COMP 1001 (Fall 2024): Practice Problem Set 5

Suggested Completion Date: Nov 5, 2024

Important Notes:

- Practice Problem Sets are not a part of the evaluation for the course.
- After the Suggested Completion Date, a rubric and solution outline will be posted on the course Brightspace shell so you can check your answers.
- Completing the Problem Sets will help you to gain hands-on experience with coding in Python, and will help you apply the concepts we have covered to solve and code solutions for the given problems. This will help you prepare for your tests and exam.
- 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; i.e., 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 9 and 12, without the graphics components. Please use only concepts from Chapters 1–6, 9, 12 of the textbook.

1. Write a Python program, in a file called `sort_list.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:

   - `find_next` 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.
   - `put_in_order` 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 `find_next` 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. (a) In a file called `music.py`, create a `Song` class that contains the following:
   - The private data fields `name`, `minutes`, `seconds` for the name of a song and its length in minutes and seconds
   - A constructor with arguments for `name`, `time` (both strings). The `time` string shoud be in the format `mm:ss`.
   - Accessor methods for `name`, `minutes`, and `seconds`.

(b) Add to your `music.py` file to create an `Album` class that contains the following:

- The private data fields `title`, `artist`, and `date` (all strings) for the title of the album, the name of the artist, and the release date (year), as well as a list of `Song`s (in data field `songs`).
- A constructor with arguments for `title`, `artist`, and `date`.
- Accessor methods for `title`, `artist`, `date`, and `songs`.
- A method to add a `Song` to the `Album`.
- A method to calculate the total length (time) of the `Album` in the format `hh:mm:ss`.

(c) Write a `main` function, in `music.py`, to illustrate the use of your `Album` and `Song` classes.

Sample input/output:

```
Enter name of album: Album1
Enter the name of the artist: Artist1
Enter the release date: 2018
How many songs? 3
Enter name of song: Song1
Enter the length of the song (mm:ss): 03:43
Enter name of song: Song2
Enter the length of the song (mm:ss): 04:02
Enter name of song: Song3
Enter the length of the song (mm:ss): 02:58
Album length (hh:mm:ss) is 00:10:43
```