A zipped folder is provided, linked next to this handout on Canvas: **h7\_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 **HW 7 Assignment** link.

**[H7-1]** (**count\_alice2.py**) Continue with refining your Lab 8 **count\_alice.py** code, which reads and analyzes **alice.txt**. Your goal 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.

**[H7-2]** (**unidump.py**) Write a program that reads an integer **N**, then prints out the Unicode characters from **chr(32)** through **chr(N-1)**; that is, print the first **N-32** Unicode characters. Print 32 characters per line, 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. (As you will see when you run your code, other later Unicode characters will also result in odd output...)  
  
The Unicode number **ord(c)** of the first character **c** of each line should preface the line, right-justified in a field of width 6 followed by a colon and a blank, as shown below. The H7 Help Video shows how to do this using the **%** operator in class.  
Use Python's built-in **chr(N)** which returns the **N**th Unicode character. For example, the output for **N == 256** 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]** (**dedupe\_list.py**) Write a program that reads a list directly from the user, then removes duplicate top-level elements in it and prints out the resulting list. You may not use the Python **set** type to do this problem.

Use **eval(input("input list: "))** to read a list from the user. **eval(s)** is a built-in Python function which evaluates its **str** argument **s**, converting it to the proper Python type and returning this value. (It's the Python interpreter as a built-in function.) The starting program contains such code.

**[H7-4]** (**strip\_comments.py**) Write a program that reads a string **fname** from the user, then opens the file **fname.py** (Python source) with that name. Copy each of its lines into a new file named '**strip\_'** **+** **fname.py**, deleting all Python comments as you go. Recall these begin with a **#** anywhere on a line and continue to the end of the line. Your output **.py** file should still be a valid program.

**[H7-5]** (**list\_quiz.py**) Write a program that administers a short one-question quiz on lists to the user, then grades it and presents the results. Your question should test on these 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.

An example of the latter 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's elements. So the final value of result is:**

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