Faculty of Information
School of Graduate Studies
University of Toronto – St. George
Semester: Fall 2018
INF1340H – Programming for Information Systems

preliminary, 2018-10-30

Assignment 2 -- 14%
due: 2018 November 12, before midnight

Write a Python program that delivers the requested results. Include appropriate documentation: comments and docstrings.

1. HTML background

HTML (Hyper-Text Mark-up Language) files contain plain text which can be interpreted by trained humans, by web browsers, and by analytical software. For more information about HTML, please consult some of the resources listed at the end of this document. For simplicity, we will categorize HTML content into
- mark-up (also called *tags*, or simply *HTML*), and
- marked-up text.

The mark-up and marked-up text which we will process in this assignment are
- headers and
- tables.

Header tags include numbered openers, such as "<h1>", "<H2>", and "<h5>", along with their corresponding closers: "</h1>", "</H2>", "</h5>", etc. Table tags include "<table>", "<tr>", and "<td>" -- again with corresponding closers. The tag letters can be in uppercase or lowercase, or even mixed case. The case of the closer should match the case of the opener. (This rule was not in effect in early versions of HTML, by the way. Note also that some HTML coders are sloppy and sometimes omit a closer entirely.)

Headers in HTML are fairly straight-forward, and you might want to work on them first to build your confidence. Tables can become quite complicated. For instance, they can header rows (<thead>), footer rows (<tfoot>), body tags (<tbody>), and even tables within tables. You can meet the requirements of this assignment without getting too complicated, but you are welcome to go further and handle more if you are motivated.

2. analysis

The text analysis required in this assignment falls under the general category of *parsing*. Basically this means that you must recognize certain strings and substrings, and then decide how to process them. Python offers several tools and modules that could help you parse HTML, including

- string operators and methods
- regex module (regular expressions)
- HTMLparser module

Given our short time frame, I suggest focusing on the string and regular-expression techniques. String techniques are a little tedious, but they can get the job done eventually. Regular expressions have a bit of a learning curve, but you might be pleased with the time that they can save you. The learning curve for HTMLparser is quite steep, so proceed with caution if decide to try it.

There are other tools, such as PLY (which is a general-purpose lexical analyzer). You might need to download them separately. Please ask for the instructor's approval before using these other tools.

3. code generation

Your Python program should generate another Python program. As an example, the code below qualifies as a Python program, and its output also qualifies:

```
if  __name__ == "__main__":
    py_file  =  open( "helloWorld.py", "w" )
    py_file.write( """print( "Hello, World!" )""" )
    py_file.close()
```

This example shows a little bit more sophistication:

```
INDENT  =  "    "
STRING_ASSIGNMENT  =  "out_string  =  'Hello, World!'"

if  __name__ == "__main__":
    py_file  =  open( "helloWorld.py", "w" )

    # Write out the program files in three sections.
    #  The first and third sections contain fixed code (aka
boilerplate).
    #   This code include newlines and has been triple-quoted.
    #  The middle section contains the string assigned above.
    py_file.write( """#
# helloWorld.py
# (This file was generated by a2example.py)
# v0.0.0 - 2018-10-26
#
```

```
if  __name__ == "__main__":
""" )
    py_file.write( INDENT + STRING_ASSIGNMENT + "\n" )
    py_file.write( """    print( out_string )

# Wow! I didn't write this program.  My computer did!
""" )

    py_file.close()
```

And this second example also illustrates how to mix boilerplate code and variable code, which you will probably need to do for this assignment. Note that there is a lot of potential for confusion, as there is Python code that will run "now" and Python code that will run later -- all mixed together. Please be careful.

This code generation is unrelated to the central task of extracting and analyzing HTML. Perhaps you should defer working on this portion of the assignment until the rest of the project is well underway.

4. command-line arguments

You need to get the name of the HTML file from the command line. This is not difficult, and the steps are as follows:

- Include this import: `import  sys`
- The file name will be at `sys.argv[1]`

This example program illustrates how it works: (Let's call it cmd_arg.py)

```
import  sys

if  __name__  ==  "__main__":
    filename  =  sys.argv[1]
    print( "The file name is ", filename )
```

Run this program from the command line as shown in this example:

```
python3  cmd_arg.py  inf1340.html
```

Your program should display the following output:

```
inf1340.html
```

If you search for on-line help, you may notice that there is a module called argparse. This has features that go far beyond what is needed for this assignment. You should not need to use argparse, and in fact it is recommended that you not bother investigating it. (not a good use of your time)

5. reading (aka "cURLing") a remote file using HTTP

You need to retrieve the user-specified HTML file from its host (i.e., from its web server). This is not difficult, and the steps are as follows:

- Import the requests module.
- Retrieve the HTML file plus some meta-data via HTTP (Hyper-Text Transfer Protocol).
- Extract the raw HTML from the retrieved object. (The get() function retrieves more than you need. You should not need any of the meta-data.)

```
import  requests

http_object = requests.get( "http://website.com/webpage.html" )
raw_html  =  http_object.content
print( raw_html )        # not a pretty sight
raw_string  =  raw_html.decode('utf-8')
```

The print() statement is included here so that you can see what the text looks like before `decode`-ing. Please do not print your raw data in the submitted assignment. The raw string in the above example still needs some processing, e.g., splitting into separate lines.

6. what your program needs to do -- overview

Obtain a URL from the command line. The URL identifies the location of an HTML file to be analyzed. Also use the name of the HTML file to create an output .py file.
Retrieve the HTML file specified on the command line.
Analyze the HTML file, extracting any header and/or table, ignoring any other content.
Develop a string variable for the header, and a nested-list variable for the table contents. E.g.,

```
    header_text  =  "Table of Important Fires"
    table  =  [ [ ... ], [ ...
```

Generate a Python file which includes appropriate boilerplate code, as well as the appropriate header and list assignments.

The marker (TA or instructor) will

1) run your submitted code (a2HTML.py),
   python3 a2-nick-and-nora.py http://www.xyz.com/fire.html
2) then run the generated file (in this case should be called fire.py),
   python3 fire.py
3) examine the displayed output of the generated file, and
4) examine both .py files for coding, comments, and other programming issues.

7. sample HTML input and corresponding displayed output

```
<html>
<body>
<h3>Table of Important Fires</h3>
<table>
 <tr>
  <td>1666</td>
  <td>London</td>
 </tr>
 <tr>
  <td>1871</td>
  <td>Chicago</td>
 </tr>
 <tr>
  <td>1904</td>
  <td>Toronto</td>
 </tr>
</table>
</body>
</html>
```

Run the generated program, and we should see...

```
Table of Important Fires


1666 London
1893 Chicago
1904 Toronto
```

Your program must work with simple HTML files, and some samples for testing will be provided. You are encouraged to try to get your program to work on real-world examples.

8. tasks along the critical path

Please make sure (as soon as possible) that you are able to
- open and read a text file
- obtain a string from the command line
- download an HTML file from a server
- write (output) a text file
- write a .py file from a program and then run it

Your study and preparation for this assignment should include
- HTML basics
- HTML table basics
- regular expressions -- because you might decide to use them

9. more details

Numbered example HTML files can be found in qsand.com/1340/, e.g.
http://qsand.com/1340/example1.html

The minimum requirement for this assignment is to be able to process the first two examples (example1.html and example2.html) properly. Those files are relatively simple and the formatting is quite nice.

The other examples are not as nice, and they may be attempted if you have enough time. By successfully processing higher-numbered examples, you can earn bonus points which could offset deficiencies (if any) in your program.

Submit your work in a single file and call it a2HTML.py
Identify the starting point by using `if __name__ == "__main__":`

resources

https://www.w3.org/ They set standards for XML, and thus implicitly for HTML
Quercus/inf1340/Files/assignment for clarifications (Check at least once before you submit.)

to facilitate marking:

- Use the Quercus assignment dropbox, and if you worked as a team, submit only one copy per pair.
- Provide your name and your partner's name in a comment at the top of the program. Put your name on line 4, and your partner's name on line 5 (lines numbering from 1). Also, provide your section number (L101 for Tuesday, L102 for Thursday).
- Give your program a title (also in a comment at the top), and indicate your revision number and the date of the revision. Your program should be compatible with Python 3.5, and it is suggested that you indicate version compatibility in a comment, too -- i.e., which version(s) of Python will your program run on?
- If you add extra features to your program, please ensure that they do not prevent smooth, simple operation. Try not to confuse the markers. **Do not ask the user any questions at run-time.** (Hint: If you need additional information from the user, use optional command-line arguments.)
- If you know, at submission time, that your program does not run successfully, please make a note of the known deficiencies in a comment at the top.
- Describe appropriate test cases in your docstring.
- Use prescribed names for constants, variables, and functions. Please draw upon this list as needed:

```
# strings containing Python assignment statements:
HEADING_assignment
TABLE_assignment

get_HTML_lines()
extract_heading()
extract_table()
extract_rows()
extract_cells()
write_output_py_file()
display_table()
```

- If your program is not easy to read, add comments, whitespace, etc., as appropriate. If your functions are long, rewrite them so that they are shorter, or split the code out into additional functions. (Note that if you use programming tricks to shorten the code, you should add comments to explain the tricks.)

example of first lines in your .py file:

```
#
# inf1340, section L101
# assignment 2 - due 2018-11-12
# Steed, John                    << Names on lines 4 and 5
# Peel, Emma                     << Blank line if no partner
#
# HTML table extraction program
# v1.0.3 - 2018-11-11
# compatible with Python versions 3.4 - 3.7
# source file:  a2HTML.py
#
```

scoring: (out of 100 possible)

declaration and use of constants as prescribed	5 marks
headers, docstrings, and bodies of functions:	
to obtain URL (and filename) from command line	10 marks
to download HTML file for analysis	10 marks
to parse the HTML file	20 marks
to write a context-specific .py file	15 marks
to format and display the extracted data	10 marks
other code, including __main__	5 marks
coding style and comments	10 marks
Program produces correct output.	10 marks
facilitation of marking (as described earlier)	5 marks

Don't be late! There is a 15% per day (or part of day) penalty for late submission.