2017

Assignment Three

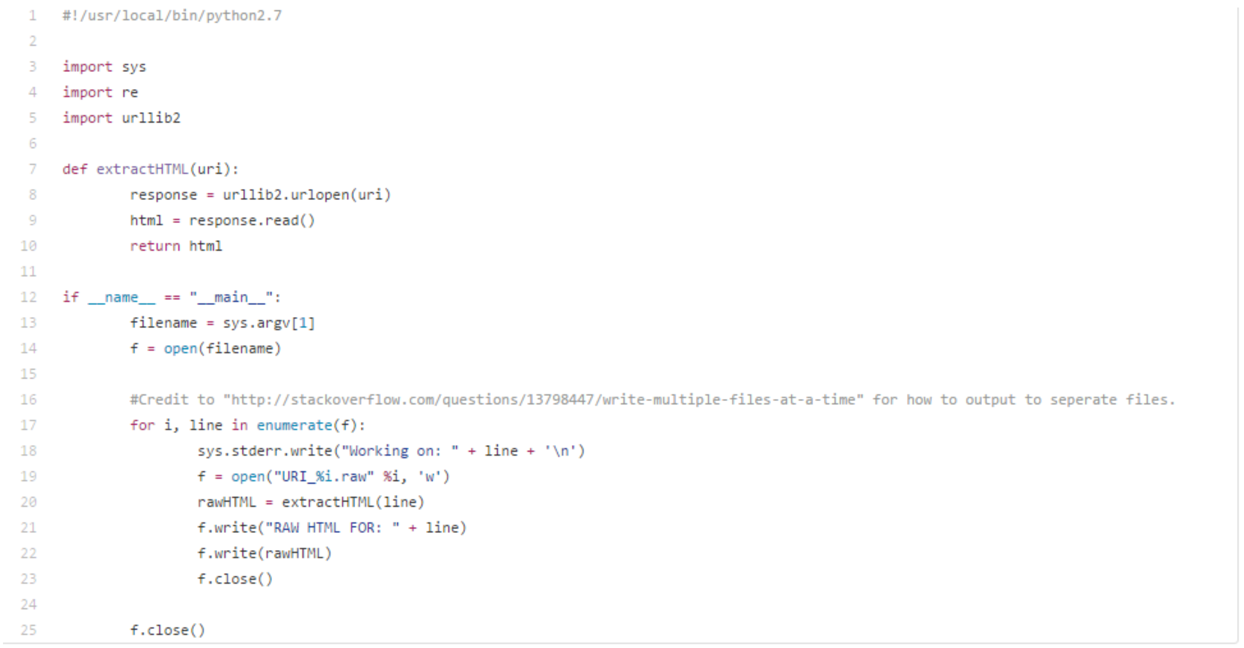
ROSS REELACHART

PROFESSOR MICHAEL L. NELSON

Question 1

Downloading the raw HTML of the 1000 URIs started off simple enough because I started with the list of 1000 URIs from the second assignment (1000Links.txt). So from that starting point, I wanted to write a program that read that text file line-by-line, using the URI on each line as a separate call. Getting the raw HTML was easy enough as I simply used the *urllib2* library to open the URI that was on the line, and then use *response.read()* to extract the HTML. However, getting each read to output to its own file, instead of one big file containing all 1000 URIs, took some research.

Fortunately, I found a solution[1]. Using *enumerate*, I made the program (getRawHTML.py) output a new file (called *.raw) for each line (URI). It also automatically appended a number to each .raw file that corresponded with the URI's position in the original text listing, creating 1000 .raw files named "URI_0.raw" through "URI_999.raw."



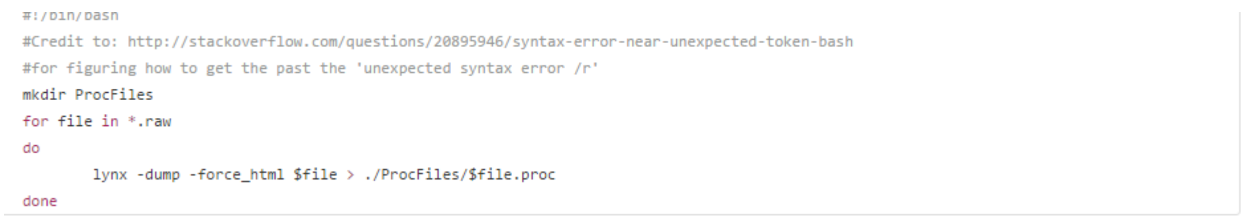
```
#!/usr/local/bin/python2.7

import sys
import re
import urllib2

def extractHTML(uri):
	response = urllib2.urlopen(uri)
	html = response.read()
	return html

if __name__ == "__main__":
	filename = sys.argv[1]
	f = open(filename)

	#Credit to "http://stackoverflow.com/questions/13798447/write-multiple-files-at-a-time" for how to output to seperate files.
	for i, line in enumerate(f):
		sys.stderr.write("Working on: " + line + '\n')
		f = open("URI_%i.raw" %i, 'w')
		rawHTML = extractHTML(line)
		f.write("RAW HTML FOR: " + line)
		f.write(rawHTML)
		f.close()

	f.close()
```

Figure 1 The code for "getRawHTML.py." Note the use of enumerate to create a file for each read URI on the list.

Stripping out the HTML tags, CSS and other extraneous tags in favor of the raw text took more time. I went through several attempts at a python program that attempted to run through each "URI_x.raw "file, strip out the necessary tags, and then either output a new cleaned file or overwrite the original file. The problems I most encountered were finding satisfactory ways to strip out the tags AND do it on a per-file basis.

My solution turned out to be a shell/bash script (procRaw2.sh). This very short script was placed in a directory that held duplicates of the .raw files obtained in the first part of this question. This script then created a new folder called "ProcFiles" in which to place its output. Then, using a FOR a loop, the script went through each file and applied the command *lynx -dump -force_html $file > ./ProcFiles/$file.proc*. This stripped out the HTML tags and then created a new file with the same name, but with .proc appended to the file name (i.e. URI_11.raw creates URI_11.raw.proc).

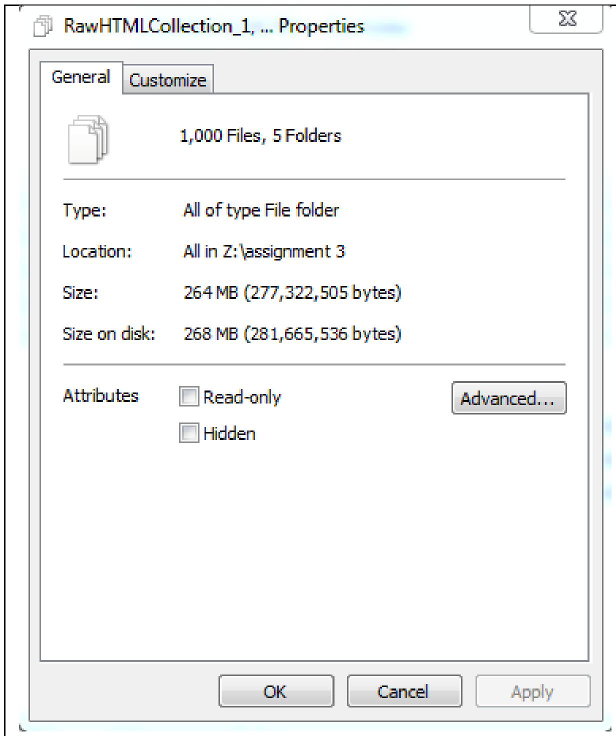
I was nearly stumped by a syntax error that prevented the script from running, but then I found a solution[2] that converted the script (written in Notepad++) into something that was pure Unix-friendly.



```
#!/bin/bash
#Credit to: http://stackoverflow.com/questions/20895946/syntax-error-near-unexpected-token-bash
#for figuring how to get the past the 'unexpected syntax error /r'
mkdir ProcFiles
for file in *.raw
do
        lynx -dump -force_html $file > ./ProcFiles/$file.proc
done
```

Figure 2 The code for "procRaw.sh." Was written in Notepad++, then used a Unix command to convert it into an acceptable form.

Note: These two programs created a total of 2000 files (.raw and .proc). Since github had an upload limit (both in terms of byte size and number of files), they will not be uploaded to my github repository. However, I do have them and will be able to provide them upon request.



RawHTMLCollection_1, ... Properties

1,000 Files, 5 Folders

Type: All of type File folder

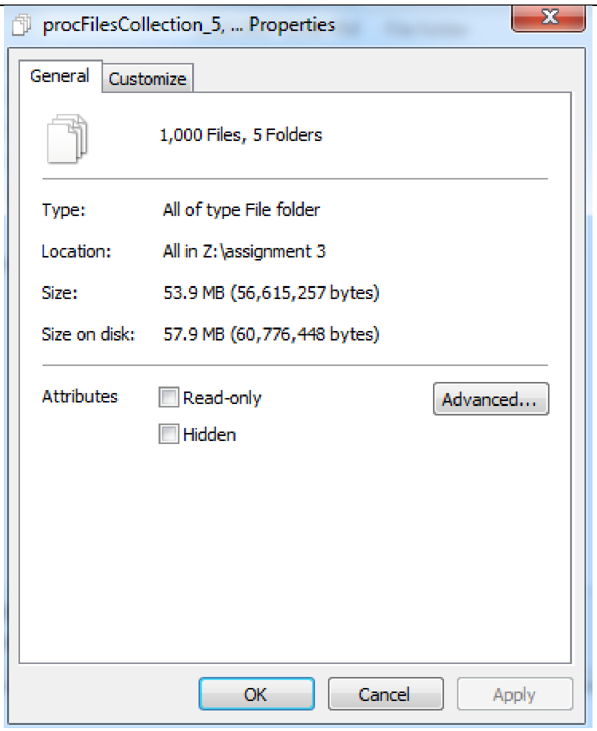
Location: All in Z:\assignment 3

Size: 264 MB (277,322,505 bytes)

Size on disk: 268 MB (281,665,536 bytes)

Attributes: Read-only, Hidden

Figure 3 The file sizes of both the raw HTML files and the processed files. Even after splitting the 1000 files into 5 folders for each, they were still very large.



procFilesCollection_5, ... Properties

1,000 Files, 5 Folders

Type: All of type File folder

Location: All in Z:\assignment 3

Size: 53.9 MB (56,615,257 bytes)

Size on disk: 57.9 MB (60,776,448 bytes)

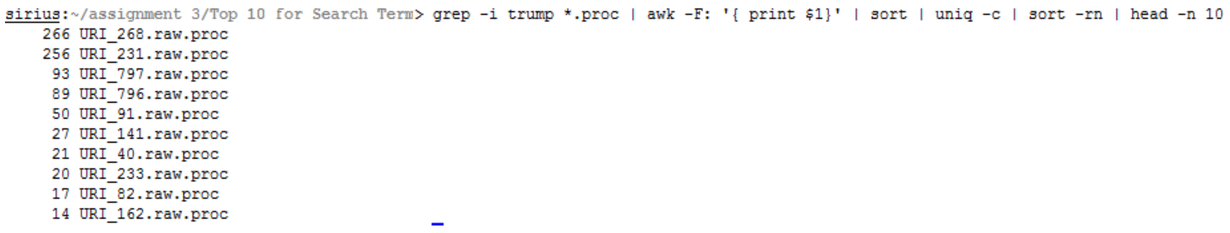
Attributes: Read-only, Hidden

(Continued on next page.)

Question 2

For this part of the assignment, I needed to search through my collected files for a search term. Despite my own reservations about the term, I felt like I might find a good amount of results by searching for the term “trump.” I used the following grep and awk commands to find the term in each file, count the number of occurrences per document, and then rank them in descending order:

*"grep -i trump *.proc | awk -F: '{ print $1}' | sort | uniq -c | sort -rn | head -n 10"*



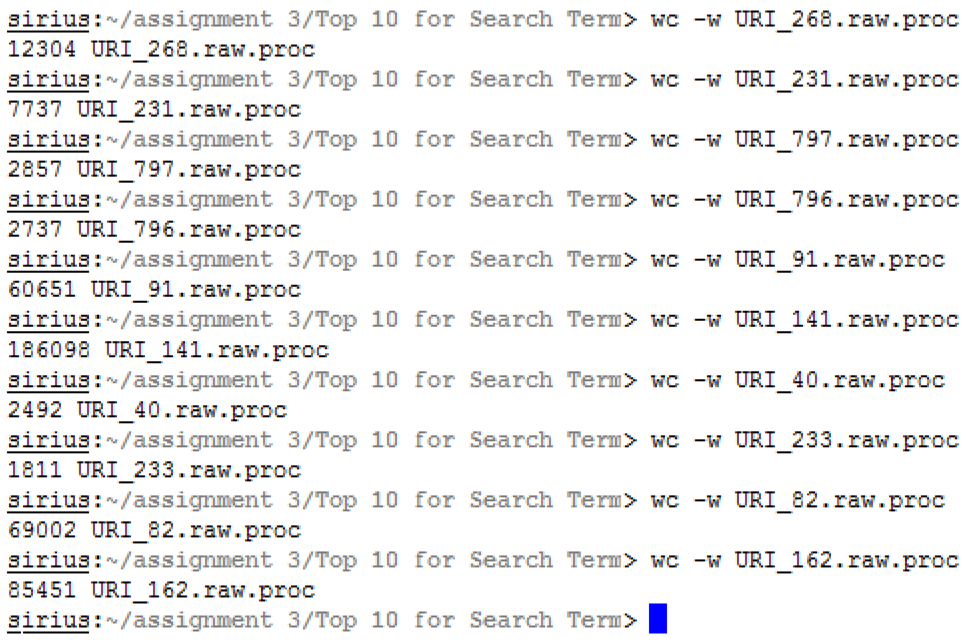
```
sirius:~/assignment 3/Top 10 for Search Term> grep -i trump *.proc | awk -F: '{ print $1}' | sort | uniq -c | sort -rn | head -n 10
    266 URI_268.raw.proc
    256 URI_231.raw.proc
     93 URI_797.raw.proc
     89 URI_796.raw.proc
     50 URI_91.raw.proc
     27 URI_141.raw.proc
     21 URI_40.raw.proc
     20 URI_233.raw.proc
     17 URI_82.raw.proc
     14 URI_162.raw.proc
```

Figure 4 Output of grep command. Only shows 10 of the 11 I found.

I honestly thought I’d find more occurrences, but I don’t think I gathered from the right sources to actually get the term frequently. But it did appear enough to be satisfactory for my purposes. I ended up with a top 11 list because the final two had the same number of occurrences (14).

I then needed the total word count for each of those files. This was simple enough to accomplish with just the 11 files I found using the following Unix word count command for each file:

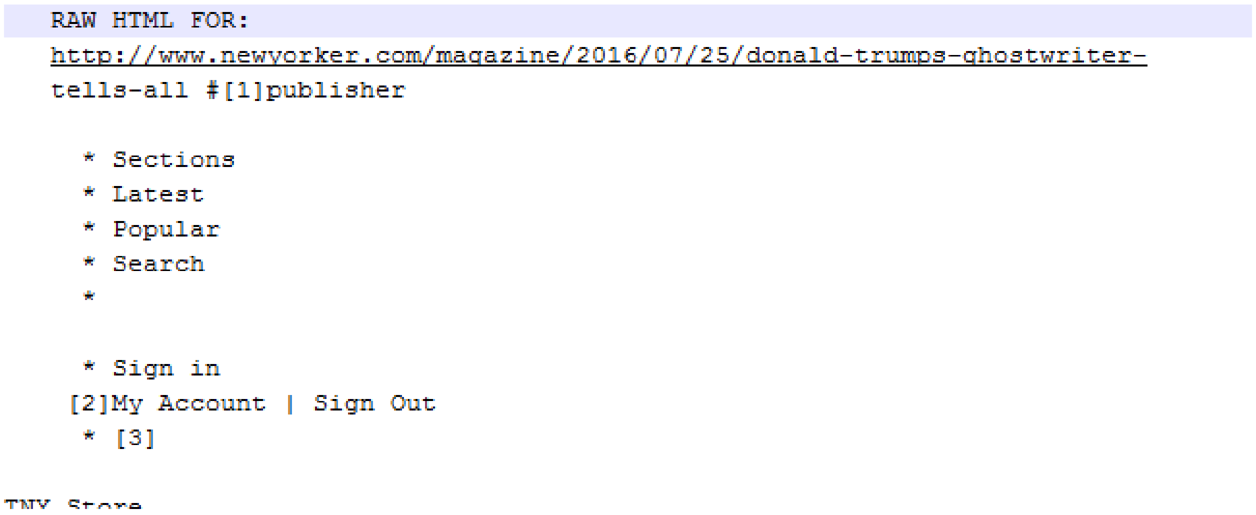
“wc –w <file>”



```
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_268.raw.proc
12304 URI_268.raw.proc
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_231.raw.proc
7737 URI_231.raw.proc
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_797.raw.proc
2857 URI_797.raw.proc
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_796.raw.proc
2737 URI_796.raw.proc
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_91.raw.proc
60651 URI_91.raw.proc
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_141.raw.proc
186098 URI_141.raw.proc
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_40.raw.proc
2492 URI_40.raw.proc
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_233.raw.proc
1811 URI_233.raw.proc
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_82.raw.proc
69002 URI_82.raw.proc
sirius:~/assignment 3/Top 10 for Search Term> wc -w URI_162.raw.proc
85451 URI_162.raw.proc
sirius:~/assignment 3/Top 10 for Search Term>
```

Figure 5 The results of running the word count command on each file. The word count is the number beside each filename.

Getting the URI for each file was made easier due to a step I took in the previous assignment, which inserted a line indicating the URI of the associated file at the top of the file.



```
RAW HTML FOR:
http://www.newyorker.com/magazine/2016/07/25/donald-trumps-ghostwriter-
tells-all #[1]publisher

  * Sections
  * Latest
  * Popular
  * Search
  *

  * Sign in
 [2]My Account | Sign Out
  * [3]
```

Figure 6 An example of the URI listing in each file. The line "RAW HTML FOR: <URI>" was added when I initially collected the URIs in assignment 2.

The normalized TF for a search term is calculated by dividing the number of search term occurrences on a page by the total word count of the page. Then I needed to calculate the inverse document frequency for the term. According to "WorldWideWebSize.com", as of Feb. 20, the indexed web contained about 4.5 billion (4,500,000,000) pages. According to Google, it has roughly 1.07 (1,070,000,000) billion results for the term "trump." So the logarithmic calculation is as follows:

$\text{Log}_2(4{,}5000{,}000{,}000/1{,}070{,}000{,}000) = \text{Log}_2(4.20561) = 2.07231$ IDF

Then finding the TFIDF is found by multiplying the normalized TF for each page with the number found for the IDF. The results are arranged in the following table, ranked by TFIDF.

(Table on next page.)

TFIDF	TF	IDF	URI
0.06857	0.03309	2.07231	http://www.newyorker.com/magazine/2016/07/25/donald-trumps-ghostwriter-tells-all
0.06745	0.03255	2.07231	https://twitter.com/realDonaldTrump/status/824078417213747200
0.06739	0.03252	2.07231	https://twitter.com/realDonaldTrump/status/815185071317676033
0.04480	0.02162	2.07231	http://www.vanityfair.com/magazine/2015/07/donald-ivana-trump-divorce-prenup-marie-brenner
0.02288	0.01104	2.07231	http://www.opednews.com/Diary/Size-matters-penis-study-by-Ron-Mexico-111121-872.html
0.01747	0.00843	2.07231	http://finance.yahoo.com/news/cisco-announces-agreement-acquire-sourcefire-120000871.html
0.00169	0.00082	2.07231	http://redlettermedia.com/best-of-the-worst-wheel-of-the-worst-10/
0.00068	0.00033	2.07231	http://redlettermedia.com/half-in-the-bag-10-cloverfield-land-and-me-him-her/
0.00052	0.00025	2.07231	http://redlettermedia.com/best-of-the-worst-plinketto-1/
0.00033	0.00016	2.07231	http://redlettermedia.com/pre-rec-rocket-league/
0.00031	0.00015	2.07231	http://redlettermedia.com/half-in-the-bag-v-batman-v-superman-dawn-of-justice/

Question 3

For the final question, I wanted to find the page rankings of the 11 URIs that contained my search term. From the suggested free PR estimators, I chose to use "http://pr.eyedomain.com/". The results are listed below, with their PR normalized to an "out-of-ten" ranking and also their TFIDF number for comparison.

Page Rank	TFIDF	URI
9/10	0.01747	http://finance.yahoo.com/news/cisco-announces-agreement-acquire-sourcefire-120000871.html
8/10	0.06857	http://www.newyorker.com/magazine/2016/07/25/donald-trumps-ghostwriter-tells-all
8/10	0.04480	http://www.vanityfair.com/magazine/2015/07/donald-ivana-trump-divorce-prenup-marie-brenner
6/10	0.02288	http://www.opednews.com/Diary/Size-matters-penis-study-by-Ron-Mexico-111121-872.html
5/10	0.00169	http://redlettermedia.com/best-of-the-worst-wheel-of-the-worst-10/
5/10	0.00068	http://redlettermedia.com/half-in-the-bag-10-cloverfield-land-and-me-him-her/
5/10	0.00052	http://redlettermedia.com/best-of-the-worst-plinketto-1/
5/10	0.00033	http://redlettermedia.com/pre-rec-rocket-league/
5/10	0.00031	http://redlettermedia.com/half-in-the-bag-v-batman-v-superman-dawn-of-justice/
0/10	0.06745	https://twitter.com/realDonaldTrump/status/824078417213747200
0/10	0.06739	https://twitter.com/realDonaldTrump/status/815185071317676033

So starting from the top, the New Yorker article has the highest TFIDF score, which is unsurprising considering that the article is just about Trump's ghostwriter, so the search term 'trump' is bound to appear frequently when compared to the page's word count and the IDF score for the term. But I found it surprising that the New Yorker domain comes in second in Page Rank to Yahoo Finance. It's only a one step difference but I thought it was neat.

Still considering TFIDF, the next rankings made sense considering it was from his twitter apparently, and the Vanity Fair article is about him specifically again. I don't know what's going on with that penis article, though. Then Red Letter Media comprised the latter half of the TFIDF rankings, and it seemed to come from the comment sections of their website devolving into the internet's usual political chatter.

As for Page Rankings, it seems pretty clear cut between journalistic or normal media sites at the top, and then RLM's personal video site and then Twitter. The New Yorker, Yahoo Finance and Vanity Fair sat high on Page Rankings because of their media presence and history, with Op-Ed News falling below due to its non-mainstream nature. RLM's site sat right in the middle because of its niche entertainment value, but it obviously get enough traffic to register. Twitter sat right at the bottom, and I suppose that's because the actual Twitter domain isn't used nearly as much as the feeds of individual users.

References

1.) http://stackoverflow.com/questions/13798447/write-multiple-files-at-a-time
2.) http://stackoverflow.com/questions/20895946/syntax-error-near-unexpected-token-bash