A zipped folder is provided, linked next to this handout on Canvas: **h\_7\_starting.zip.** It contains starting **.py** files for each of the following problems. Use them if you wish.

At the start of each of these problems, the name of a Python file is given in **blue**: **foo.py**. You should create and save the requested Python program source code in a file with the same name. Also add a comment at the top of each giving your name.

When finished, upload each **.py** file with the specified name to the Canvas **H7 Assignment** link. For **[H7-4]**, upload your **.mp4** video (media) file to the separate **H7 Video Assignment** link.

**[H7-1]** (**count\_alice2.py**) Continue with refining your **[L8-3]** **count\_alice.py** code, which reads and analyzes **alice.txt**. Your goal here is to create a program that prints out a list of all lower-cased words in the text file without words including unnecessary punctuation. They should be printed in ascending alphabetic order, one per line. At the end, print out the number of words.

**[H7-2]** (**unidump.py**) Write a program that reads an integer **N**, then prints out the Unicode characters from **chr(32)** through **chr(32+N-1)**, 32 characters per line and formatted as described below. Note we are skipping the first 32 characters **chr(0)** through **chr(31)**; these are the ASCII control characters, which do strange things when printed to the console. But as you will see when running your code, printing other Unicode characters also sometimes has strange results.

Note there are no spaces between adjacent Unicode characters in the output, and the start of the first line **0032:** is a blank/space, which is **chr(32)**

The Unicode number **ord(c)** of the first character **c** of each line should preface the line, right justified in a field of width 4 followed by a colon and a blank, as shown below. The **H7** Help Video shows how to do this using the **%** operator, and the starting code provides nested loops which you can extend to print the pattern below.  
Use Python's built-in **chr(N)** which returns the **N**th Unicode character. For example, the output for **N == 224** should look something like this:  
   
**0032: !"#$%&'()\*+,-./0123456789:;<=>?**

**0064: @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_**

**0096: `abcdefghijklmnopqrstuvwxyz{|}~**

**0128: **

**0160:  ¡¢£¤¥¦§¨©ª«¬­®¯°±²³´µ¶·¸¹º»¼½¾¿**

**0192: ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß**

**0224: àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ**

'Keep in mind that the Python console may not display all Unicode characters correctly, and your particular operating system may affect which Unicode characters are displayed. Example: in the above output on MacOS X, many of the characters in the range **chr(127)**- **chr(150)** don't display correctly, and will likely appear differently in your console than in the above example.

And… you'll see even stranger output for larger **N**. Very large **N** may result in runtime errors: don't worry about this.

**[H7-3]** (**list\_quiz.py**) Write a program that gives a short one-question quiz on **Python lists** to the user, then grades it and presents the results. Your question should test on those topics discussed in HTT10 ("Lists").   
  
Your quiz should be interactive; it should print a question, read the user's answer and print if it's correct or not. If it's not correct, give the correct answer. In both cases, print out a brief description of why the correct answer is correct.   
  
You might present multiple-choice questions, or else a question that asks the user to enter the output printed by some given Python code. In either case, your program should be able to read the user's response and determine if it's correct or not.

[Continued on next page...]

An example of the latter (but you cannot use this or a similar question in your quiz) is:  
  
**Enter the output of the following code:   
  
 s = 'moxie, sasha, sandy'  
 slist = s.split()**

**result = [elt[:-1] for elt in slist]   
 print (result)**  
If the user enters: **['moxie','sasha','sand']** then output might be: **Correct!**   
  
If the user enters: **['moxie','sasha','sandy']** then output might be: **Sorry, that is incorrect.**  
  
In both cases, the explanation of the correct answer should be printed, like this:  
  
**s.split() returns a list of strings delimited by whitespace in the original - not by commas. Thus, slist = ['moxie,','sasha,','sandy']. Then the list comprehension builds a new list result that trims the last character from each of slist element. So result is:**

**['moxie','sasha','sand']**

An example solution **list\_quiz\_example.py** to the above is provided within the starting code. Feel free to use it - except you must provide your own quiz question.

**[H7-4]** (**4 points**) (**NAME\_quiz.mp4**) Record a short desktop video that presents your quiz question and solution for the previous problem **[H7-3]**. Use Zoom to do this, saving your video file to your local PC under the name ***NAME*\_quiz.mp4**, where ***NAME*** is the first 4-5 letters of your last name.

A tutorial video link is posted next to this handout, showing the details of using Zoom. "How to Record a Zoom Video" details how to use Zoom to record your shared desktop and audio. It discusses details of the FP-3 assignment, but it will also help you to record the video for this problem.

In your **[H7-4]** video here, do each of the following:

- Show a simple slide that presents your name, class/section number, and a text presentation of your quiz question.

- Run your code and enter the correct answer. Discuss the resulting output and why it occurs.

- For each incorrect answer: rerun your code again and enter an incorrect answer. Discuss the resulting output and why it occurs.

- Show your Python code for **list\_quiz.py** and describe how it works.

Along with your **.py** files, upload your **.mp4** video file to the separate **H7 Video Assignment** Canvas portal for this assignment.