**Introduction to Programming with Python**

**Homework 1**

***Due on July 12, 2020***

1. **Arithmetic, Strings, Lists, Loops, and Formatting (100 points)**
2. Create a new, empty Python code file named **hw1.1.py**. You may use IDLE, Spyder, or some other Python Integrated Development Environment (IDE). In the lecture, we saw how to compute a Fahrenheit-to-Celsius temperature conversion, and how to display output like:

80 degrees Fahrenheit is 26.666666666666668 Celsius

Add code in **hw1.1.py** to perform this computation and display the same kind of

output for Fahrenheit temperatures 0, 32, 45, 60, 70, 95, and 212. (Remember

that when you make changes to a Python code file in IDLE, you must save the

file first [which you can do in Windows by pressing Ctrl-S] before you can run

the modified program [which you can do in Windows by pressing F5].)

1. Add code to **hw1.1.py** to perform Celsius-to-Fahrenheit temperature conversion, and display output for Celsius temperatures 0, 10, 15, 20, 25, 30, and 100.
2. Here is a table of predicted high temperature (Fahrenheit), low temperature, and percent humidity in Pittsburgh for 15 days, starting on Aug. 30, 2018:

Date Hi(F) Lo(F) %Hum

AUG 30 79 64 60

AUG 31 82 66 20

SEP 1 85 68 20

SEP 2 85 68 30

SEP 3 87 69 30

SEP 4 87 69 20

SEP 5 87 69 20

SEP 6 85 67 20

SEP 7 81 63 30

SEP 8 79 61 20

SEP 9 77 61 20

SEP 10 77 59 20

SEP 11 77 59 20

SEP 12 77 59 20

SEP 13 77 59 50

You can copy-and-paste this table into your **hw1.1.py** file. Since this table is *not* valid Python code, you can “comment it out” by enclosing the entire table within triple quotes:

**"""**

Date Hi(F) Lo(F) %Hum

**...**

SEP 13 77 59 50

**"""**

After you have copied-and-pasted the table and “commented it out” by using the

triple quotes, save and run your program to make sure the commented-out table

does not cause an error.

1. Define a **list** named **dt** containing the dates as strings (**str**):

**dt = ['AUG 30', ..., 'SEP 13']**

(At this point, we are *not* going to try to create **dt** directly out of the triple-quoted

string containing the table. Just define **dt** manually, as illustrated above.)

Define a **list** named **hi** containing the high temperatures, a **list** named **lo**

containing the low temperatures, and a **list** named **hm** containing the humidities.

Then, use a **for** loop to display a copy of the table to the user’s screen, like this:

**print('Date Hi(F) Lo(F) %Hum')**

**num\_days = len(dt)**

**for i in range(num\_days):**

**print(dt[i], hi[i], lo[i], hm[i])**

Save and test. Does the displayed table look exactly like the original table?

1. You can add spaces between fields of output in a **print()** function call by displaying strings of spaces! For example, change the **print()** function call to:

**print(dt[i], ' ', hi[i], lo[i], hm[i])**

You will see that this adds three spaces between the end of **dt[i]** and the beginning of **hi[i]**. (Why *three* spaces? Because the string contains two spaces, and the comma after the string causes another space to be displayed.)

Revise your **print()** function call so that the displayed columns line up exactly as in the original table. Save and test.

1. You can compute the sum of the values in a **list** of numbers using a loop, like this:

**m = [1, 2, 3, 5]**

**tot = 0**

**for val in m:**

**tot += val**

**print('Sum of values in', m, 'is', tot)**

Output:

Sum of values in [1, 2, 3, 5] is 11

Or, you can use Python’s built-in **sum()** function:

**m = [1, 2, 3, 5]**

**tot = sum(m)**

**print('Sum of values in', m, 'is', tot)**

Output:

Sum of values in [1, 2, 3, 5] is 11

After the display of the table, add code to compute and display the *mean*, *median*, and *sample standard deviation* of high temperatures, low temperatures, and humidities for the 15 dates in the table. *Hint:* How can you put the values of a **list** in ascending order, so that the middle value is the median? You will *not* want to do this on the original lists of temperatures or humidities: make a *copy* of a **list**, and then modify the copy:

**hi\_copy = hi.copy()**

Now you can safely modify **hi\_copy** without modifying **hi**.

Recall that the sample standard deviation for **N** observations of some value **x***i* is:



1. In part (e), making your displayed table look like the original table by using strings of spaces between the data, high temp, low temp, and humidity really works because of luck: the temperature and humidity values were all 2 digits wide. What if some were 3 digits wide and some were just 1 digit wide? Consider these (made up) 15 days in Death Valley, California:

Date Hi(F) Lo(F) %Hum

AUG 30 107 78 15

AUG 31 109 80 8

SEP 1 104 79 12

SEP 2 101 78 13

SEP 3 99 77 9

SEP 4 98 78 8

SEP 5 95 78 8

SEP 6 98 78 10

SEP 7 98 77 10

SEP 8 102 80 10

SEP 9 104 81 12

SEP 10 101 80 8

SEP 11 99 78 15

SEP 12 96 77 15

SEP 13 99 78 12

Now we will need a more precise way of formatting.

Python 3.7 provides several generations of string formatting facilities. Here we will work with the **str** type’s **format()** method, which allows us to control field widths, left-, center-, and right-justification, digits of precision displayed for floating-point numbers, and so forth.

Read the very nice introduction to the **format()** method at:

<https://www.digitalocean.com/community/tutorials/how-to-use-string-formatters-in-python-3>

and try the examples given there. If we had our Death Valley table’s columns stored in lists **dt2**, **hi2**, **lo2**, and **hm2**, we could reproduce the table using **format()** as follows:

**print('{:<6s}{:>7s}{:>7s}{:>6s}'.format(**

**'Date', 'Hi(F)', 'Lo(F)', '%Hum'))**

**num\_days = len(dt2)**

**for i in range(num\_days):**

**print('{:<6s}{:>7d}{:>7d}{:>6d}'.format(**

**dt2[i], hi2[i], lo2[i], hm2[i]))**

Create the lists **dt2**, **hi2**, **lo2**, and **hm2** from the Death Valley data, and confirm

that the above code works for accurately reproducing the table.

1. Now, use the Fahrenheit-to-Celsius conversion formula to produce the table in Celsius rather than Fahrenheit. Display 2 digits after the decimal point in the Celcius temperatures. For example,

**print('[{:>7.2f}]'.format(26.666666666666668))**

will display:

**[ 26.67]**

You will need to widen the field widths for the second and third columns. The first couple of lines of the Celsius table should look like:

Date Hi(C) Lo(C) %Hum

AUG 30 41.67 25.56 15

1. In the Python 3.7 documentation:

https://docs.python.org/3.7/library/string.html#formatstrings

read section **Format Specification Mini-Language** and try out the examples in the **Format Examples** section that follows. (You do not need to include these Format Examples in your homework code file.)

1. You can use *nested* **for** loops, that is, one **for** loop within another, like this:

**for** *var1* **in** *iterable1***:**

**for** *var2* **in** *iterable2***:**

*stmt*

*...*

Create a nicely formatted table with 20 rows of 5 columns, where the rows are for the integer values 0 through 19, and the columns are the cube root, square root, integer value, square, and cube of each integer values. Not all of the cube roots or square roots will be integers, of course, so display all of these as floating point values with a precision of 6 digits after the decimal place. Arrange for all five columns to be of equal width.

***REMEMBER*** to put all authors’ names into your source code file.Upload your **hw1.1.py** file to Canvas for this assignment.