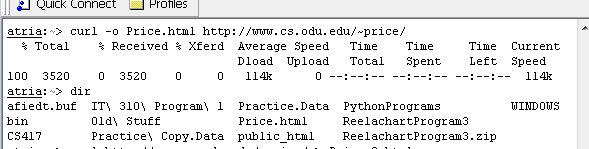
**Question 1**

CURL turned out to be a difficult function to use, at least in terms of POSTing to a form. That difficult seemed to stem from a difficulty in finding website that allowed me to POST form data easily for a result that was worth recording.

However, I was still able to practice using CURL to retrieve the raw HTML of a URI.

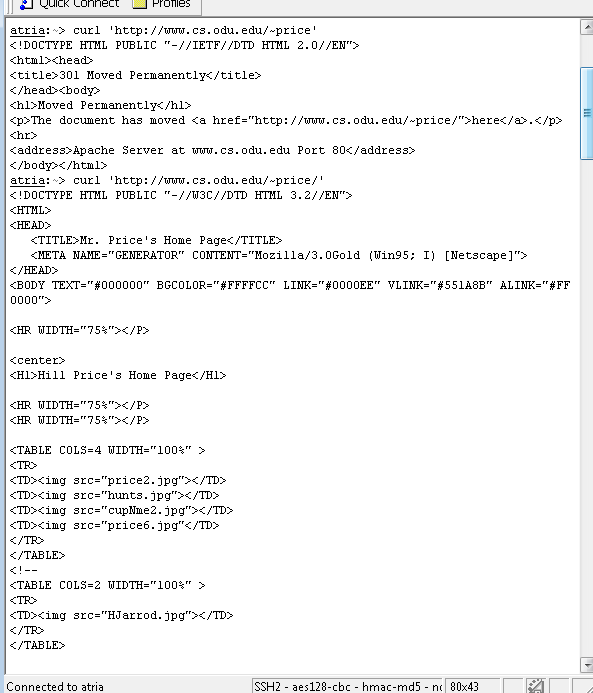
For example “curl ‘http://www.cs.odu.edu/~price/” returns the raw HTML of the provided URI (*Figure 1)*. Then, by using the command “curl –o Price.html <http://www.cs.odu.edu/~price/>” I was able to store the HTML output into a file named “Price.html” (*Figure 2*). This HTML file could be open in Chrome to display the website, without the images (*Figure 3*).

All images are available in the github under the names “Curl Price.PNG”, “Curl-o Price.PNG”, and “Price HTML.PNG” respectively.



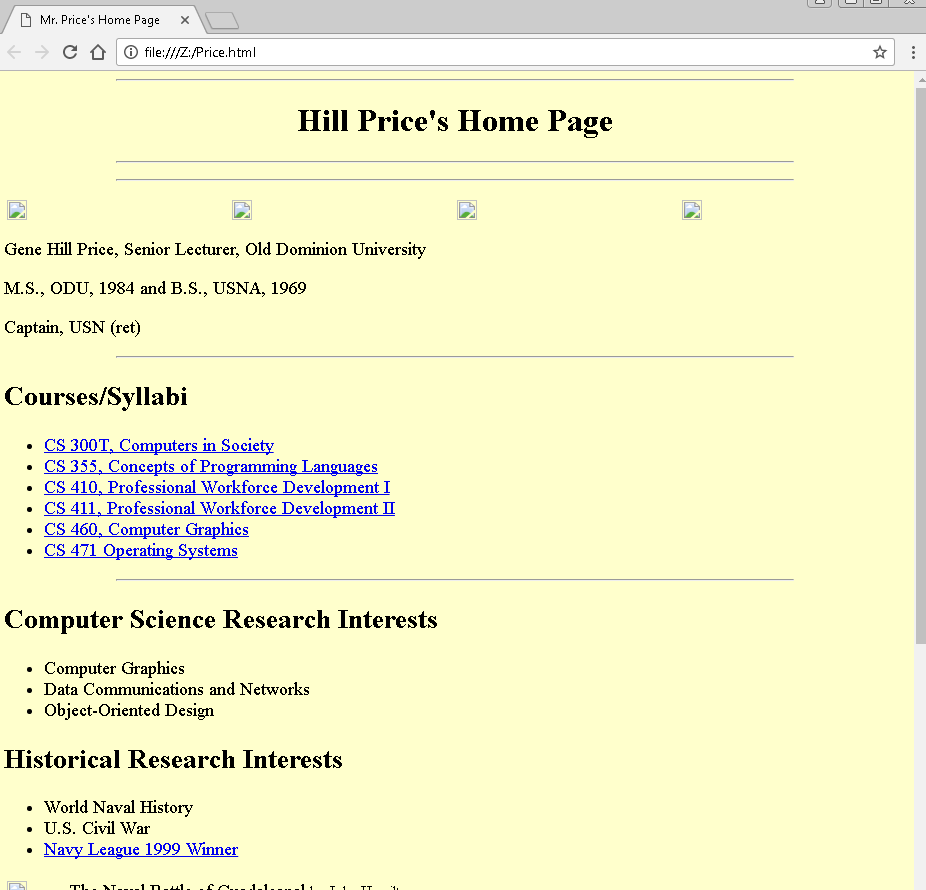
Figure

*Continued below*



Figure

*Continued below*



Figure

*Continued below*

**Question 2**

The Python program associated with this question is in the file “ParsePDFv2.py”.

This program can be from the command line with the syntax “python ParsePDFv2.py [URI]”.

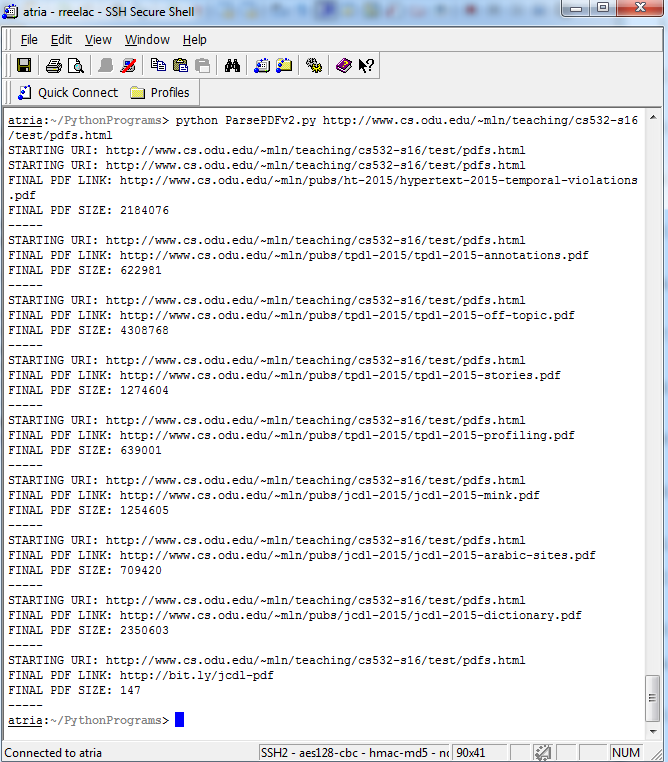
The program starts (line 8) by getting the URI from the command line, which is stored in the command line variable *argv[1]*. This URI string is stored in the variable *url*. Then the program sends a request to the URI and stores the response in *resp* (line 10). Using BeautifulSoup, the content of the URI is parsed and stored in variable *soup* (line 11).

Using a FOR loop, the program finds all available links on the starting URI (line 15). For each link found, the link is stored in variable *current\_link* (line 16). Using a conditional IF statement, the program determines if the URI for *current\_link* ends in “pdf”. While this is not ideal, this is the best I could manage.

If *current\_link* is found to end in the “pdf” file type, the program prints the original starting URI, the URI for the found, and the size (in bytes) of the found pdf. This repeats until all links on the original URI are searched.

Results for the example, required URI are shown below in *Figure 4*. The raw screenshot is also available on the in the github repository as “First URI Screen.PNG”. Results for two other URIs did not pan out as hoped and are not shown.

*Continued below*

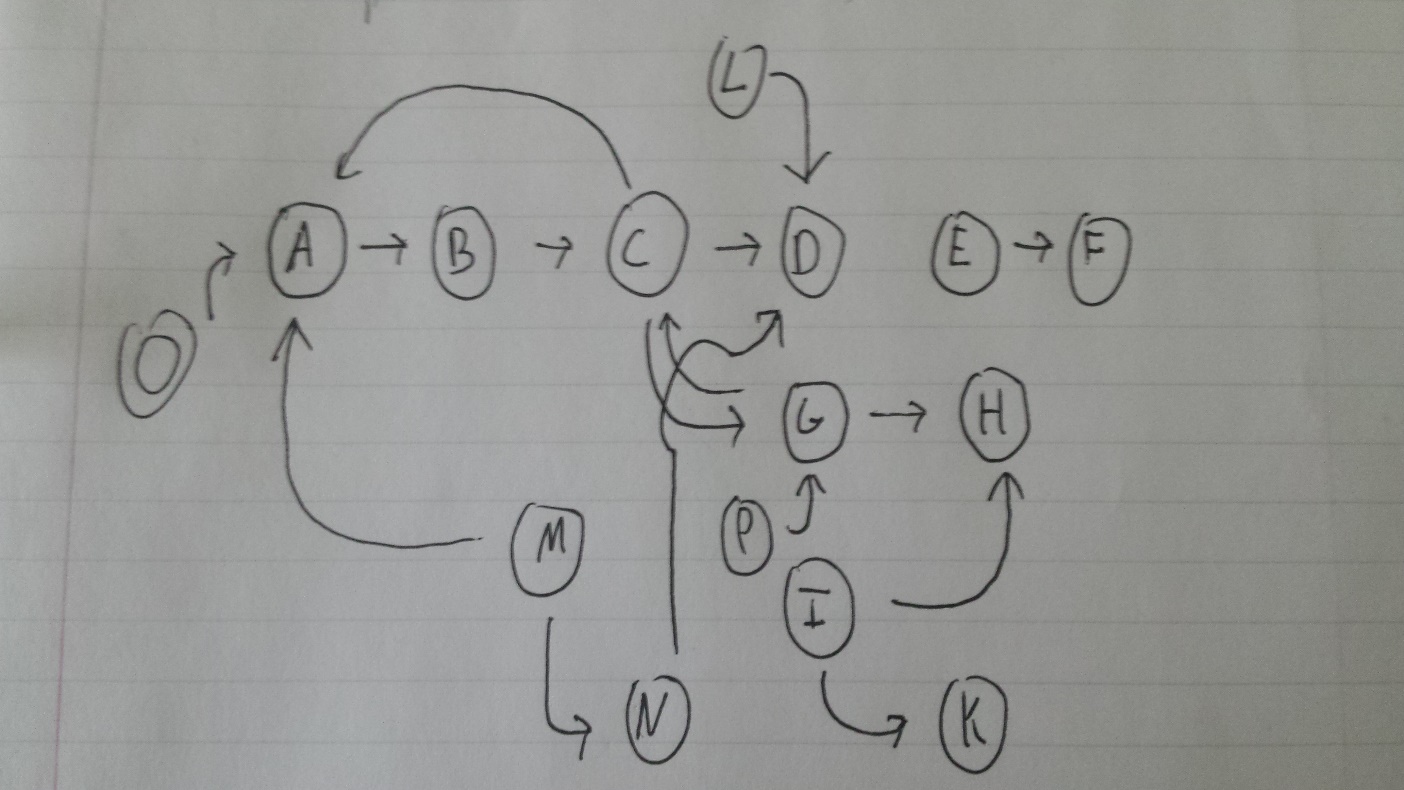
****

Figure

*Continued below*

**Question 3**

A hand-drawn mapping of the given nodes and edges are shown below in *Figure 5*.

****

Figure

IN: *O, M*

IN components are starting points on the graph. They go into the SCC and nowhere else. They may have TENDRILS coming out of them that do not connect to the SCC.

SCC: *A, B, C, G*

SCC components are heavily connected with each other. In other words, it is possible to iterate through these components without leaving into the OUT. And they can only be accessed through the IN.

OUT: *D, H*

OUT Components exit the SCC and do not join back into it or IN.

TENDRILS: *I, K, L*

TENDRILS are components that are not within the SCC. They either come from somewhere else to join the IN, or go off from OUT.

TUBES: *N*

TUBES are connections that go directly from IN (i.e. M) to OUT (i.e. D) without ever joining the SCC.

*Continued below*

DISCONNECTED: *E, F*

DISCONNECTED components are completely disconnected from the rest of the graph. They may connect to each other, but not to the IN, SCC, OUT, TENDRILS, or TUBES.