Homework 2

[ 100 points - due by 11:59 pm, Sunday, February 5, 2017 ]

Submit these files to the CS submission system at the usual place by 11:59. You may work on your own or with 1-2 partners on the programming portions of this assignment. (The reading/response is individual only.) Groups larger than 3 should split into smaller groups. Remember that partners need to work in the same physical location, share composition time equally (or each compose on their own machines) and be fully equal owners and producers of their work. Have fun experimenting! [cs35 homepage](https://www.cs.hmc.edu/~dodds/cs35/)

**Downloads**

Starter files to download -- grab them at the start of class & follow along:

* [The zip file to start all of this week's problems...](https://drive.google.com/open?id=0BwPWh-3AmiLxV0lMR29MYXlfNW8)

**Setup**

Let me know if your Python and/or submission-site account isn't yet working - we'll help fix it!

**Submission**

Although there are several files in each assignment, for ease of handling them, we'll ask you to submit a single zipped archive named hw2.zip with all of those files (except the reading response, which is on its own) at the [usual submission site](http://cicero.cs.hmc.edu/).

**Problem 0: Reading and response**  (**hw2pr0.txt** - 5 pts)

We're using Python to write web languages this week, so take a big-picture view at <http://www.evolutionoftheweb.com/> ! Load that page and take a look at the timeline and technologies that have defined a significant fraction of the modern computing experience (though it ends in 2012!) There's too much to look over it all carefully, but do the following sampling: Choose four technologies from the graphic including HTML and CSS2 as two of them. For those two (and two others), click through to a description of the technology and read a bit about what they are. Then, in your reflection, respond to either of these two prompts: (**a**) *What amount of overlap with the mechanisms of web presentation do you feel is meaningful for the path you're on?* (**b**) *How do web technologies differ/compare to traditional programming languages such as Python? Which would you give more emphasis in the curriculum?* As always, a short thoughtful answer is all that we're looking for here! Since this is done individually, submit this as hw2pr0.txt, not in your hw2.zip archive.

**Problem 1: Lab ~ diving into web technologies** (**hw2pr2.py** - 35 pts)

Problem 1 this week is the lab problem -- join in for lab on Monday evening (after the presentation part of class), or T/W evenings to try it out. As usual, if you do come to lab, you'll receive full credit on the lab portion of the hw.

Lab this week has two parts. For part 1, don't spend more than 10-15 minutes!

Part 1: [**5pts**] Find a webpage you'd like to investigate/modify and use the *Inspect* option to view and edit the source of the page (well, the copy of the page that's on your computer!) Change at least three of the content elements of the page (text, images, etc.) and take a before/after screenshot to contrast your changes. Include both screenshots in your hw2.zip archive.

Part 2: [**30pts**] Presidential/Unpresidented… For this rest of this lab, begin with the simple presidential\_start.html file in the Problem 1 starter code. Your task is to use HTML and CSS in order to create (1) a more polished version of this file and (2) an HTML-horrible version!

[**the "polished" version**] Create one altered version, named presidential.html, which improve upon the formatting of the original so that it is a more polished presentation -- of the data. You should use online resources to guide your changes (I like [w3schools](http://www.w3schools.com/)), and should include *at least* the following:

* Add at least two rows of data! Feel free to use arbitrary-placeholder data, as with the original page (*for now* ) Later, you'll add your analyses' results...
* A much-neatened table of data, for example, using w3school's [HTML tables](http://www.w3schools.com/html/html_tables.asp) and/or [CSS tables](http://www.w3schools.com/css/css_table.asp) resources as a guide.
* A better-centered or more careful layout of the elements on the page, for example, using the [HTML layout examples](http://www.w3schools.com/html/html_layout.asp).
* Links from the presidents' names to their Wikipedia pages. You might use the [HTML links](http://www.w3schools.com/html/html_links.asp) and/or [CSS links](http://www.w3schools.com/css/css_link.asp) resources here.
* Experiment with the background, e.g., with colors and/or images. References I've used are [CSS background](http://www.w3schools.com/css/css_background.asp) and [CSS colors](http://www.w3schools.com/css/css_colors.asp).
* Include one more thing you choose from the HTML/CSS/Javascript resource of w3schools. It could be [images](http://www.w3schools.com/html/html_images.asp), or [buttons/other events](http://www.w3schools.com/js/js_events.asp), or something completely different.

[**the "horrible" version**] Plenty of sites with jarring layout and formatting choices are out there - this asks you to join them! websites are stuck in the 90's - or worse. The challenge is to create the *least-polished*  version of the data. The experimentation above might lead you to some ideas. Call this version unpresidented.html . (Want a challenge? Blinking text and auto-playing music are a great way to add to the horribleness of an HTML layout… So are rotating things. Seek them out!) And, do keep everything in the good-natured, light-hearted spirit of this assignment, which our grutors will be going over… .

Include both your presidential.html and your unpresidented.html files in your hw1.zip archive. The graders will run an informal contest for both the most-polished and least-polished pages… .

[**EC**] Then, if you'd like, show off your analysis from last week! Use your last week's code to change the data in the table so that it reflects one or more of the analyses of the inaugural addresses you built last week. With this, you should have a web-ready file -- (the good one, presumably) -- and, if you'd like post this and your code on your GitHub repository. See the extra-credit, below.

**Problem 2: Python as web developer...!** (**hw2pr2.py** - 30 pts)

By-hand web development is common, but it's possible -- for special-purpose tasks -- to leverage Python's strengths to *generate* web content. This is the underlying idea of many web template engines (Django and Flask are two of the most popular in Python). This problem asks you to write a Python go-between for CSV (comma-separated values files) and the web languages HTML/CSS. Both CSV and HTML are foundational methods for sharing information: CSV is the raw material of Excel, Google Sheets, and any spreadsheet program, just HTML is for browsers.

The task is to create a table of *letter-frequencies* in the spirit of the ones on [this Wikipedia page](https://en.wikipedia.org/wiki/Letter_frequency). The difference is that you will be answering *your own questions* with the data -- and then using Python to generate the output in HTML and CSV.

Building from the problem 2 examples, you should design and name the functions as you see fit, but here are the pieces to handle:

* Your project should read all of the data from wds.csv, which is a file of 10,000 word-frequencies from Project Gutenburg ([link](https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/PG/2006/04/1-10000)). This example code has a first-letter-counter that is *unweighted*, that is, it does not take into account the relative frequency of each word in English:

from collections import \*

#

# unweighted counting of first letters!

#

def UWcount():

""" returns a dictionary (defaultdict) of

unweighted first-letter counts from

the file wds.csv

"""

LoR = readcsv( "wds.csv" ) # List of rows

print("LoR is", LoR)

counts = defaultdict(int)

for Row in LoR:

word = str(Row[0]) # the word is at index 0

num = float(Row[1]) # its num occurrences is at index 1

first\_letter = word[0] # the first letter of the word

counts[first\_letter] += 1 # add one to that letter's counts

# done with for loop

return counts

Your task is to create (1) a first-letter-counter that is *weighted* : it *does* account for the relative frequencies of words in English (according to that file). Then (2) and (3) create two more letter-frequency analyses of your own design using the *weighted* data. For example, you might ask the relative frequency if the last letter of words in English (or second letter or second-to-last). Alternatively, you might ask the most-frequently-found *position* of each letter in English. (Where is 'z' most likely to occur?) Or, you might ask the frequency of the letter in the middle of words (handle ties as you see fit - but deliberately!). Or the frequency with which each letter appears twice or twice-in-a-row in English words. Feel free to use two of these -- or devise something different.

* Then, your project should **write out to a csv file**, named frequencies.csv, a table of letters and their relative frequencies. Use the provided example of writing to a CSV file as a starting point and/or [Python's reference](https://docs.python.org/3/library/csv.html)!
* Create a function that reads in your CSV file data from the previous part and then outputs a string of HTML that formats that data as a table. It does not need to be as polished as presidential.html, but it should look neatly formatted.
* Be sure to describe, briefly, how we should run your project. main(), perhaps?
* Finally, create (most likely by hand) a webpage named letter\_frequencies.html that shows off the results of your analysis. You're welcome to paste the results of the previous part, by hand, into your letter\_frequencies.html file. (Other, more automated approaches are extra - if you try writing the whole page directly to a file using Python, be sure to highlight that fact!)

**EC options** Also extra are (1) adding this letter\_frequencies.html file and the Python code that helped created it to your repository, (2) adding any graphical elements - e.g., Wikipedia has bars showing relative sizes of frequencies with different colors for vowels and consonants. Or anything else, as always!

**Problem 3: Annotating text for the web** (**hw2pr3.py** - 30 pts)

Problem 3 this week is a very different combination of Python and Web languages, inspired by this [XKCD panel](https://xkcd.com/1288/) (so popular, it inspired [another](https://xkcd.com/1625/) and [another](https://xkcd.com/1679/)!) In this problem, you'll automatically substitute and/or annotate a text by applying CSS's color and title attributes.

Here's an example page to get the idea: it's a snippet from Hamlet, Act 1, Scene 4, in which some of the words (the ones I considered unfamiliar) have been annotated with synonyms. The basic mechanism is to replace simple text, e.g.,

The king doth wake tonight and takes his rouse

With styled text. Since CSS styling must be within a tag, we use the generic <span> </span> tag with inline styling. Here is the result from that page (check the source!)

The king <span style="color:rgb(0, 0, 150);" title="does">doth</span> wake tonight and takes his <span style="color:rgb(0, 0, 150);" title="partying">rouse

Your task is to create a Python project (in the file hw1pr3.py) whose functions do this automatically, *by reading in a csv file with the substitutions you'd like!* Here's one possible outline:

* Create a function that reads in a csv file of substitutions, with two columns: the left column is the original, the right is the annotation. Your function should return these as a dictionary (or defaultdict)
* Create a function, we called it annotate\_text( text, subs ) , which takes as input
  + text, any (possible large) string of text
  + subs, a dictionary or defaultdict of {string key : string value} pairs that are the annotations or substitutions
  + Then, annotate\_text should output a string of styled HTML that can be displayed in a webpage.
* Feel free to do what we did above (only annotate) or you can substitute (and keep the original as annotation) or some other combination. The essential piece is to alter the text using the substitution dictionary via CSS styling.
* Be sure to tell us how to run your code - and include a working example!

Here's an example call to annotate\_text. First, two variables for input:

text = "The king doth wake tonight and takes his rouse"

subs = {"doth":"does", "rouse":"partying"}

Then, the call to annotate\_text( text, subs ) returns this larger string:

The king <span style="color:rgb(0, 0, 150);" title="does">doth</span> wake tonight and takes his <span style="color:rgb(0, 0, 150);" title="partying">rouse

*You'll need those double-quotes -- watch out! (Use Python's single-quoted strings)*

Since this works for general texts and substitutions - *congratulations*, you've built a web templating system in Python! Small, perhaps, but full-fledged!

[**Create an example page!**] Create at least one HTML page that includes your own substitutions (or, perhaps, some of XKCD's). Feel free to hand-copy your original and annotated texts into the page. Single-word substitutions are most natural at this point; feel free to use only those, as well.

Be sure to tell us how to run your text-annotater!

**EC options** Multiple-word substitutions are a challenging extension (we'll practice with a technique called *regular expressions* later in the term that makes those much less challenging.) But there are many paths you could follow from here, e.g., adding more csv columns with color information, or including background color, or borders, or different fonts (tricky!), or handle mouseover with javascript events, or … . Phew! Welcome to the world of web-content automation -- it's *far larger* than the world of hand-built web content!

**Problem 4: Where’s the Grutor!?** (**hw2pr4.py** and **hw2pr4.html**- ?? pts)

If you don't have the files from last week, [here](https://github.com/ScriptingBeyondCS/CS-35/tree/master/week_0_to_2) they are!

The grutor has now been missing for 3 weeks! Last week’s results gave you 5 possible locations he could possibly be, but you have neither the time nor the resources to search each one. You decide to continue rummaging through the laptop in search of more clues. In ***<path here>*** you come across a folder called *last\_hope*. There are two files within in it. Open these files now. One is a text file with a jumble of numbers on the top and some gibberish in letters at the bottom. The other is an Excel file with a spreadsheet with two columns of letters. You realize the Excel file is the code for deciphering the letters at the bottom of the text file. Each set of 5 letters in the text file corresponds to one true letter. Because you are efficient, you decide to write a script to decode the message rather than doing it manually.

**To do:** Write a function that takes in a string of text, parses the letter-matching code in message.xlsx, and prints the true message based on this code. Use the following code from [Stack Overflow](https://stackoverflow.com/questions/7372716/parsing-excel-documents-with-python) as an example of how to parse Excel data (very similar to csv files):

from openpyxl import load\_workbook

wb = load\_workbook(‘Book1.xlsx’)

ws = wb.active

for row in ws.iter\_rows():

for cell in row:

print(cell.value)

You have successfully deciphered the message, and it is clear that the grutor was afraid he might have to disappear and left his laptop for someone to find him. But the riddle is still unclear, so you show the message to his roommate. His roommate tells you that “my number” might refer to 42, the grutor’s favorite number. If this is the case, the poem is telling you to make all the factors of 42 in the file light green. But how would that help? Nevertheless, you decide to give it a try.

**To do:** Write a function that reads message.txt and outputs an html file in which all the single-digit factors of 42 (1, 2, 3, 6, and 7) appear lightgreen (this will need to be one complete word when you write your function).

Congratulations! You have successfully deciphered the code and you travel to the specified location and find the grutor. He tells everything that transpired. ***Finish the story Forest***

**Extra Credit: Post your work!** (up to +10 or more points, **hw2ec.txt**)

One of the goals of cs35 is to create a portfolio that will show off your final project. If you like, it'd be wonderful to show off some of the intermediate projects along the way. This week has invited you to create several things you may want to show off:

* the polished presidential.html page (and, if you'd like, the unpresidential.html page!)
* a letter-frequency analysis of your own design and rendering via Python (frequencies.html)
* a before-and-after page of substitutions/synonyms (or other annotations)

All of these involve algorithms of your own design and implementation -- and they use your own Python-based HTML/CSS renderer for the final version. If you'd like, show them off on your GitHub account and point us to the results by including links in a file named **hw2ex.txt**

**Good luck with homework #2, everyone!**