CS595 Intro to Web Science, Assignment #1

Valentina Neblitt-Jones

September 12, 2013

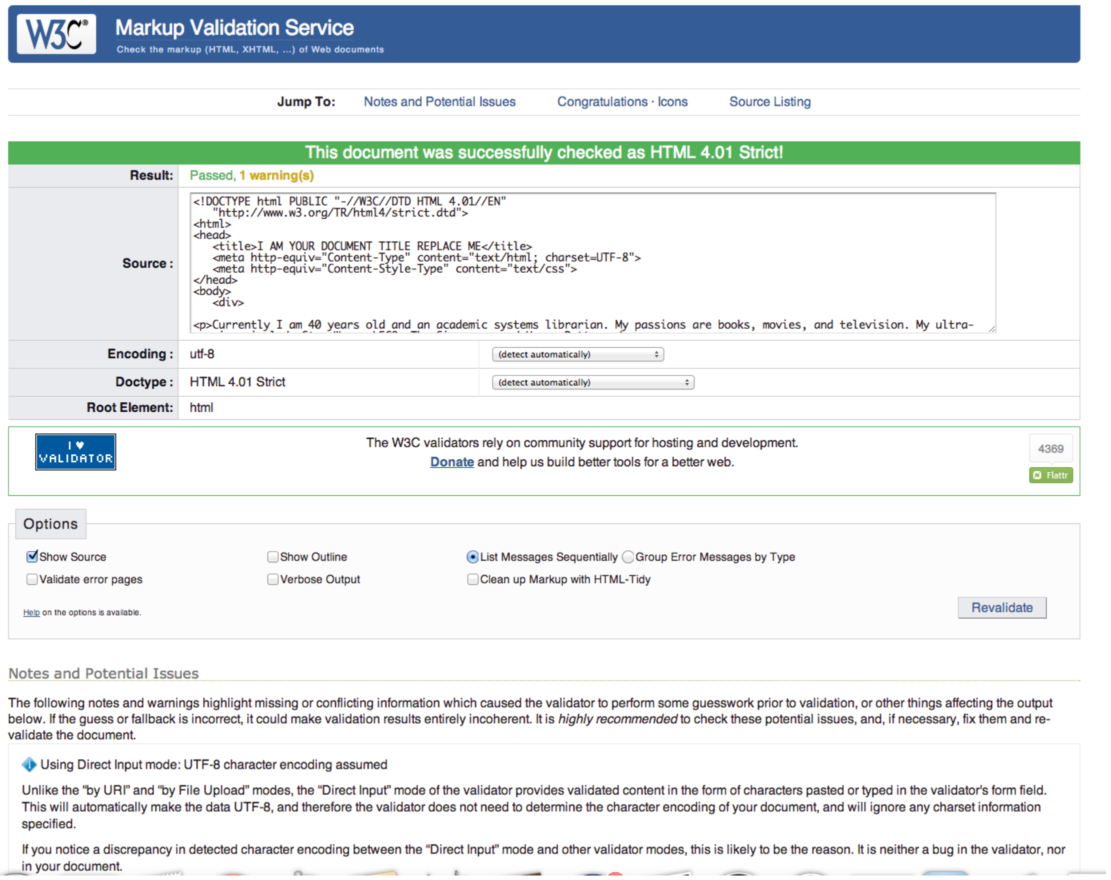
1 cURL Exercise

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).

Finding a web form that did not require me revealing my credentials and actually used the POST method was quite difficult. I used W3C Markup Validator Service `http://validator.w3.org/#validate_by_input`. It was being extra fussy about the special characters (–urlencode was not solving the problem) so so I had to substitute the appropriate URL encoding using `http://www.w3schools.com/tags/ref_urlencode.asp`. My cURL statement follows.

```
curl --data "fragment=%3Cp%3ECurrently I am 40 years old and an academic
    systems librarian. My passions are books, movies, and television. My ultra
    -passions include Star Wars, LEGO, The Simpsons, and Harry Potter.%3C%2Fp
    %3E&prefill=1&doctype=Inline&fbd=1&prefill_doctype=html401&group=0&ss=1&st
    =1&outline=1&No200=1&verbose=1" --url http://validator.w3.org/check -o
    debug.html
```

The four screenshots below illustrate the difference in how the page was rendered when using the browser versus cURL to fill out the form. Figures 1 & 2 show the browser version and Figures 3 & 4 show the cURL version. Although the stylesheet and images are missing from the cURL version, the same information is present.


Markup Validation Service
Check the markup (HTML, XHTML, ...) of Web documents
Jump To: Notes and Potential Issues Congratulations · Icons Source Listing

This document was successfully checked as HTML 4.01 Strict!

Result:	Passed, 1 warning(s)	
Encoding :	utf-8	(detect automatically)
Doctype :	HTML 4.01 Strict	(detect automatically)
Root Element:	html	

Source :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
   <title>I AM YOUR DOCUMENT TITLE REPLACE ME</title>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <meta http-equiv="Content-Style-Type" content="text/css">
</head>
<body>
   <div>

<p>Currently I am 40 years old and an academic systems librarian. My passions are books, movies, and television. My ultra-
```

The W3C validators rely on community support for hosting and development.
Donate and help us build better tools for a better web.

Options

[✓ Show Source] Show Outline [● List Messages Sequentially] Group Error Messages by Type
Validate error pages Verbose Output Clean up Markup with HTML-Tidy

Help on the options is available.

Notes and Potential Issues

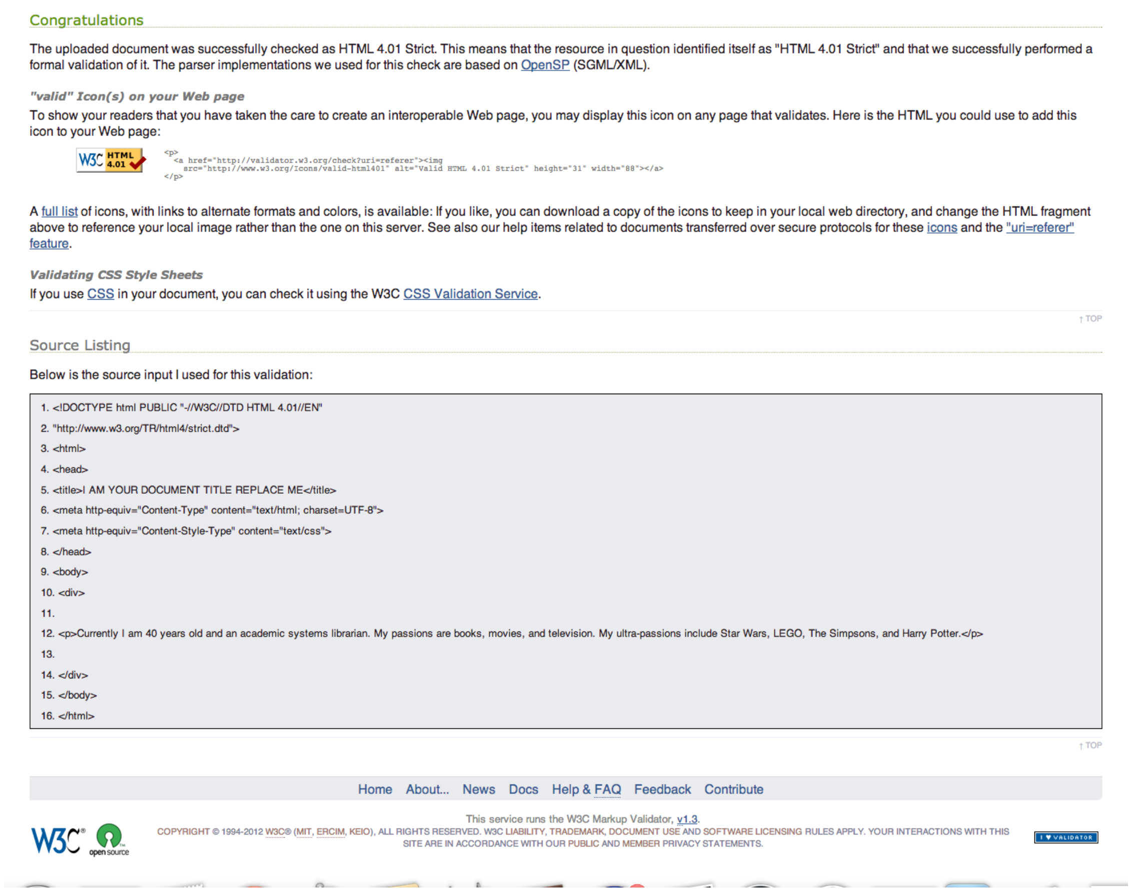
The following notes and warnings highlight missing or conflicting information which caused the validator to perform some guesswork prior to validation, or other things affecting the output below. If the guess or fallback is incorrect, it could make validation results entirely incoherent. It is *highly recommended* to check these potential issues, and, if necessary, fix them and re-validate the document.

Using Direct Input mode: UTF-8 character encoding assumed

Unlike the "by URI" and "by File Upload" modes, the "Direct Input" mode of the validator provides validated content in the form of characters pasted or typed in the validator's form field. This will automatically make the data UTF-8, and therefore the validator does not need to determine the character encoding of your document, and will ignore any charset information specified.

If you notice a discrepancy in detected character encoding between the "Direct Input" mode and other validator modes, this is likely to be the reason. It is neither a bug in the validator, nor in your document.

Figure 1: This is the top half of the screen representing output when the browser was used to fill out the form



Congratulations

The uploaded document was successfully checked as HTML 4.01 Strict. This means that the resource in question identified itself as "HTML 4.01 Strict" and that we successfully performed a formal validation of it. The parser implementations we used for this check are based on OpenSP (SGML/XML).

"valid" Icon(s) on your Web page

To show your readers that you have taken the care to create an interoperable Web page, you may display this icon on any page that validates. Here is the HTML you could use to add this icon to your Web page:

```
<p>
  <a href="http://validator.w3.org/check?uri=referer"><img
    src="http://www.w3.org/Icons/valid-html401" alt="Valid HTML 4.01 Strict" height="31" width="88"></a>
</p>
```

A full list of icons, with links to alternate formats and colors, is available: If you like, you can download a copy of the icons to keep in your local web directory, and change the HTML fragment above to reference your local image rather than the one on this server. See also our help items related to documents transferred over secure protocols for these icons and the "uri=referer" feature.

Validating CSS Style Sheets

If you use CSS in your document, you can check it using the W3C CSS Validation Service.

↑ TOP

Source Listing

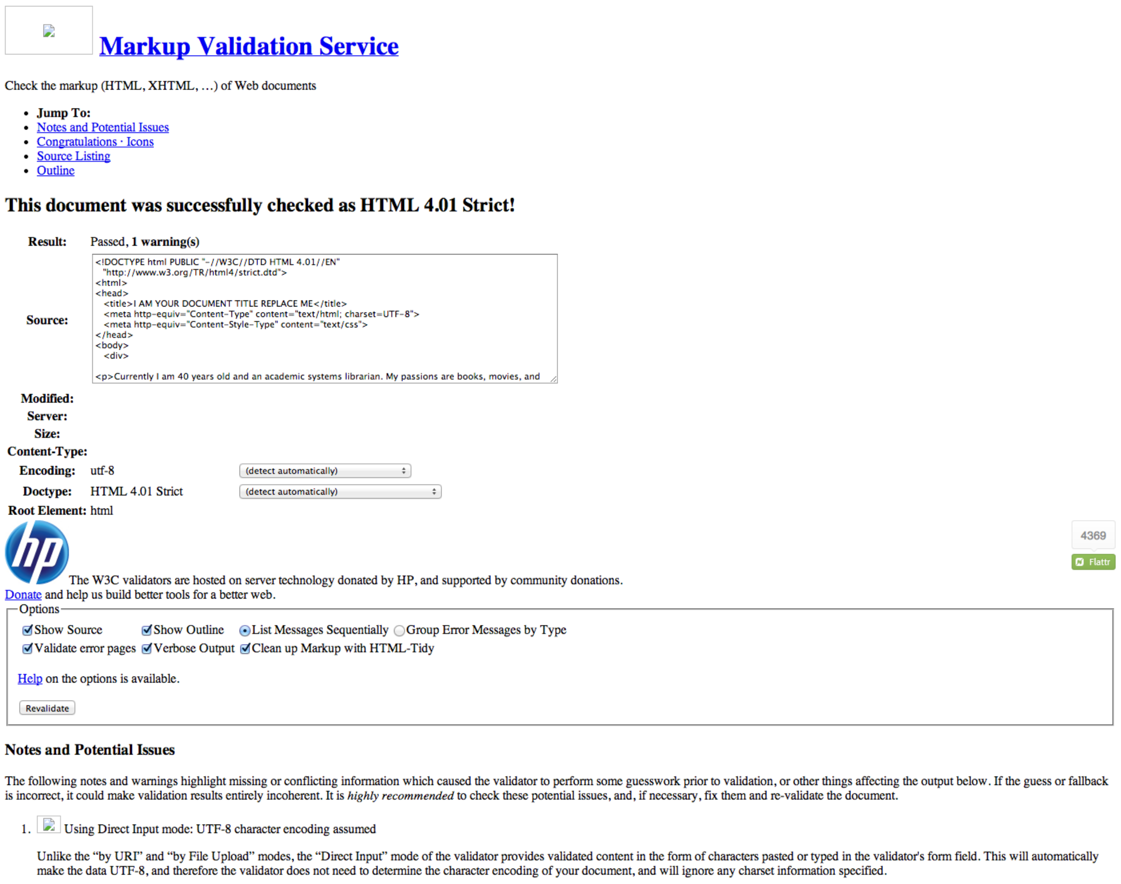
Below is the source input I used for this validation:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>I AM YOUR DOCUMENT TITLE REPLACE ME</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="Content-Style-Type" content="text/css">
</head>
<body>
<div>

<p>Currently I am 40 years old and an academic systems librarian. My passions are books, movies, and television. My ultra-passions include Star Wars, LEGO, The Simpsons, and Harry Potter.</p>

</div>
</body>
</html>
```

↑ TOP

This service runs the W3C Markup Validator, v1.3.

Figure 2: This is the bottom half of the screen representing output when the browser was used to fill out the form



Markup Validation Service

Check the markup (HTML, XHTML, …) of Web documents

- **Jump To:**
- Notes and Potential Issues
- Congratulations · Icons
- Source Listing
- Outline

This document was successfully checked as HTML 4.01 Strict!

Result: Passed, **1 warning(s)**

Source:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>I AM YOUR DOCUMENT TITLE REPLACE ME</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <meta http-equiv="Content-Style-Type" content="text/css">
</head>
<body>
  <div>

<p>Currently I am 40 years old and an academic systems librarian. My passions are books, movies, and
```

Modified:
Server:
Size:
Content-Type:
Encoding: utf-8 (detect automatically)
Doctype: HTML 4.01 Strict (detect automatically)
Root Element: html

The W3C validators are hosted on server technology donated by HP, and supported by community donations.
Donate and help us build better tools for a better web.

Options
[✓] Show Source [✓] Show Outline [● List Messages Sequentially] ○ Group Error Messages by Type
[✓] Validate error pages [✓] Verbose Output [✓] Clean up Markup with HTML-Tidy

Help on the options is available.

Revalidate

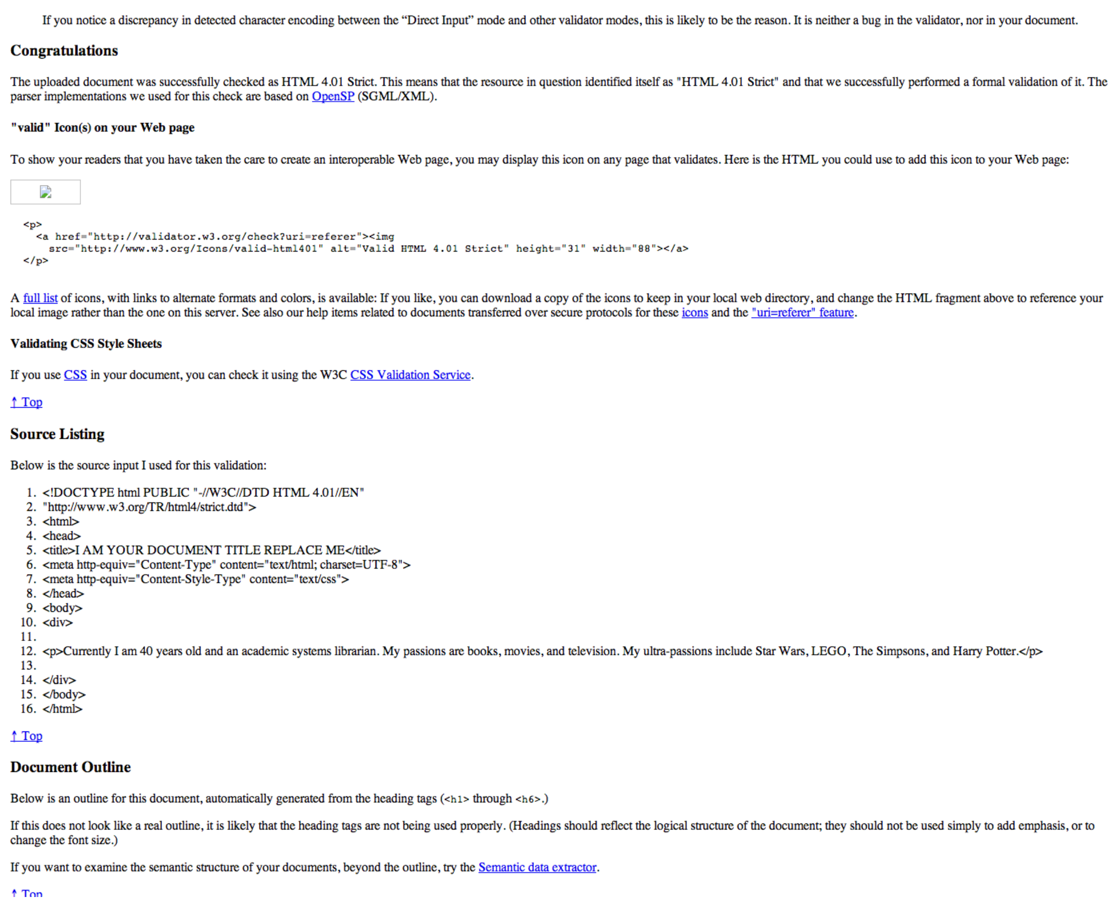
Notes and Potential Issues

The following notes and warnings highlight missing or conflicting information which caused the validator to perform some guesswork prior to validation, or other things affecting the output below. If the guess or fallback is incorrect, it could make validation results entirely incoherent. It is *highly recommended* to check these potential issues, and, if necessary, fix them and re-validate the document.

1. Using Direct Input mode: UTF-8 character encoding assumed

   Unlike the "by URI" and "by File Upload" modes, the "Direct Input" mode of the validator provides validated content in the form of characters pasted or typed in the validator's form field. This will automatically make the data UTF-8, and therefore the validator does not need to determine the character encoding of your document, and will ignore any charset information specified.

Figure 3: This is the top half of the screen representing output when cURL was used to fill out the form



If you notice a discrepancy in detected character encoding between the “Direct Input” mode and other validator modes, this is likely to be the reason. It is neither a bug in the validator, nor in your document.

Congratulations

The uploaded document was successfully checked as HTML 4.01 Strict. This means that the resource in question identified itself as "HTML 4.01 Strict" and that we successfully performed a formal validation of it. The parser implementations we used for this check are based on OpenSP (SGML/XML).

"valid" Icon(s) on your Web page

To show your readers that you have taken the care to create an interoperable Web page, you may display this icon on any page that validates. Here is the HTML you could use to add this icon to your Web page:

```
<p>
  <a href="http://validator.w3.org/check?uri=referer"><img
    src="http://www.w3.org/Icons/valid-html401" alt="Valid HTML 4.01 Strict" height="31" width="88"></a>
</p>
```

A full list of icons, with links to alternate formats and colors, is available: If you like, you can download a copy of the icons to keep in your local web directory, and change the HTML fragment above to reference your local image rather than the one on this server. See also our help items related to documents transferred over secure protocols for these icons and the "uri=referer" feature.

Validating CSS Style Sheets

If you use CSS in your document, you can check it using the W3C CSS Validation Service.

↑ Top

Source Listing

Below is the source input I used for this validation:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>I AM YOUR DOCUMENT TITLE REPLACE ME</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="Content-Style-Type" content="text/css">
</head>
<body>
<div>

<p>Currently I am 40 years old and an academic systems librarian. My passions are books, movies, and television. My ultra-passions include Star Wars, LEGO, The Simpsons, and Harry Potter.</p>

</div>
</body>
</html>
```

↑ Top

Document Outline

Below is an outline for this document, automatically generated from the heading tags (`<h1>` through `<h6>`.)

If this does not look like a real outline, it is likely that the heading tags are not being used properly. (Headings should reflect the logical structure of the document; they should not be used simply to add emphasis, or to change the font size.)

If you want to examine the semantic structure of your documents, beyond the outline, try the Semantic data extractor.

↑ Top

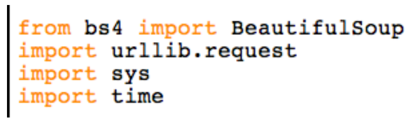
Figure 4: This is the bottom half of the screen representing output when cURL was used to fill out the form

2 Python Exercise

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://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. and (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).**

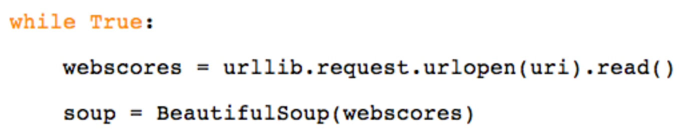
The Code

I used Beautiful Soup for this exercise since it was highly advised by classmates. I also needed to use specifically urllib.request since I was using Python 3.3.2. The sys library was used for support accepting arguments from the command line and time was used to support the "sleep" requirement. See Figure 5 for libraries. I used a curl command to capture a live page and use it for testing, but when the code was ready I changed from using the static file to using the live page (Fig. 6). Using "Inspect Element" on Chrome or Firebug on Firefox, I reviewed the page source for the web page and identified tags and attributes containing the pertinent information. Mod-Content (Fig. 7) was the smallest element that contained all the relevant information. Within Mod-Content, Team Visitor and Score classes and Team Home and Score classes held the team name and score (Figs. 8 and 9). I had to run through the ¡li¿ tags to find the last score and that method allowed for finding the score even if the game was not over. Once all the information was identified and could be referenced, I was able to create the print statement to output the team names and scores and the statement to time out after the specified seconds in the second argument (Fig. 10). The while statement in Figure 6 was used to make the program continue to look for and output the current score.



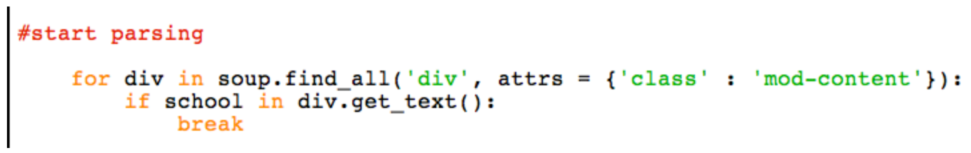
```
from bs4 import BeautifulSoup
import urllib.request
import sys
import time
```

Figure 5: Libraries Used for Exercise



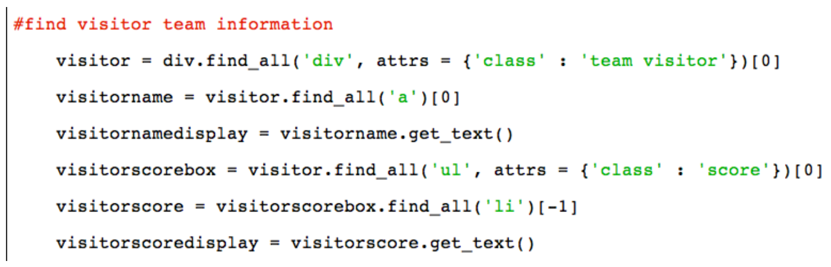
```
while True:

    webscores = urllib.request.urlopen(uri).read()

    soup = BeautifulSoup(webscores)
```

Figure 6: Fetching the web page



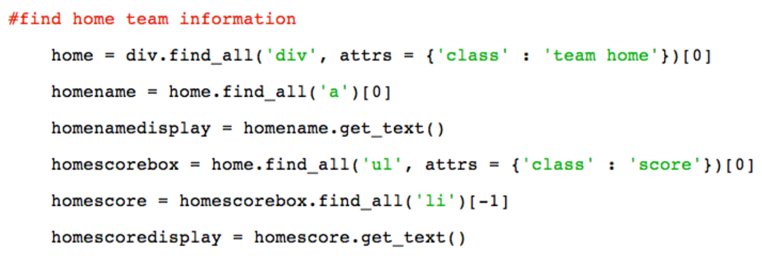
```
#start parsing

    for div in soup.find_all('div', attrs = {'class' : 'mod-content'}):
        if school in div.get_text():
            break
```

Figure 7: Libraries Used for Exercise



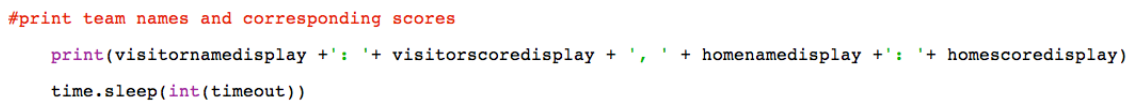
```
#find visitor team information
    visitor = div.find_all('div', attrs = {'class' : 'team visitor'})[0]
    visitorname = visitor.find_all('a')[0]
    visitornamedisplay = visitorname.get_text()
    visitorscorebox = visitor.find_all('ul', attrs = {'class' : 'score'})[0]
    visitorscore = visitorscorebox.find_all('li')[-1]
    visitorscoredisplay = visitorscore.get_text()
```

Figure 8: Code segment for Collecting Visitor Team Information



```
#find home team information
    home = div.find_all('div', attrs = {'class' : 'team home'})[0]
    homename = home.find_all('a')[0]
    homenamedisplay = homename.get_text()
    homescorebox = home.find_all('ul', attrs = {'class' : 'score'})[0]
    homescore = homescorebox.find_all('li')[-1]
    homescoredisplay = homescore.get_text()
```

Figure 9: Code segment for for Collecting Home Team Information



```
#print team names and corresponding scores
    print(visitornamedisplay +': '+ visitorscoredisplay + ', ' + homenamedisplay +': '+ homescoredisplay)
    time.sleep(int(timeout))
```

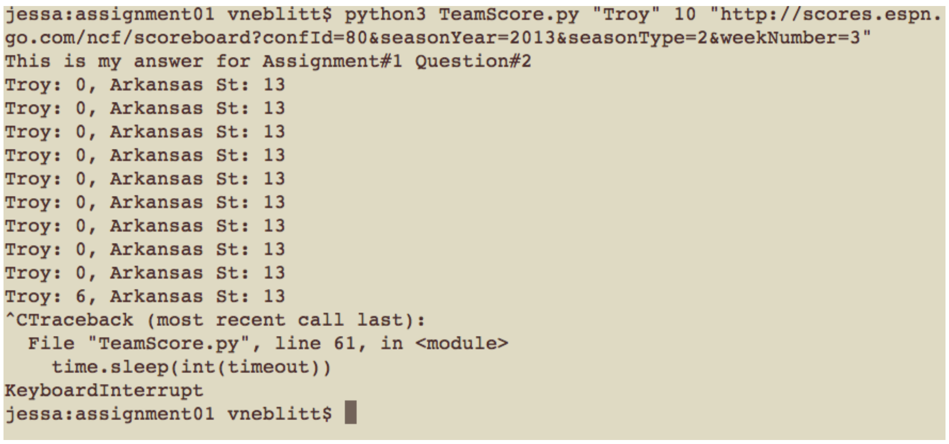
Figure 10: Code segment for Final Output and Time Out

The Execution

Figure 11 shows the execution of the code. The arguments were:

1. school = Troy
2. timeout = 10
3. uri = `http://scores.espn.go.com/ncf/scoreboard?confId=80&seasonYear=2013&seasonType=2&weekNumber=3`

I ran the program during a game that had not finished to test that it would reflect score changes. You can see in the last line that Troy went from 0 to 6 points. You can also see that the program terminates after using Ctrl-C.



```
jessa:assignment01 vneblitt$ python3 TeamScore.py "Troy" 10 "http://scores.espn.
go.com/ncf/scoreboard?confId=80&seasonYear=2013&seasonType=2&weekNumber=3"
This is my answer for Assignment#1 Question#2
Troy: 0, Arkansas St: 13
Troy: 0, Arkansas St: 13
Troy: 0, Arkansas St: 13
Troy: 0, Arkansas St: 13
Troy: 0, Arkansas St: 13
Troy: 0, Arkansas St: 13
Troy: 0, Arkansas St: 13
Troy: 0, Arkansas St: 13
Troy: 0, Arkansas St: 13
Troy: 6, Arkansas St: 13
^CTraceback (most recent call last):
  File "TeamScore.py", line 61, in <module>
    time.sleep(int(timeout))
KeyboardInterrupt
jessa:assignment01 vneblitt$ 
```

Figure 11: Running the Program

Further Work

It seems like I should have been able to have some functions in here so maybe the code is not as elegant as it could be. Furthermore, it does not handle the situation where the "school" is not found on the page well. I had to remind the cataloger in me that the exercise's intention was to learn how to scrape a web page when I desperately wanted to institute name authority control. Name authority control would have taken care of the ODU v. Old Dominion v. Old Dominion University input problem. However, a cursory calculation of the number of schools represented revealed at least 120 schools potentially needing name authority control.

3 Web Graph Structure Exercise

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 graph above, give the values for: IN, SCC, OUT, Tendrils, Tubes, and Disconnected.
The only nodes I am reasonably sure about is that both E and F are disconnected. They have no connection to any of the other nodes and while F has E linking to it, nothing links to E. I started with D since it had a lot of links to it with no links coming out. This seemed to match with OUT. H had a similar situation. L links to D and nothing links to L so it looks like a Tendril. K has a similar situation. With this model it seems that all nodes are defined by their relationship directly or indirectly to SCC, but it was still hard to determine which nodes were SCC. A, B, C, G, N allows INs or other SCCs to pass through to OUTs. M is linking to an SCC so it is an IN. If I is an IN, then J is a Tube since it is not an SCC and I is reaching D without an SCC. Figure 12 is my diagram of the nodes.

- IN: I, M
- SCC: A, B, C, G, N
- OUT: D, H
- Tendrils: L, K
- Tubes: J
- Disconnected: E, F

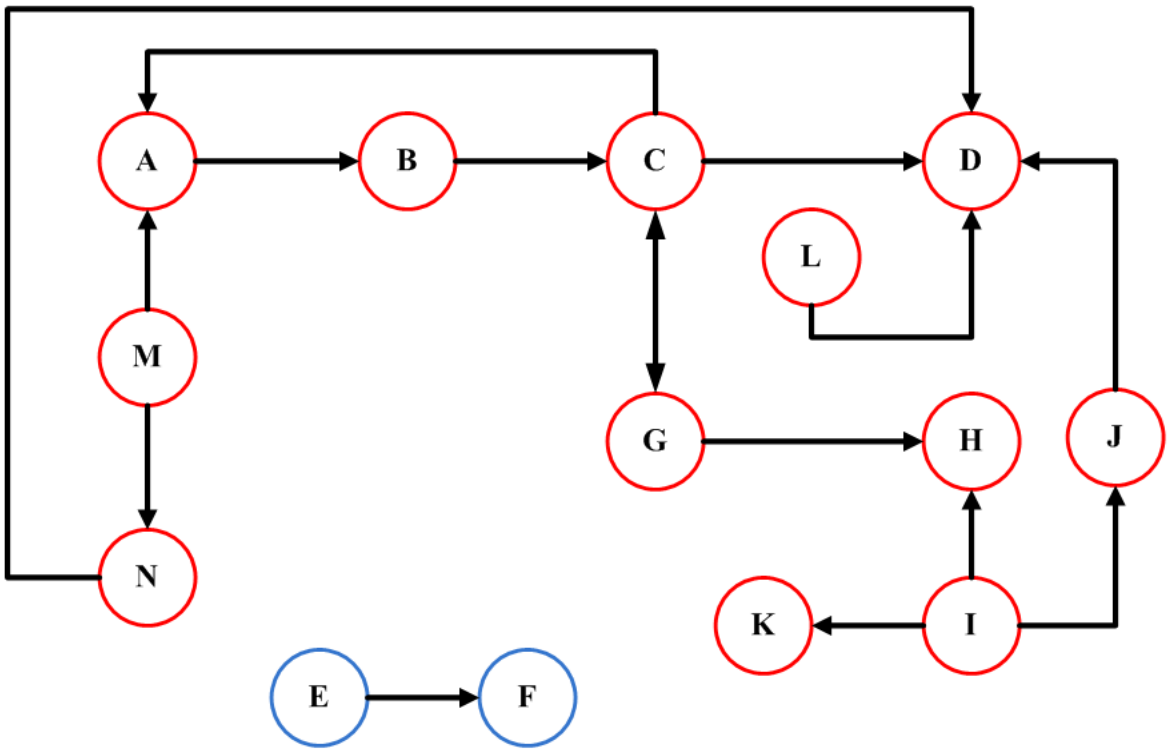


Figure 12: Visual representation of the connecting nodes