Directions

You will create a new **YourOhioIdHW4.py** file which will include comments and code to address the following problems.

IMPORTANT: Be sure to read and adhere to the provided "coding expectations" document to avoid losing points.

Problems

For the following problems, write the necessary code to produce answers to the problems. Note, I expect to see techniques/data structures/functions/methods/etc. that were demonstrated in the learning activity to produce the necessary output for each of the problems.

1. We will start with some basic loop structures. As you have seen, a for loop can use a range (*e.g.* ***for counter in range (3)*** *allowed us to produce output of 0, 1, 2 in the learning activity*). We can override the starting value of the range by providing a start value and ending non-inclusive value (*e.g.* ***for counter in range (1,4)*** *would allow us to produce an output of 1, 2, 3 in the previous example*). Next, a chr function (*similar to an str function*) will allow you to take a numeric value and translate it to its ASCII equivalent value (*you can see this website http://www.asciitable.com/ for the* ***dec****imal equivalent values of the uppercase letters*). Write a for loop that only uses range numeric values to print out the uppercase alphabet.

   Expected output example (*this output has been shortened to save space – your list will show A-Z*):

   A
   B
   C
   …
   X
   Y
   Z

   Hint:
   Pay attention to the example ***for counter in range (1,4)*** and consider what was said in the learning activity about the word that follows "for," in this case "counter" – that value gives you access to the range. If you look at the provided website, you can see that the value for capital A is 65, and the value for capital Z is 90. Review module 1 where sting ranges were discussed and recall that "the second number in the range is not inclusive of the data."

2. Copy your code for the previous problem and modify it so that the output is horizontal and comma-delimited.

   Expected output example (*note, the comma at the end is expected*):
   A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,

Hint:
Pay attention to what was said in the learning activity about how change the way that the print function works.

3. Re-write the code from the previous problem so that uses a while loop instead of a for loop (*you will need to introduce new variables to keep track of your starting range and ending range – think about where this needs to happen, e.g. inside the loop or outside of the loop*). Your output should be the same as the previous problem.

   Expected output example:
   A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,

4. Recall how we have previously used ranges on string variables (*e.g.* ***print (myMessage [0:7])*** *displays characters within the string from an index of 0 through 6*). Re-use the code from either of the previous 2 problems (*the for or while loop*) and modify it so that the output is **stored** in a string variable **rather than using the print function to generate the output within the loop**. Once you have created that new string, leverage your understanding of range and a length (len) function to print that variable (*this will occur* ***outside*** *of the loop*). Your output will be the same as your previous output **minus** the trailing comma (*think range – from the beginning of the string to the end of the string minus 1*).

   Expected output example (*note, no comma at the end this time*):
   A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z

   Hint:
   When using the "+=" assignment operator (*in this case, within the loop*) it will add the contents of the left operand to the right operand. If the right operand is a number, the "+=" operates as addition. For example:
   intValueWeWantToKeepTrackOf = 0
   intValueWeWantToKeepTrackOf **+=** 1
   intValueWeWantToKeepTrackOf **+=** 3
   The new state of intValueWeWantToKeepTrackOf now is 4. This is like the example in the learning activity related to tip, food, and tax and how they were added to the total.

   If the variable you are working with is string data type (*as is the case here since the chr() function generates string output*), then the "+=" operates using concatenation, meaning it will append the left operand to the end of whatever is in the right operand. For example:
   strValueWeWantToKeepTrackOf = "" *# initialize the empty string variable before the loop*
   strValueWeWantToKeepTrackOf **+=** "test1" + "," *# concatenate to the string within the loop...*
   strValueWeWantToKeepTrackOf **+=** "test2" + "," # *etc....*
   The new state of strValueWeWantToKeepTrackOf once the loop is finished is now "test1,test2,".

5. Using a **for loop** (*if you use a while loop, the problem is wrong*), build a dictionary where the key is the uppercase alphabet letter and the value is the associated numeric value (*basically, what you have just been looking at in the previous problems*). **You are only permitted to use a maximum of 2 lines of code to do this** – 1 line for the "for declaration line" and 1 line of code within the loop (***yes**, before the loop, you will need to create an empty dictionary, and you likely have startValue and endValue variables, and following the loop you will have 2 lines print out the keys and values of the dictionary which means that you will have more than 2 lines of code, I am only restricting the number of lines within the loop structure – not what you do before or after the loop*).

   Expected output example:

```
dict_keys(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
'T', 'U', 'V', 'W', 'X', 'Y', 'Z'])
dict_values([65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90])
```

   Hint:
   Think about how a dictionary is created:
   myDictionary[key] = value # how do you generate the key? # where does the value come from?

6. Next, build a new dictionary variable (*like the previous problem*), but this time build it using a while loop, read through each key and value of the dictionary that you created in the previous problem and store them in a new dictionary. Outside of that while loop (*meaning this loop exists outside of the loop that you just wrote to build the new dictionary*), use a for loop to print the keys and values of that new dictionary.

   Expected output example (*this output has been shortened to save space – your list will show keys A-Z and values 65-90*):
   key: A value: 65
   key: B value: 66
   key: C value: 67
   ...
   key: X value: 88
   key: Y value: 89
   key: Z value: 90

7. Now let's switch gears into using external datasets. For the remaining problems you are required to use the CSV module (*only – no other data reading modules are permitted*) to read in our external data. Also, as a good practice, **always** include error handling in code that reads in external data (***similar to the last example demonstrated at the end of the learning activity, e.g. test to see if the file exists and include try/except***) – failing to do this for all remaining problems will result in the problem being incorrect.

You have been provided with a dataset (*wineData.csv*) which contains more than 2 million rows of wine-related data (*the dataset contains an **id** used to uniquely identify the wine, **points** awarded, a **price**, and **location** describing where it came from*). Use the **csv.reader** function to read the first 10 rows (*think about how a counter can be used here*) of the file and generate output just like the expected output (*include the "row #" in front of each row*).

Expected output example (*use a counter to limit your output*):

```
row #1: ['id', 'points', 'price', 'location']
row #2: ['0', '83', '89', 'Ticino']
row #3: ['1', '91', '4', 'Dealu Mare']
row #4: ['2', '89', '21', 'Judean Hills']
row #5: ['3', '91', '18', 'Codru Region']
row #6: ['4', '83', '38', 'Haut-Judeé']
row #7: ['5', '81', '93', 'Victoria']
row #8: ['6', '81', '61', 'Maule Valley']
row #9: ['7', '99', '97', 'Pafos']
row #10: ['8', '88', '95', 'Korinthia']
```

Extra code to consider and Hint:

In the learning activity we used a "with" command to open our "data.csv" file like this:

```
with open("data.csv") as fileWithData:
```

It is pretty common when working with data to encounter characters that your code may not like resulting in an error like this:

```
UnicodeDecodeError: 'charmap' codec can't decode byte
```

If this happens to you, the work around is to describe the type of data encoding like this:

```
with open("data.csv", encoding="utf-8-sig") as fileWithData:
```

8. Leveraging a counter, print out the answer to "how many rows of data are in the dataset (***excluding** the header row*)?" (*note, that you **cannot** open this file in Excel since it is larger than the allowed possible rows in Excel*) Also, consider that you can just keep track of the counter in a for loop, but you should not be printing out each number within the for loop or it will take a very long time to run; instead, only print out the final answer after you have counted all of the rows.

Expected output example:

```
2097150
```

Hint:

This is pretty similar to the learning activity – just keep track of the counter rather than printing out rows.

9. Determine statistics about the dataset including the lowest, average, and highest points and prices for wine.

Expected output example:

```
lowest points awarded: 80
average points awarded: 90.0
highest points awarded: 100

lowest price: $4.00
average price: $52.00
highest price: $679.00
```

Hint:

Use lists to keep track of the points and price data when reading the dataset (*be sure to avoid the header row – e.g., something like counter > 1*). Also, think about the listWithIncomeData example from later in the learning activity where you were casting a specific column of data as int (*even though int might not be the best choice for data that might have decimal points, like price data*).

10. **(BONUS +2 POINTS – no partial credit for this one and no hints)** While reading through the data, build a dictionary where the id will be stored as the key, and the value for that key will be a tuple that will store the points, price, and location. Note, the id should be stored as an integer; the points and price will be stored as an integer within the tuple and the location will be stored as a string. Once the dictionary has been built, outside of your "with" statement that was used to build the dictionary, write a for loop that will read through the dictionary and print the keys and values where the key is either less than 10 or the key is greater than or equal to 2097140.

Expected output example:

```
key: 0 value: (83, 89.0, 'Ticino')
key: 1 value: (91, 4.0, 'Dealu Mare')
key: 2 value: (89, 21.0, 'Judean Hills')
key: 3 value: (91, 18.0, 'Codru Region')
key: 4 value: (83, 38.0, 'Haut-Judeé')
key: 5 value: (81, 93.0, 'Victoria')
key: 6 value: (81, 61.0, 'Maule Valley')
key: 7 value: (99, 97.0, 'Pafos')
key: 8 value: (88, 95.0, 'Korinthia')
key: 9 value: (82, 85.0, 'Goumenissa')
key: 2097140 value: (98, 72.0, 'Piedmont')
key: 2097141 value: (81, 72.0, 'California')
key: 2097142 value: (87, 78.0, 'Upper Galilee')
key: 2097143 value: (97, 56.0, 'Viile Timisului')
key: 2097144 value: (84, 74.0, 'Constantia')
key: 2097145 value: (95, 44.0, 'New York')
key: 2097146 value: (87, 78.0, 'Ella Valley')
key: 2097147 value: (90, 55.0, 'Terras do Sado')
key: 2097148 value: (81, 64.0, 'Wairau Valley')
key: 2097555 value: (82, 42.0, 'Spain Other')
```