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. 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 1:

The solution simply involves using curl with the -d or --data option to post data to a site, as seen in Figure 1. The command in full to post the name Bethany DeMerchant to the given test site is:

curl -d ‘fname=Bethany&lname=DeMerchant’ https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php

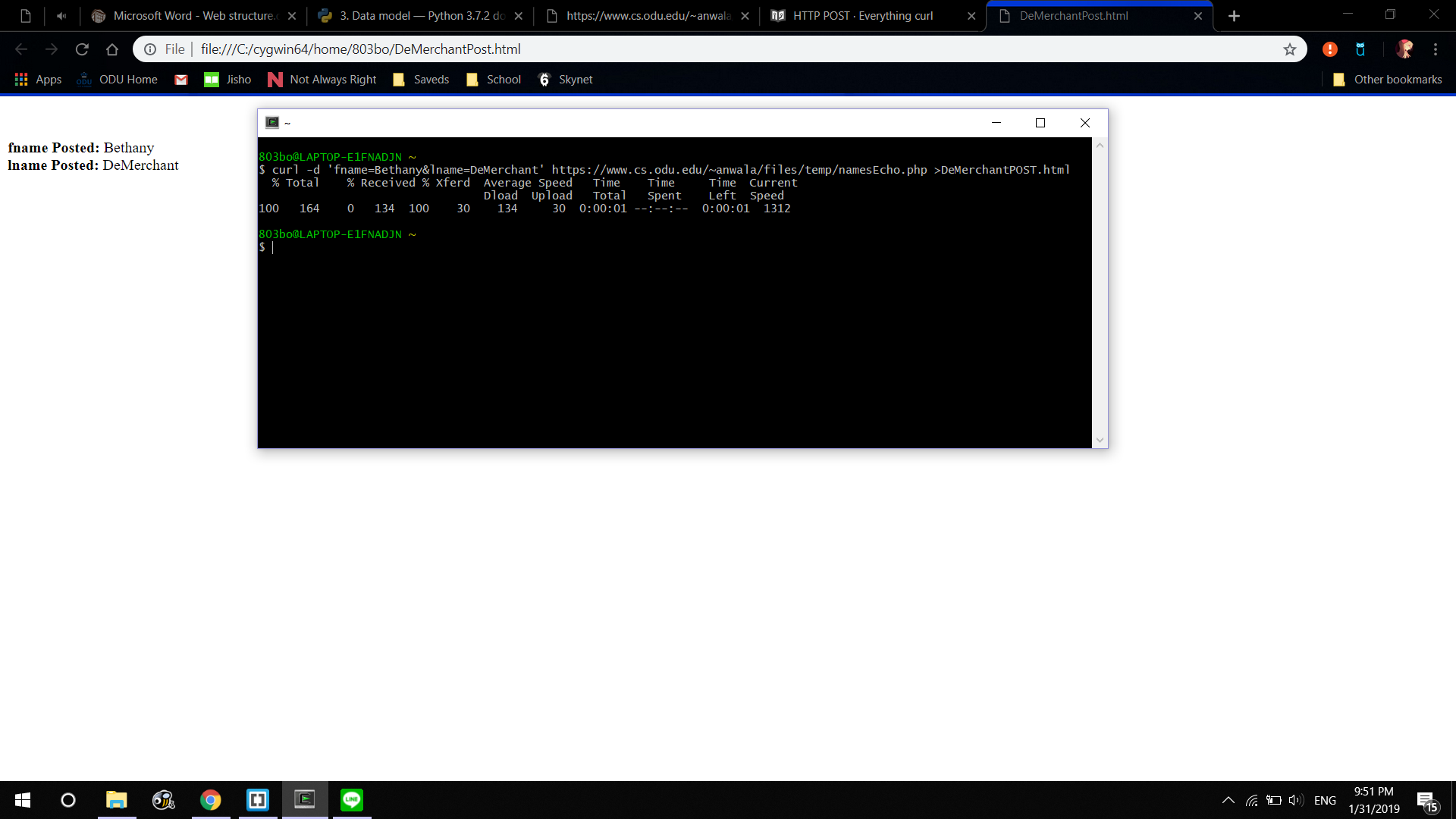


Figure 1: Post Data Using curl

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. (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 2:

The solution was executed in Python using external modules Requests (http://docs.python-requests.org/en/master/) and BeautifulSoup (https://www.crummy.com/software/BeautifulSoup/) as well as built-in modules argparse and urllib.parse.

The argparse module was used to generate command-line interface and ensure proper functioning if a user fails to provide a URI argument (see Figure 2).

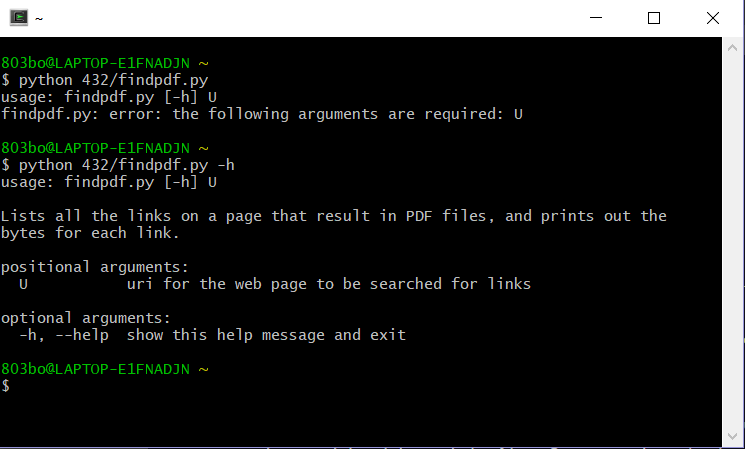


Figure 2: Command-Line Interface Using argparse

The URI provided as an argument is dereferenced using the Requests module, which performs a GET request and creates a response object containing the page contents. Assuming the page contents exist, they are then parsed by BeautifulSoup, which scans for the <a> tag which represents a hyperlink in HTML. For each <a> tag found, the URI address of the link is added to a list of all links in the page. Each address in the complete list is then checked with a HEAD request performed using the Requests module. If the header’s Content-Type header is “application/pdf”, the page’s URI and the value of the Content-Length header are added to a list of links resulting in PDF files. Finally, the finished list of links to PDF files is printed to the console.

The demonstration page (<http://www.cs.odu.edu/~mln/teaching/cs532-s17/test/pdfs.html>) contains twenty links, eleven of which lead to PDF files. Figure 3 shows that the program reported all eleven PDF files and no links to non-PDF pages, and also that it successfully followed redirects.

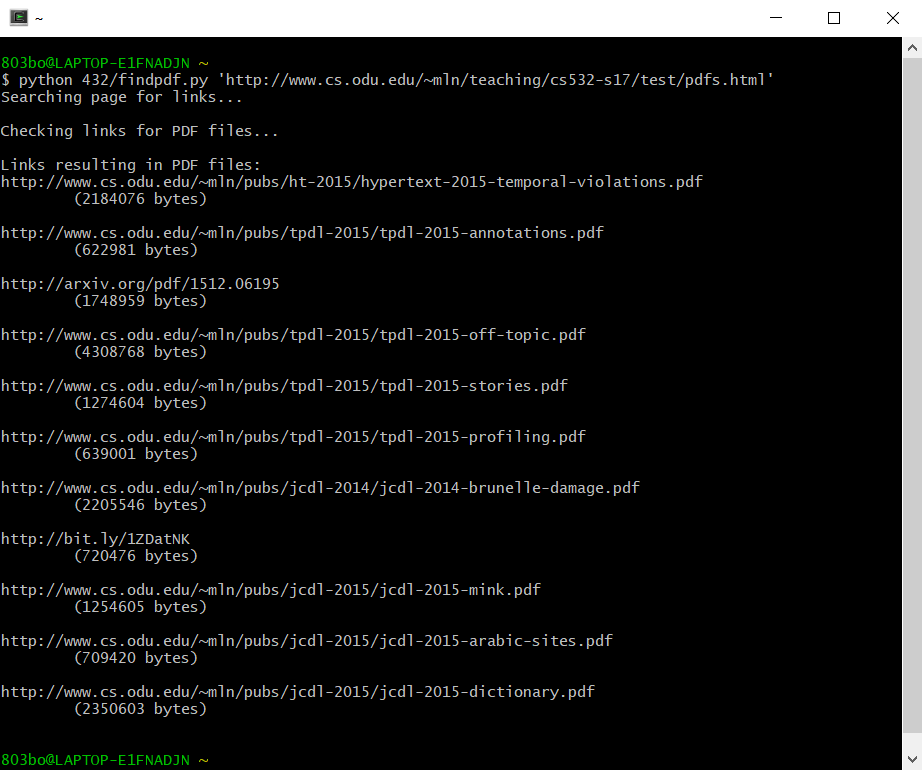


Figure 3: Results from Demonstration Page

The program was also tested on two Wikipedia articles and a test page created specifically for the purpose. The results can be seen in Figure 4.

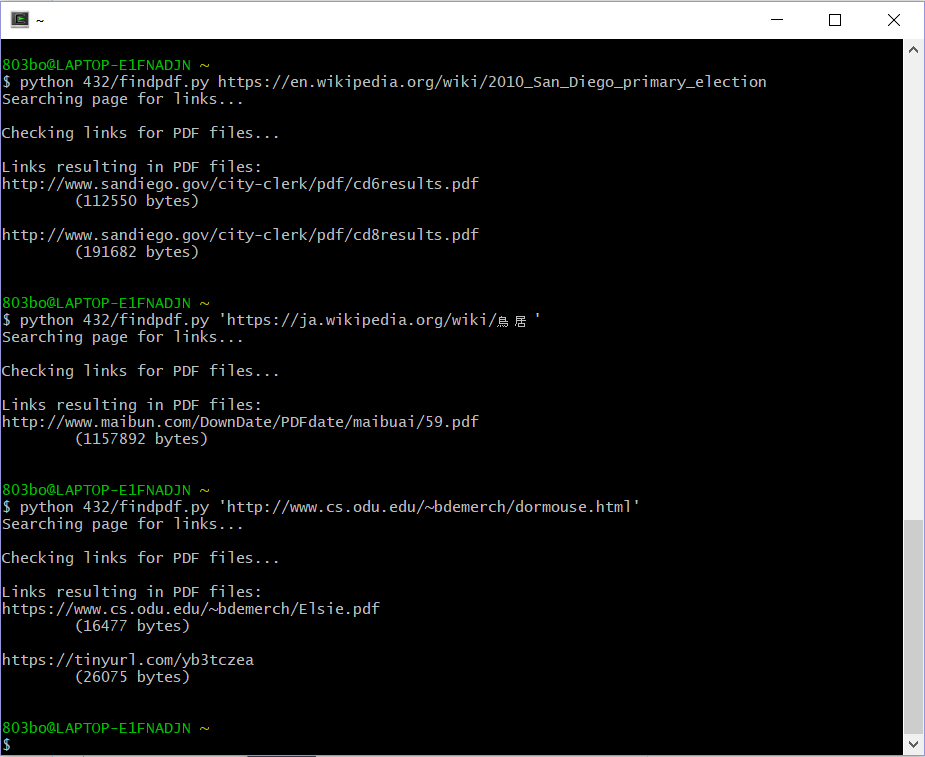


Figure 4: Results of Secondary Testing