ENGR 14100
Python 3 PA

Individual Assignment: See the course syllabus for a definition of what this constitutes.

Task 1 (of 2)

Objectives: Predict the output of a complete for and while loop in Python; Use for loops to conduct repetitive operations in Python; Use loops to create corresponding data sets (i.e. lists, arrays, and dictionaries) in Python; Create, manipulate, and read from arrays, lists, and dictionaries in Python; Manipulate and extract information from lists, arrays, and dictionaries in Python; Use loops to compare and evaluate associated elements in data sets in Python.

Background:
Algorithms are used in a wide range of engineering and science disciplines. They are generic step-by-step lists of instructions for solving a problem. One important task for engineers to solve algorithmically is sorting numbers. Most of the time engineers are dealing with thousands or millions of data points that must be sorted before they can be analyzed. In this context, an algorithm that conserves memory space and/or allows for sorting to be done in a timely, efficient manner is desirable.

The simplest way of sorting a group of items is to determine the smallest item in the group and then swap its position with the item that is currently first in the list. Then, starting with the item in the second position, each item in the list is compared to the second item. If there is a value that is smaller than that in the second position, the two are swapped, and the process starts over. This process continues until all values in the list have been compared to each other and put in the appropriate order.

An example of this sort can be seen below:

Input :
`4 7 9 3 1`

`[4 7 9 3 1]` ⟶ Every value in the list after 4 is compared with 4, starting with 7. 3 is smaller than 4.
`[3 7 9 4 1]` ⟶ Every value in the list after 3 is compared with 3, starting with 7. 1 is smaller than 3.
`[1 7 9 4 3]` ⟶ Every value in the list after 1 is compared with 1, starting with 7. No values are smaller than 1.
`[1 7 9 4 3]` ⟶ Every value in the list after 7 is compared with 7, starting with 9. 4 is smaller than 7.
`[1 4 9 7 3]` ⟶ Every value in the list after 4 is compared with 4, starting with 9. 3 is smaller than 4.
`[1 3 9 7 4]` ⟶ Every value in the list after 3 is compared with 3, starting with 9. No values are smaller than 3.
`[1 3 9 7 4]` ⟶ Every value in the list after 9 is compared with 9, starting with 7. 7 is smaller than 9.
`[1 3 7 9 4]` ⟶ Every value in the list after 7 is compared with 7, starting with 9. 4 is smaller than 7.
`[1 3 4 9 7]` ⟶ Every value in the list after 4 is compared with 4, starting with 9. No values are smaller than 4.
`[1 3 4 9 7]` ⟶ Every value in the list after 9 is compared with 9, starting with 7. 7 is smaller than 9.
`[1 3 4 7 9]` ⟶ The entire list has been sorted.

This algorithm is noted for its simplicity and is useful where swapping is very expensive. It has a performance advantage in optimizing the use of more modern/complex algorithms which, for example, require input data to be in sorted lists to produce human-readable output.

As can be seen from the example above, this sort consists of swapping elements in an array where no additional memory is required to do the sorting. This means you are required to use the original array where you swap the values, and cannot just create a new array and store the values in it. (Using a temporary swapping variable, however, is OK.) This also must be done without using any built in Python sorting algorithms or methods.

Part A:
Create a flow diagram of the sorting algorithm described above.

Save your file in a PDF called:

`Py3_PA_login.pdf`

Part B:
Translate your previously created flow diagram into a Python program. Your program should prompt the user for a list of numbers to be entered on a single line, where each number is separated by spaces. Next, it should pass the list of numbers to a function called `list_sort`, where the list will be sorted. Once sorted, it should pass back the new list to the main program and print the list in order from the smallest number to the largest. Note that this function should NOT be a separate module.

Save your file as:

`Py3_PA_Task1_`*`login`*`.py`.

Task 1 File:

1) `Py3_PA_`*`login`*`.pdf`
2) `Py3_PA_Task1_`*`login`*`.py`

Task 2 (of 2)

Objectives: Predict the output of a complete for and while loop in Python; Use while loops to conduct repetitive operations to perform numerical operations in Python.

Background:
In order to develop and further improve processes, engineers must establish the relationship between the variables in the process. One common practice is to plot the variables on different scales, including semi-logarithmic and logarithmic scales, in order to determine which plot yields a linear relationship. A line of best fit is then drawn to establish the relationship between the variables in the form:

$$f(x) = a_1 x + a_2$$

Instructions:
Given the coefficients a_1 and a_2, a goal value y, a step size d, the initial number of allowed steps m, a maximum error e, and a maximum number of step size reductions t, your program should determine the value of x in the linear equation

$$f(x) = a_1 x + a_2$$

for which the value of x is such that the magnitude of the error,

$$|y - f(x)|$$

is less than or equal to e **or if the number of allowed steps is exceeded.** If the number of allowed steps is exceeded, the process should be restarted with a step size equal to one half the previous step size, and the number of allowed steps should be doubled. The step size can be reduced up to t times, after which your program should exit with a useful error message. The program should first test the value of x = 0 and then increment by *d* as needed.

You will ask the user to input (in this order):

1. The coefficients *a1* and *a2*
2. The goal value y
3. The step size *d*
4. The maximum error *e*
5. The maximum number of iterations *m*
6. The maximum number of times *t* the step size can be decreased if a solution is not found within the limit of iterations

Note that each item in the above list corresponds to one line of input. The user should not be asked to enter any other values, and the program should run to termination after these values are input. After running, your program should output, to the millionth decimal place:

1. The estimated value of x
2. The magnitude error, | $y - f(x)$ |
3. The **total** number of iterations required to calculate the solution
4. The number of times that the step size was reduced

Develop a flowchart to represent the algorithm used to solve this problem. Save in the previously created PDF.

Example:
Please note that the actual computation results may not be correct. This example is intended to elucidate matters of format only (inputs bold).

```
Enter a1 and a2: 1.0 0.0
Enter y: 23.0
Enter d: 0.1
Enter e: 0.1
Enter m: 250
Enter t: 0
The algorithm converges to the following values:
Estimated Solution: 22.9
Magnitude error: 0.1
Total number of iterations: 230
Number of times step size was reduced: 0
```

Save your file as:

`Py3_PA_Task2_`*`login`*`.py`.

Task 2 Files:

1) `Py3_PA_`*`login`*`.pdf`
2) `Py3_PA_Task2_`*`login`*`.py`

Make sure you include your INDIVIDUAL HEADER in your file(s) and submit those files to the appropriate turn-in box on Blackboard.