Directions

You will create a new **YourOhioIdHW3.py** file (replace YourOhioId appropriately) which will include all comments and code necessary to address the following problems.

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

Problems

Note, each of the problems that involve conditional logic (*which is most of them*) should only include a single if statement. That is not to say some might include complex if statements, like nested if statements – for example, this is a “single if statement” since there is one “large if” statement that contains a “nested if” – that is OK:

```
if (condition):                    # consider this to be the "large if"
    print("doing this stuff")
    if (anotherCondition):         # here is the nested if
        print("do more stuff")
    elif (otherCondition):
        print("do more stuff 2")
    else:
        print("do more stuff 3")
else:                              # the else for the "large if"
    print("don't do stuff")
```

Consider that their will only be one “large if” statement for each of your problems. There is no need to copy the large if statement and paste it multiple times to test the various scenarios in each of the problems. **Instead**, simply **execute the entire block of code multiple times to test your different expected output**.

For example, in problem 1, you have 3 different scenarios to test, 1st where a value is less than another value, a 2nd where the values are equal, and a 3rd where a value is greater than another value. You will write a single if statement (*you decide on the type of if statement, like a simple if/else, an if/elif/else combination, or nested if – this is up to you*), and then execute that if statement 3 times to generate the output for the 3 scenarios. If it turns out that you need to include multiple screenshots to demonstrate the output, that is fine (*see the “screenshots” section of our coding expectations document*). If you can do it with one screenshot, that is also fine.

1. We will start with some basic conditional tests. Ask the user for 2 numeric values (*input function*) and store each value in a variable. Use conditional logic to demonstrate the following output (*demonstrate all 3 scenarios*).

   Expected output example
   (*the following are just example values, you can change them to whatever you want, but I want to see <u>all 3 scenarios demonstrated</u> in your output – note, I made the text* ***bold*** *just to help you identify what prompts to ask the user in your console window – do not worry about making your text bold when you write your code*):

(*scenario 1 – less than*)
Please enter value 1: 10
*(your console window should show "Please enter value 1: " and you can assume that the number **10** is entered as a response to your prompt)*

Please enter value 2: 20
(repeat the same idea, but for value 2..., but 20 is entered this time)

10 is less than 20
(*this is your output based on your conditional test – i.e. is 10 less than 20, yes "10 is less than 20" is your exact output; if value 1 is 11 and value 2 is 33, then the output would be "11 is less than 33" etc.... **let's consider this to be <u>scenario 1</u>** – you do not need this note in your output about scenario 1, I just wanted to make it obvious as to what the scenario looks like – after you get your screenshot for scenario 1, just re-run the same code and enter the values for scenario 2 and scenario 3 – if it can all fit in a single screenshot, that is fine – if not, just capture the enter Spyder window for the first scenario and then just the console output for scenario 2 and 3*)

(*scenario 2 – equal to*)
Please enter value 1: 20
Please enter value 2: 20
20 is equal to 20

(*scenario 3 – greater than*)
Please enter value 1: 21
Please enter value 2: 20
21 is greater than value 20

Hint:
Recall that all data entered using an input function is of data type **string**. So, entering a value of 10 is actually "10" with double quotes around it, **<u>which is NOT a number</u>**. Like a **str()** function, an **int()** function can be used to convert a string to a number (*go back to module 1 if you need a refresher on how str was used*) – an example could be **value1 = int(stringValue1)**. You **<u>will</u>** need to use the **int** function to change your value from str to int prior to using your values when you perform your conditional test. Also, for this problem, we will ignore error handling and **<u>assume</u>** that the user only is permitted to enter a valid integer. If you do not "cast" (this is the technical term) your string as an int for the purpose of testing withing your conditional logic, **this problem will be wrong** (e.g., if "1" > "2" is wrong, the logic must be if 1 > 2 -- you must compare numbers, not strings).

2. Next, let us look at ranges with conditional logic. Request a sales volume value from the user and display the appropriate message. I expect to see all 3 scenarios demonstrated in your output, or the problem will be considered incomplete.

   Expected output example (*these are just example values, you can change them to whatever you* *want – again, bold text is what you are asking or displaying to the user – do not make your text bold, I am just making it bold to make it obvious what the input question is*):

(scenario 1)
Please enter a value for sales volume to find out how it is classified: 20.0
If the value is 0 - 20 (inclusive of 20, but nothing larger than 20), display:
20.0 is classified as small sales volume

(scenario 2)
Please enter a value for sales volume to find out how it is classified: 59.9
For values 20 – 60 (anything larger than 20 and inclusive of 60, but nothing larger than 60), display:
59.9 is classified as medium sales volume

(scenario 3)
Please enter a value for sales volume to find out how it is classified: 61
For values larger than 60, display:
61.0 is classified as large sales volume

Demonstrate an example of **each** scenario (*small/medium/large*) and in your comments, and pay close attention to testing "edge cases" as the example in the hint suggests.

Hint:
Similar to how an **int()** function can be used, a **float()** function can be used to convert a sting like "60.1" to a float data type. You must "cast" your string values as float, or **this problem will be wrong**. **This will be true for all remaining problems and assignments in the course.** If you want your data to be treated as numeric data, it must be cast either as int or float, accordingly. **I will not give the hint to cast your data beyond this point.**

Also, as a best practice whenever using ranges in conditional logic, always test the edge cases (*meaning in the example above, which message should be applied when the value is 20 and 60 since the numbers that you will use will be listed in the comparison logic in 2 different conditional tests – since the directions indicate that 0-20 is inclusive of 20, it means that 20 should display "small sales volume" and not be included with "medium sales volume" – testing edge cases can help you identify logic problems in your code*)

3. Let us request some text from the user with an input function and store it in a variable. Specifically, ask if they would like to view sale volume data. If the answer is **yes** (or **YES**, or **Yes**, or **YeS**, or **yeS**), then display a message like "viewing sales volume data"

   Expected output example:
   Would you like to view sales volume data? yes *(is what the user types in)*
   You are viewing sales volume data
   Otherwise (think else), display:

Other data is currently not available

Hint:
Note, you do not need to consider any other input other than "yes" (or "YES" or "YeS" or "Yes" or "yeS"). If you only have 1 of 2 possible outcomes based on a single input, this will always dictate a simple if/else statement.

Also, always consider that when working with string data in the context of conditional logic, it may be advantageous to use the **.lower()** method of the string (*e.g., myStringVariable = myStringVariable***.lower()** *→ be sure to include the parentheses*) to **force** the data to lowercase (or **.upper()** for uppercase) to make a specific comparison, like if they enter "yes" or if the entered "YES" you always want to treat it like they typed in "yes" for sake of comparison, so you could force your string to lowercase just by doing something like **myText = myText.lower()**. Always be sure to test your input against possible unexpected variations, which upper and lower make it easy to know what to expect since you don't leave it up to chance.

In the next module, we will work with error handling to further improve dealing with unexpected variation in data and/or user input. For now, the .lower() (*or .upper()*) should be applied in any scenario where you are performing comparisons against string values that you are expecting user input like "yes" or "no" – otherwise, your logic **WILL** fail if you do not account for variation (e.g., Yes, YES).

4. Next, combine the previous 2 problems. Use the scale from problem 2 (0 <= 20, >20 and <= 60, >60) along with the conditional logic from problem 3 (e.g., "do you want to view sales volume data?"). If the user responds "yes" to question about viewing the sales volume data, next ask them to "enter a value for sales volume" and based on the value display the same messages that you did for problem 2.

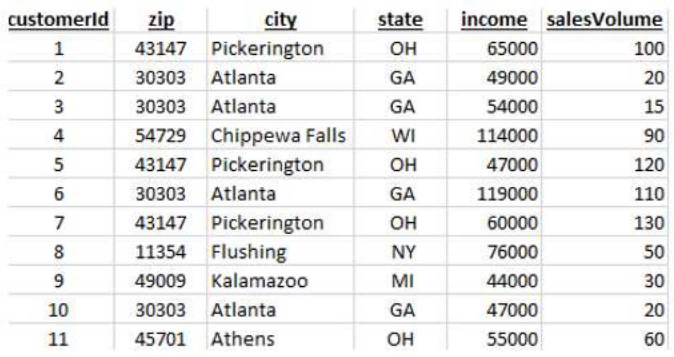
   Expected output example:
   (scenario 1)
   Would you like to view sales volume data? yes
   Please enter a value for sales volume: 20.0
   20.0 is classified as small sales volume

   (scenario 2)
   Would you like to view sales volume data? yes
   Please enter a value for sales volume: 60.1
   60.1 is classified as large sales volume

   (scenario 3)
   Would you like to view sales volume data? no
   Other data is currently not available

Demonstrate all 3 scenario output messages. **Also**, address what happens if a negative number is entered (*you decide how to handle it*) – this will be the 4th scenario to be demonstrated in your output. Failure to include examples of all 4 scenarios will result in the problem being incomplete.

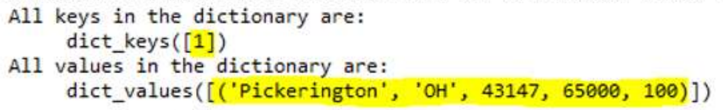
5. Here is the dataset (*this is the same data that you ended up with at the end of HW2 with the exception of the new "salesVolume" column*) that you will use that lists a customer ID along with the customer's partial shipping address, reported annual income, and associated sales volume for the customer:



customerId	zip	city	state	income	salesVolume
1	43147	Pickerington	OH	65000	100
2	30303	Atlanta	GA	49000	20
3	30303	Atlanta	GA	54000	15
4	54729	Chippewa Falls	WI	114000	90
5	43147	Pickerington	OH	47000	120
6	30303	Atlanta	GA	119000	110
7	43147	Pickerington	OH	60000	130
8	11354	Flushing	NY	76000	50
9	49009	Kalamazoo	MI	44000	30
10	30303	Atlanta	GA	47000	20
11	45701	Athens	OH	55000	60

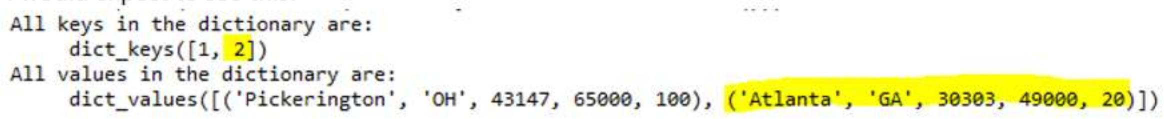
Note, similar to the previous homework, if you have worked ahead or have other programming experience, loops are ***not*** *permitted for this problem.*

We are going to pick up where we left off in homework 2 for the following problems (*it is OK to go back and re-use some code to get you started with problem 1*). Create a new tuple variable to store the customer data for customer ID 1. This time, however, include **city**, **state**, **zip**, **income**, and **salesVolume** in the tuple (*do not include the ID since that will become the key for the value in our dictionary*). Next, create a new empty dictionary. Recall that dictionary keys must be unique and immutable, where the value can be anything. For each of our customers, store the customer ID as the key, and for each key store the tuple as the value. For example, if we only store customer ID 1 and the city, state, zip, income, and salesVolume (as a tuple) for that key, and then output the keys and values for the dictionary, I would expect to see this:



```
All keys in the dictionary are:
     dict_keys([1])
All values in the dictionary are:
     dict_values([('Pickerington', 'OH', 43147, 65000, 100)])
```

Next, if I store the data for customer Id 2, and the tuple that contains the data for customer Id 2, I would expect to see this:



```
All keys in the dictionary are:
     dict_keys([1, 2])
All values in the dictionary are:
     dict_values([('Pickerington', 'OH', 43147, 65000, 100), ('Atlanta', 'GA', 30303, 49000, 20)])
```

and so on until we have all 11 customer IDs stored as keys with the respective tuple as the value which represents the correct data for each customer ID. If we display all keys and values for the dictionary, it should look like this:

```
All keys in the dictionary are:
     dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
All values in the dictionary are:
     dict_values([('Pickerington', 'OH', 43147, 65000, 100), ('Atlanta', 'GA', 30303, 49000, 20),
('Atlanta', 'GA', 30303, 54000, 15), ('Chippewa Falls', 'WI', 54729, 114000, 90), ('Pickerington',
'OH', 43147, 47000, 120), ('Atlanta', 'GA', 30303, 119000, 110), ('Pickerington', 'OH', 43147, 60000,
130), ('Flushing', 'NY', 11354, 76000, 50), ('Kalamazoo', 'MI', 49009, 44000, 30), ('Atlanta', 'GA',
30303, 47000, 20), ('Athens', 'OH', 45701, 55000, 60)])
```

which is a little tough to read, so let's only display data from the dictionary for key value 10:

```
('Atlanta', 'GA', 30303)
```

Finally, display the value for key 11, and then following that display all keys and all values.

Expected output example (*I will be flexible with formatting here, but I do expect all aspects of sentences and data correctly accounted for***)**:

```
The data associated with key 11 is: ('Athens', 'OH', 45701, 55000, 60)
All keys in the dictionary are:
     dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
All values in the dictionary are:
     dict_values([('Pickerington', 'OH', 43147, 65000, 100), ('Atlanta', 'GA', 30303, 49000, 20),
('Atlanta', 'GA', 30303, 54000, 15), ('Chippewa Falls', 'WI', 54729, 114000, 90), ('Pickerington',
'OH', 43147, 47000, 120), ('Atlanta', 'GA', 30303, 119000, 110), ('Pickerington', 'OH', 43147,
60000, 130), ('Flushing', 'NY', 11354, 76000, 50), ('Kalamazoo', 'MI', 49009, 44000, 30),
('Atlanta', 'GA', 30303, 47000, 20), ('Athens', 'OH', 45701, 55000, 60)])
```

Extra code to consider and Hint:
Please go back to the previous assignment to review any extra code listed there along with any hints. **Moving forward, I will not make this recommendation**; I will just **assume** that you know to take notes on any previous examples and can re-use them, as necessary. Think about what the general process sounded like: for example, create a series of list variables to store customerIds, zipCodes, etc.; next, create an empty dictionary, and then, using 3 lines of code construct the dictionary by changing whichIndexValue. Again, go back and read the hint for the last problem of the previous homework. You may even want to read back through all the problems from start to finish.

6. Demonstrate the function that is used to delete variables by removing the dictionary that was created in the previous problem.

Expected output example
For this problem, I cannot give the screenshot without giving the function away. What I expect to see highlighted in your code window will be the command that you just executed, and in the console will be the successful execution of the command AND the Variable Explorer window that shows me that your dictionary has been removed.

7. Let us go back to how you created your dictionary. I want you to define a new function (*something like createMyDictionary*) that encapsulates all the code used to create the dictionary (*including data*) from problem 6. Similar to how we made "DetermineAwardVersion2" function in the learning activity, but in this new function, **we will pass it no parameters** (*go back and re-listen to that part of the learning activity if you are unsure what I mean by parameters*).

   Since you have deleted your dictionary, once you create this function, you should be able to call it similar to how we called DetermineAwardVersion2 in the learning activity (*however, you will **not** pass it any data*). Recall that the function code must be executed prior to being able to call the function. For example, if I try to call the function prior to its execution, I will get an error like this:

```
NameError: name 'createMyDictionary' is not defined
```

   Next, once you have successfully created and executed your createMyDictionary function (*also, be sure to describe what the function does in the function's "**docstring**"*), we still have another problem, the function will create the dictionary; however, the dictionary will disappear as soon as the function's code is finished. **Look at the extra code to consider below.** What we need to do **OUTSIDE** of your new function (after the **return** line) is to create a new dictionary variable (*in the example below, I called it dictionaryWithCustomerData*) and **ASSIGN** it the output of your new function. Once you do that, display all of the keys and values for the dictionary just like we have done before (*just to be clear, you will do this **outside** of the createMyDictionary function and assign the output of that function to a new dictionary variable like you see here*).

   Expected output example

```
All keys in the dictionary are:
     dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
All values in the dictionary are:
     dict_values([('Pickerington', 'OH', 43147, 65000, 100), ('Atlanta', 'GA', 30303, 49000, 20),
('Atlanta', 'GA', 30303, 54000, 15), ('Chippewa Falls', 'WI', 54729, 114000, 90), ('Pickerington',
'OH', 43147, 47000, 120), ('Atlanta', 'GA', 30303, 119000, 110), ('Pickerington', 'OH', 43147, 60000,
130), ('Flushing', 'NY', 11354, 76000, 50), ('Kalamazoo', 'MI', 49009, 44000, 30), ('Atlanta', 'GA',
30303, 47000, 20), ('Athens', 'OH', 45701, 55000, 60)])
```

   Extra code to consider:
   When you "**return**" at the end of a function, if we want to return any data, for example, a dictionary or any other variable that is used within the function, all we have to do is type in the name of the variable **following** the word **return**. So, for example, if we wanted to not only create the dictionary, and populate it with all of the data (which is what the code copied from problem 6 does), once that dictionary is created in your function, we can **return** it by doing this on the **last line** in our function (*assuming the dictionary you create is called myDictionary*):

```
 return myDictionary
```

   Next, if our function returns a dictionary, we must assign that data to a variable outside the function. We have used the assignment operator many times (=), so how do we assign the data returned by a function to a variable (*in this case a variable that can hold a dictionary*) – like this:

```
dictionaryWithCustomerData = createMyDictionary()
```

   which just reads "*call the createMyDictionary function, don't pass it any parameters, and store the data that is returned by the createMyDictionary function in a new variable called* *dictionaryWithCustomer data*." The fact that your createMyDictionary function returns data that is in the format of a dictionary structure will dictate that dictionaryWithCustomer will become a dictionary with all the properties and methods that are available to a dictionary (like .keys() and .values()). Remember, use a type function to make sure your variable is what you think it is.

Hint:
No print functions will be used in your function – they will only occur outside of the function based on the variable that you create when calling your "createMyDictionary" function.

8. Now that you have a new dictionary containing all customerIDs as keys, and all customer data for each key stored as a tuple for the appropriate value, we will use that data to work with some conditional logic. Using an input function, ask the user to enter a customerID, if the customerID equals 10, display the value from the dictionary associated with **customerID 10**; otherwise, display a message that shows "The data is currently not available for other customers." This means that if they enter customerId 10, you see the correct associated data for 10, and if they enter ANY other integer value, it will display the "otherwise" message below.

   Expected output example:
   Enter a customerID: 10 *(assuming that 10 is entered)*
   ```
   The data associated with key 10 is: ('Atlanta', 'GA', 30303, 47000, 20)
   ```
   Otherwise, for all other integers entered, display:
   ```
   The data is currently not available for other customers
   ```

9. Let us look at the tuple data stored within the dictionary. We will ask the user to enter a customer ID. Based on the customer ID entered, we will query our dictionary and return an appropriate message based on the salesVolume that we have stored for the customer (*see the extra code to consider*). These messages will use values similar as before (*0 – 20 "small sales volume, 20-60 "medium sales volume", 60 and up "large sales volume"*)

   Expected output example:
   The output will look similar to this:

   Please enter a customerID: 1
   customerID 1 has large sales volume (*following the scale used previously and listed above*)

   Please enter a customerID: 2
   customerID 2 has small sales volume

   Please enter a customerID: 8
   customerID 8 has medium sales volume

   Demonstrate each scenario just like you have been doing in previous problems (so, you should have 3 examples in the output – small, medium, large), **AND**, address what happens if a negative number is entered (*this will be a 4th scenario – you decide how to handle it*). Failure to include examples of all 4 scenarios will result in the problem being incomplete.

Extra code to consider:
We have previously seen how you can look at data within a dictionary based on its key to see the associated value. For example, if we want to see the tuple associated with customerId 1, we will just write the dictionary name and the key like this:

```
dictionaryWithCustomerData[1]
('Pickerington', 'OH', 43147, 65000, 100)
```

Now, if we want to access data directly in that associated tuple, we can use a zero-based index value in the tuple. For example, if we want to get access to the zip code in the tuple, it is index value 2 within the tuple, so we could access the data directly in the tuple like this:

```
dictionaryWithCustomerData[1][2]
43147
```

Use that concept to help you access the sales volume data within the tuple for the customerID entered by the user.

10. Using your dictionary again, ask the user if they would like to view sales volume data, if so, ask for which customer ID, then display the sales volume data along with its classification (*like the previous problem*). You will also verify that the customer ID is valid and if it is not, display an appropriate message (*see the expected output and the hint*).

    Expected output example:

    Would you like to view sales volume data? yes
    Please enter a customerId: 1
    Sales volume data for customerId 1 is 100 and is classified as large sales volume *(scenario 1)*

    (do the same as scenario 1, but pick a customer ID that is classified as medium, and then again for small – these will be scenario 2 and scenario 3)

    Would you like to view sales volume data? yes
    Please enter a customerId: 22
    CustomerId 22 is not valid *(scenario 4)*

    Would you like to view sales volume data? no
    Other data is currently not available *(scenario 5)*

    Demonstrate each of the 5 scenarios (*so, small, medium, large, along with the choice when they enter "no" to view sales volume data, and also address when they enter a customerId that is either smaller or larger than the existing customerIDs*).

    Hint:
    We have previously looked at min and max functions. Consider how they can be used on your

dictionary’s keys to help you determine if the customer ID is valid or not (e.g., enteredId >= min and enteredId <= max).