oddly funky lets try it: http://bit.ly/jcdl-pdf

Hey were are ok! 200 Done going down the rabit hole for http://twitter.com/webscidl

The link

http://www.cs.odu.edu/~mln/pubs/ht-2015/hypertext-2015-temporal-violations.pdf was a link to an actual pdf file its length is 2184076

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-annotations.pdf was a link to

The link http://arxiv.org/pdf/1512.06195 was a link to an actual pdf file but it was misleading http://arxiv.org/pdf/1512.06195.pdf its length is 1748961

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-off-topic.pdf was a link to an

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-stories.pdf was a link to an actual pdf file its length is 1274604

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-profiling.pdf was a link to an actual pdf file its length is 639001

The link http://bit.ly/1ZDatNK was a link to an actual pdf file but it was misleading

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-mink.pdf was a link to an actual pdf file its length is 1254605

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-arabic-sites.pdf was a link to an actual pdf file its length is 709420

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-dictionary.pdf was a link to an actual pdf file its length is 2350603

Oddly familiar would you not agree with me.

# Question 3

```
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
For the above graph, give the values for:
 IN:
 SCC:
 OUT:
Tendrils:
Tubes:
Disconnected:
• IN: [M,O,P] nodes(pages) that can reach the SCC, but cannot be reached
 from it
```

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

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

• SCC: [A,B,C,G] central core, all of whose pages can reach one another

• OUT: [D,H]: pages that are accessible from the SCC, but do not link back

• Tendrils: [I,J,K,L] pages that cannot reach the SCC, and cannot be

along directed links

reached from the SCC

- Tubes: [N] passage from a portion of IN to a portion of OUT without touching SCC
- **Disconnected**: [E,F] pages who do not fit the descriptions above or has no links to the core or other pages

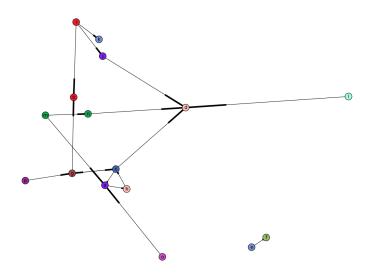


Figure 3: A different means of looking at the graph

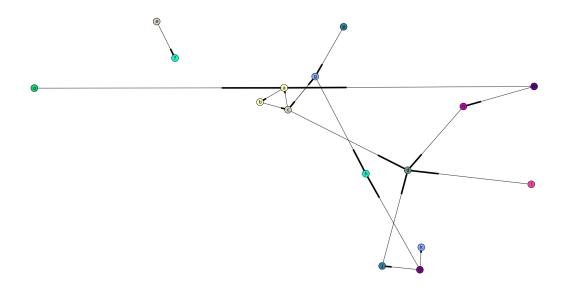


Figure 4: A different means of looking at the graph 2  $\,$