Fall 2020
Due Oct 26th

Computer Science 1001
Problem Set #5

Section 001/002

Important Notes:

- **This problem set is optional. If you choose to not submit a solution, the problem set portion of the evaluation will include one less problem set. If you choose to complete the problem set, you should submit a solution to ONE of the TWO questions.**
- Even though this, or any other subsequent, Problem Set is only graded based on whether or not a reasonable attempt was made for each problem, it is a good idea to complete each Problem Set by the due date and upload it to the appropriate dropbox in the course Brightspace shell to get credit for completing the Problem Set and see if you have understood the required concepts.
- While working on your Problem Set you can get help by posting your questions in the appropriate folder in the discussion forum. Note that sometimes, even though your code can generate the expected output, it may still not be correct as it may work for a specific data set and not for all valid data sets. Also, it may not use the programming concepts and best practices that we have emphasized in the course.
- Completing the Problem Sets will help you to gain hands-on experience with coding in Python and understand the learnt concepts well enough so that you can apply the concepts to solve and code the solution for the given problems. All of this will help you in doing well on your tests and exam.
- The files that you upload to the dropbox should be your source code (`.py`) files, as practiced in Lab 0, and any other requested solution files.
- While coding solutions for the problems given below, keep in mind that on the test/exam you will also be marked on the following:
  - Efficient solution of the problem. A given problem can be solved in a number of different ways, but the best solution is the one that is efficient; ie., the one that uses the right concepts in a very productive way.
  - Including sufficient descriptive comments in your program. The person looking at your code should be able to understand how your code is solving the given problem even if the person reading your Python program does not know the Python language. In addition, the reader of your program should be able to understand what each variable represents.
  - Labelling of input and output. All input and output should have a descriptive label so that the reader of your program understands what input is expected and what output the program has generated.
  - Program style - consistent formatting and indentation of program statements, meaningful variable names (identifiers) and the use of constants (constant identifiers), where appropriate.

Practicing these rules will build a good foundation for programming.

- This Problem Set is based on Chapter 12 of the textbook, without the graphics components. Please use only concepts from Chapters 1–6 and 12 of the textbook.
- Rubrics/solution outlines for each Problem Set will be provided after the grades for the Problem Set have been released.

Full solutions will not be posted, however you may get help to complete your Problem Set if the rubrics/solution outlines are insufficient.

---

1. Write a Python program, in a file called `sortList.py`, which, given a list of names, sorts the names into alphabetical order. Use a one dimensional array to hold the list of names. To do the sorting use a simple sorting algorithm that repeatedly takes an element from the unsorted list and puts it in alphabetical order within the same list. Initially the entire list is unsorted. As each element is placed in alphabetical order, the elements in the portion where the data is in alphabetical order is referred to as the **sorted portion** of the list and the rest of the list is referred to as the **unsorted portion**. The process of moving an element from the unsorted portion of the list to the sorted portion of the list is repeated until the entire list is sorted.

   Include the following functions in your solution:

   - `findNext` which, given the list and the index of the first element in the unsorted portion, finds and returns the index of the lowest string (alphabetically) in the unsorted portion of the list.
   - `putInOrder` which, given the list, the index of the first element in the unsorted portion, and the index of the next name to be placed in order (index returned by the `findNext` function), swaps the names at the two indices (ie. swaps the name at the beginning of the unsorted portion with the lowest, alphabetically, name)
   - a `main` function that initializes the unsorted list, prints the unsorted list, uses the functions above to sort the list, and then prints the sorted list.

   For example, given the following list of names to start,

   ```
   ["Zita", "Henny", "Benny", "Harold", "Danny", "Penny"]
   ```

   the output should be:

   ```
   Unsorted list:
   ['Zita', 'Henny', 'Benny', 'Harold', 'Danny', 'Penny']
   Sorted list:
   ['Benny', 'Danny', 'Harold', 'Henny', 'Penny', 'Zita']
   ```

   For testing, you do not necessarily need to use user input to obtain the original list, you may include the list in your `main` function as a literal if you wish.

2. The formula for the distance between 2 points (x_1, y_1, z_1) and (x_2, y_2, z_2) is given by

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

Write a Python program, in a file called `distance.py`, to find the distance between all pairs of points in a two-dimensional list such as the following:

```
points = [[4,2,1],[-1,3,5],[6,9,-2],[8,-1,5]]
```

Output the coordinates of the two points that are closest to each other along with the distance between the points. Using insertion sort, output the distances between points in descending order. Your solution should include a function `distance(a,b)`, where `a` and `b` are points represented by lists of (x, y, z) coordinates (ex. `a=[4,2,1]`), and a function to implement the insertion sort. Your sorting function should not use any built-in Python functions (except for `len`).

Sample input/output (for the `points` list above):

```
Sorted distances:
12.37
11.58
 9.85
 7.87
 6.48
 6.40
Closest points were:
[4, 2, 1] and [8, -1, 5] with distance = 6.40
```