Assignment 1

CS 595: Introduction to Web Science

Fall 2013

Shawn M. Jones

Finished on September 12, 2013

1

Question

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" (e.g., save it to a file and then view that file in a browser and take a screen shot).

Answer

The `curl` command can be used to post data to a form in several ways, but the most easy is the following:

```
curl --data 'variable1=value1&variable2=value2' \
  http://www.example.com
```

The more difficult prospect is finding a URI on the Internet that accepts form submission without authentication. Fortunately, my recent work in using Mediawiki has led me to a web application that accepts formatted CSV and returns a table formatted in Mediawiki syntax. The URI is http://area23.brightbyte.de/csv2wp.php. The rendered form is shown in Figure 1.

After reading the HTML, the form action is not shown, so the URI to POST to is also http://area23.brightbyte.de/csv2wp.php.

I posted the CSV data "1,2" to the form like so:

```
curl -i --data 'csv=1,2&separator=,&quotes=&quot;&escape=\&break
    =SPACE&convert=html&output_encoding=UTF-8&to_wp=Convert+To+
    Mediawiki!'
    http://area23.brightbyte.de/csv2wp.php
```

and got back the following response from `curl`:

```
HTTP/1.1 200 OK
Date: Sat, 31 Aug 2013 23:20:02 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze16
Content-Description: csv2wp-1377991202.UTF-8.txt
Content-Length: 23
Content-Type: text/plain; charset=UTF-8

{|
|1
|2
|----
|}
```

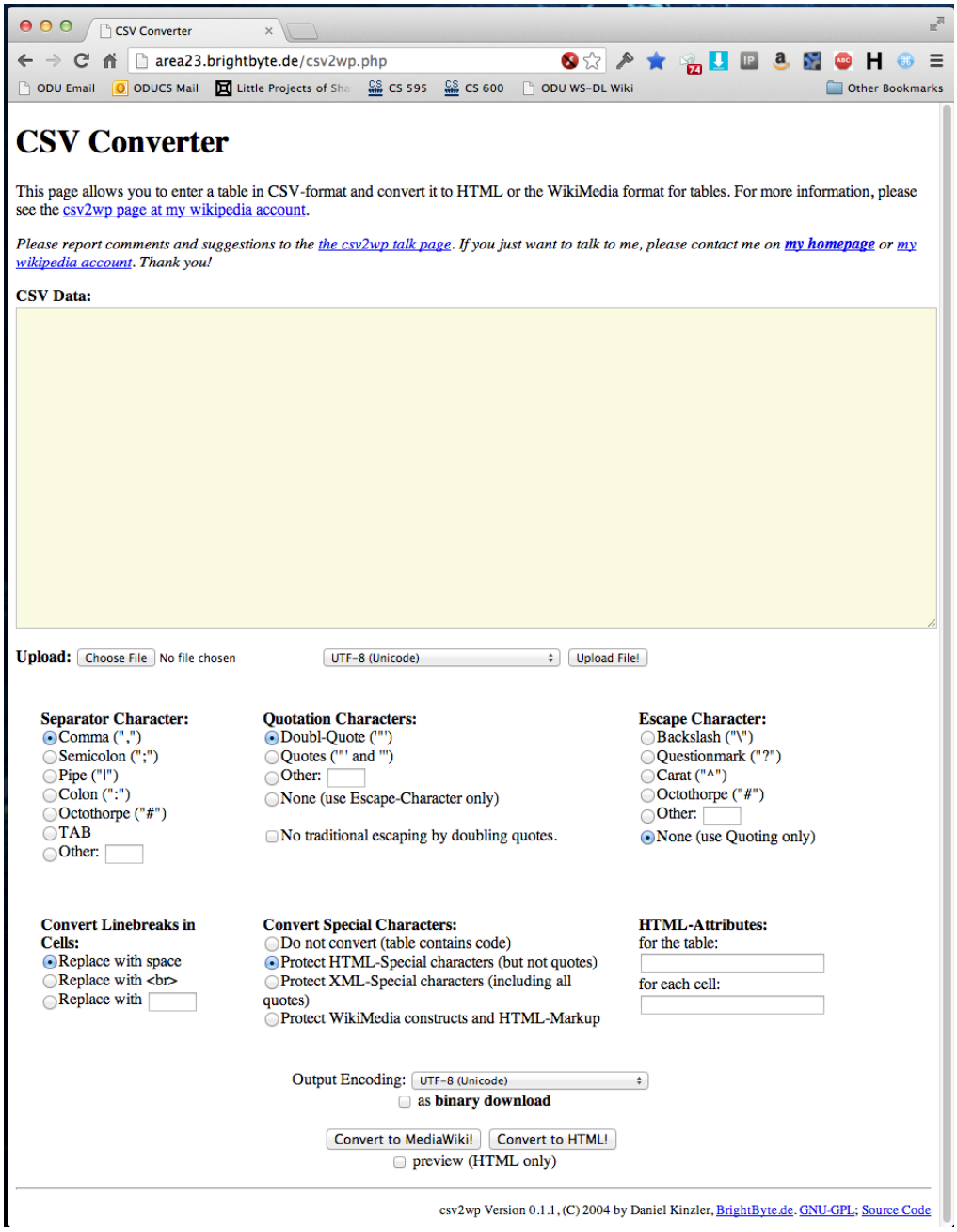

Figure 1: The form for http://area23.brightbyte.de/csv2wp.php, rendered in Google Chrome

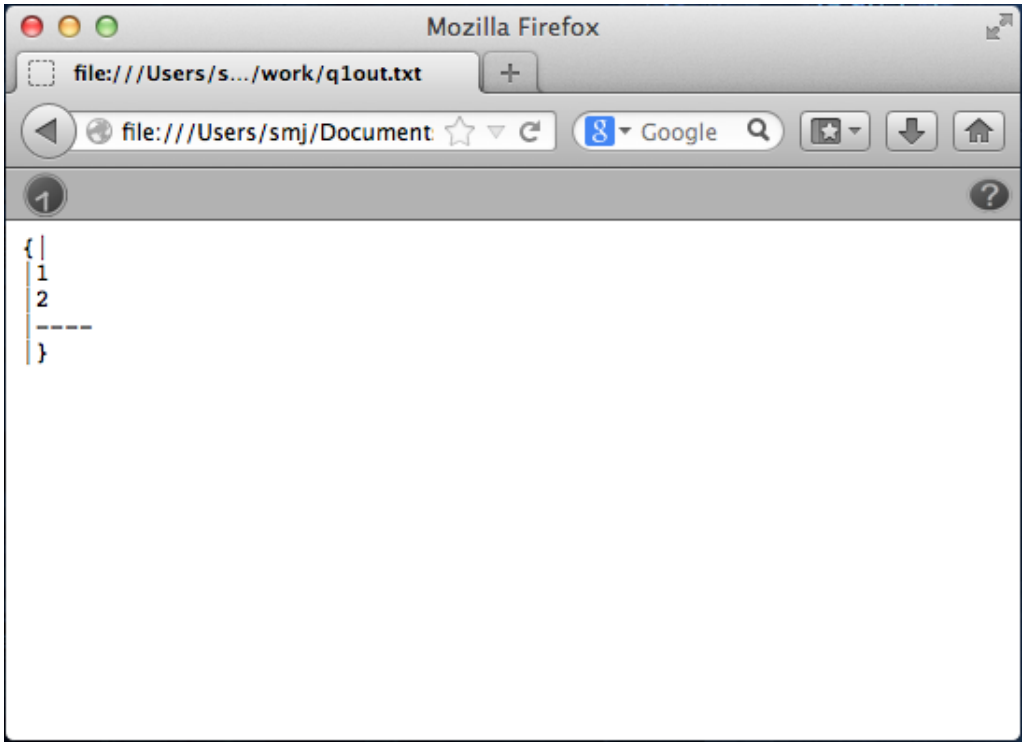

Figure 2: The output of the curl command, rendered in Mozilla Firefox

Executing again, without -i, allows me to redirect the data to a file:

```
curl --data 'csv=1,2&separator=,&quotes=&quot;&escape=\&break=
    SPACE&convert=html&output_encoding=UTF-8&to_wp=Convert+To+
    Mediawiki!'
   http://area23.brightbyte.de/csv2wp.php \
   > work/q1out.txt
```

which renders in the browser like shown in Figure 2.

2

Question

Original:

```
Write a Python program that:
1. takes one argument, like "Old Dominion" or "Virginia Tech"
2. takes another argument specified in seconds (e.g., "60" for
   one minute).
3. takes a URI as a third argument:
   http://sports.yahoo.com/ncaa/football/scoreboard
   or
   http://sports.yahoo.com/ncaa/football/scoreboard?w=1&c=all&y=2012
   or
   http://sports.yahoo.com/ncaa/football/scoreboard?w=2&c=all&y=2012
   etc.
4. downloads the URI, finds the game corresponding to the team
   argument, prints out the current score (e.g., "Old Dominion 27,
   East Carolina 17), sleeps for the specified seconds, and then
   repeats (until control-C is hit).
```

Updated Wed, September 4 10:31:37 EDT 2013:

```
Write a Python program that:
1. takes one argument, like "Old Dominion" or "Virginia Tech"
2. takes another argument specified in seconds (e.g., "60" for
   one minute).
3. takes a URI as a third argument, such as:
   http://scores.espn.go.com/ncf/scoreboard?confId=80&seasonYear=2013&seasonType=2&weekNumber=2
   or
   http://scores.espn.go.com/ncf/scoreboard?confId=80&seasonYear=2013&seasonType=2&weekNumber=1
   or
   http://scores.espn.go.com/ncf/scoreboard?confId=80&seasonYear=2012&seasonType=2&weekNumber=1
   etc.
4. downloads the URI, finds the game corresponding to the team
   argument, prints out the current score (e.g., "Old Dominion 38,
   East Carolina 52), sleeps for the specified seconds, and then
   repeats (until control-C is hit).

   You can use any source for college football box scores that you'd like.
```

Answer

```
#!/usr/local/bin/python3

import sys
import time
import urllib.request
import re
from bs4 import BeautifulSoup

def usage():
    print("")
    print("Usage:  " + sys.argv[0] + " <school name> <refresh> <
        uri>")
    print("  where:")
    print("    school name - name of school from which to
        display scores")
    print("    refresh - number of seconds to refresh the score"
        )
    print("    uri - URI to use for score lookup")
    print("")

def printCurrentScore(school1, score1, school2, score2):

    print(school1 + " " + score1 + ", " + school2 + " " + score2
        )

def fetchCurrentScore(uri):

    pagehandle = urllib.request.urlopen(uri)
    pagedata = pagehandle.readall()
    pagehandle.close()
    return pagedata


def getScoresFromPage(webpage, school):

    soup = BeautifulSoup(webpage)

    scripts = soup.find_all('script')

    # the HTML is buried in some JavaScript
    for script in scripts:
        if school in script.text:
            break

    # strip out all of the weird backslash JavaScript stuff
    htmlish = script.text.replace("\\n", '').replace('\\', '')
    soup = BeautifulSoup(htmlish)
```

```
44      rows = soup.find_all('tr')
45
46      for row in rows:
47          if school in row.text:
48              break
49
50      structuredData = row.text.replace('Final', '')
51      structuredData = re.sub(r'\([0-9]+\)', '', structuredData)
52      structuredData = re.sub(r'[ ]{2,}', ',', structuredData)
53      structuredData = structuredData.strip(',')
54      structuredData = structuredData.replace(' - ', ',')
55      structuredData = structuredData.split(',')
56
57      game = {}
58      game['school1'] = structuredData[0]
59      game['score1'] = structuredData[1]
60      game['score2'] = structuredData[2]
61      game['school2'] = structuredData[3]
62
63      return game
64
65  def main(args):
66
67      try:
68          school = args[1]
69          refresh = int(args[2])
70          uri = args[3]
71      except (IndexError, e):
72          usage()
73          sys.exit(1)
74
75      while(True):
76          webpage = fetchCurrentScore(uri)
77          game = getScoresFromPage(webpage, school)
78
79          printCurrentScore(
80              game['school1'], game['score1'], game['school2'],
                    game['score2']
81              )
82          time.sleep(refresh)
83
84
85  if __name__ == '__main__':
86      main(sys.argv)
```

Listing 1: Python program for acquiring NCAA Football scores from Yahoo sports

The code is shown in Listing 1 starting on page 6:

- Takes an argument specifying the school (line 68)
- Takes an argument specified in seconds (line 69)
- Takes a URI as a third argument (line 70)
- Downloads the URI (line 76), finds the game corresponding to the team argument (line 77), prints out the current score (line 79), sleeps for the specified seconds (line 82), and then repeats (line 75)

The core work is done in *getScoresFromPage* (lines 30 to 63). The trick to parsing the page is to realize that the actual data is inside HTML that is embedded in JavaScript defined inside one of the script tags. The loop on line 37 searches all of the script tags for the given school. Once the pertinent script tag is found, string replacement is done on line 42 to turn the JavaScript string into an actual HTML fragment, which is then fed into BeautifulSoup again, and a second search is performed on the actual entries.

The command is run like so:

```
./gimmescore.py "Oregon" 10 http://sports.yahoo.com/ncaa/
    football/scoreboard
```

And produces output like the following:

```
Nicholls 3, Oregon 66
Nicholls 3, Oregon 66
^CTraceback (most recent call last):
  File "./gimmescore.py", line 91, in <module>
  File "./gimmescore.py", line 87, in main

KeyboardInterrupt
```

After rerunning the script on September 9, 2013, it became evident that this implementation only worked for the September 2, 2013 and prior versions of the representation at http://sports.yahoo.com/ncaa/football/scoreboard. Since that date, changes in the HTML to this page have rendered the script useless.

This is a common problem with "screen scraping" web pages, resulting in the re-writing of many tools. Yahoo! has no obligation to continue to format their HTML as I expect, hence any such scripts can not be counted on to be permanent and should require frequent testing to remain valid.

Also, this script is very specific to this representation, meaning it will not work for a Fox Sports page or a CNN sports page.

A corrected version of the script (gimmescore2.py) is shown in Listing 2 on page 11. This script also includes improved error handling.

It is run and produces output like the following:

```
bash $ --> ./gimmescore2.py "San Diego St." 10 "http://sports.
    yahoo.com/college-football/scoreboard/?week=2&conf="
San Diego St. 7, Ohio St. 42
San Diego St. 7, Ohio St. 42
^CTraceback (most recent call last):
  File "./gimmescore2.py", line 85, in <module>
  File "./gimmescore2.py", line 81, in main

KeyboardInterrupt
```

and the following (for a different week):

```
bash $ --> ./gimmescore2.py "Virginia Tech" 10 "http://sports.
    yahoo.com/college-football/scoreboard/?week=1&conf="
Alabama 35, Virginia Tech 10
Alabama 35, Virginia Tech 10
Alabama 35, Virginia Tech 10
^CTraceback (most recent call last):
  File "./gimmescore2.py", line 98, in <module>
    main(sys.argv)
  File "./gimmescore2.py", line 92, in main
    time.sleep(refresh)
KeyboardInterrupt
```

If used against a school that isn't listed, it also gracefully recovers:

```
bash $ --> ./gimmescore2.py "South Harmon Institute of
    Technology" 10 "http://sports.yahoo.com/college-football/
    scoreboard/?week=2&conf="
School "South Harmon Institute of Technology" not found in
    scores this week, maybe try again another day?
```

It also gracefully recovers from a game that hasn't started yet:

```
bash $ --> ./gimmescore2.py "Vanderbilt" 10 "http://sports.yahoo
    .com/college-football/scoreboard/"
School "Vanderbilt" found, but game has not started yet
```

And, of course, reports the scores while the game is running:

```
bash $ --> ./gimmescore2.py "Troy" 60 "http://sports.yahoo.com/
    college-football/scoreboard/?conf=fbs_all"
Troy 14, Arkansas St. 20
Troy 14, Arkansas St. 20
Troy 14, Arkansas St. 20
Troy 20, Arkansas St. 20
Troy 20, Arkansas St. 20
Troy 21, Arkansas St. 20
^CTraceback (most recent call last):
  File "./gimmescore2.py", line 107, in <module>
    main(sys.argv)
  File "./gimmescore2.py", line 101, in main
    time.sleep(refresh)
KeyboardInterrupt
```

```
1  #!/usr/local/bin/python3
2
3  import sys
4  import time
5  import urllib.request
6  import re
7  from bs4 import BeautifulSoup
8
9  class GimmeScoreException(Exception):
10
11     def __init__(self, value):
12         self.value = value
13
14     def __str__(self):
15         return repr(self.value)
16
17 def usage():
18     print("")
19     print("Usage:  " + sys.argv[0] + " <school name> <refresh> <
           uri>")
20     print("  where:")
21     print("    school name - name of school from which to
           display scores")
22     print("    refresh - number of seconds to refresh the score"
           )
23     print("    uri - URI to use for score lookup")
24     print("")
25
26 def printCurrentScore(school1, score1, school2, score2):
27
28     print(school1 + " " + score1 + ", " + school2 + " " + score2
           )
29
30 def fetchCurrentScore(uri):
31
32     pagehandle = urllib.request.urlopen(uri)
33     pagedata = pagehandle.readall()
34     pagehandle.close()
35     return pagedata
36
37 def getScoresFromPage(webpage, school):
38
39     soup = BeautifulSoup(webpage)
40
41     trs = soup.find_all('tr')
42
43     for tr in trs:
44         if school in tr.text:
```

```
45              break
46
47      if not tr.text:
48          raise GimmeScoreException('School "' + school + '" not
                found in scores this week, maybe try again another
                day?')
49
50      if '@' in tr.text:
51          raise GimmeScoreException('School "' + school + '" found
                , but game has not started yet')
52
53      # get rid of unnecessary whitespace
54      structuredData = tr.text.strip()
55
56      # insert commas for delimiting
57      structuredData = structuredData.replace('\n\n\n', ',').
            replace('-', ',')
58
59      # get rid of the unneded 'Final', if game is finished
60      structuredData = structuredData.replace('\nFinal', '').strip
            ()
61
62      # get rid of the unneeded team ranking, if present
63      structuredData = re.sub(r'\([0-9]+\)', '', structuredData).
            strip()
64
65      # get rid of extra data at the beginning
66      structuredData = re.sub(r'^.*\n', '', structuredData).strip
            ()
67
68      # get rid of extra data at the end
69      structuredData = re.sub(r'\n.*$', '', structuredData).strip
            ()
70
71      # create a list of data
72      structuredData = structuredData.split(',')
73
74      # fill the game dictionary with the data
75      game = {}
76      game['school1'] = structuredData[0].strip()
77      game['score1'] = structuredData[1].strip()
78      game['score2'] = structuredData[2].strip()
79      game['school2'] = structuredData[3].strip()
80
81      return game
82
83  def main(args):
84
85      try:
```

```
86          school = args[1]
87          refresh = int(args[2])
88          uri = args[3]
89      except (IndexError, e):
90          usage()
91          sys.exit(1)
92
93      try:
94          while(True):
95              webpage = fetchCurrentScore(uri)
96              game = getScoresFromPage(webpage, school)
97
98              printCurrentScore(
99                  game['school1'], game['score1'], game['school2'
                        ], game['score2']
100                 )
101             time.sleep(refresh)
102     except GimmeScoreException as e:
103         print(e.value)
104
105
106 if __name__ == '__main__':
107     main(sys.argv)
```

Listing 2: Updated Python program for acquiring NCAA Football scores from Yahoo sports

3

Question

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

For the above graph, give the values for:

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

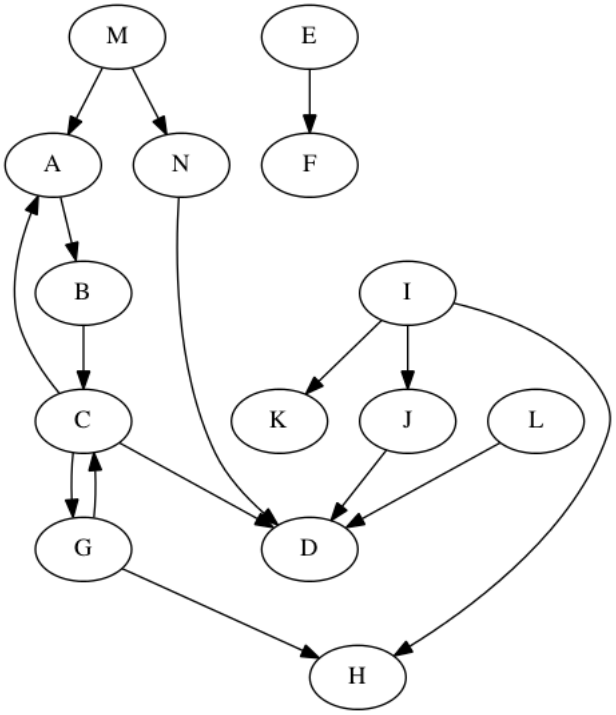

Figure 3: Drawing of the example graph

Answer

Figure 3 is a graph of the included points, generated using Graphviz.

The descriptions of each node is a matter of interpretation. Explanations have been provided for each value assignment.

IN: M

The IN components form the starting point for connection to the SCC[1]. They all sit at the start of the graph. In this graph, because of how the SCC and tubes are defined, the only IN component is M.

SCC: A, B, C, G

The Strongly Connected Component consists of those heavily linked items connected to from the list of nodes listed as part of IN[1]. It's an A, B, C, G world.

OUT: D

The OUT components exit the SCC, but do not link back to it[1]. The only member of OUT is D, who forms a sink from members of the SCC and the tendrils.

Tendrils: I, J, K, L

The *Tendrils* are pages that cannot reach the SCC or are not reached from the SCC[1]. The tendrils come from other graphs and only join the whole via D.

Tubes: N

Then there are *tubes*, which pass from IN to OUT without going through SCC[1]. N is a tube linking from M (from IN) to D (from OUT).

Disconnected: E, F

The Disconnected components link to no one in the graph, and stand alone. They are not defined explicitly by Broder, et. al, but their meaning is implied within the paper.

References

[1] Andrew Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. In *Ninth International World Wide Web Conference*, 2000.