FIT1045 Algorithmic Problem Solving – Assignment (15%).

Due: Midnight, Friday 23rd September, 2016.

Objectives

The **objectives of this assignment** are:

- To demonstrate the ability to implement algorithms using basic data structures and operations on them.
- To gain experience in designing an algorithm for a given problem description and implementing that algorithm in Python.

Submission Procedure

1. Put you name and student ID as a comment on each file.
2. Save your files into a single folder and create a zip file of this folder called yourFirstName_yourLastName.zip
3. Submit your zip file containing your solution to Moodle. Make sure that your assignment is not in "Draft" mode. You need to click "Submit" to successfully submit the assignment.
4. Your assignment will not be accepted unless it is a readable zip file.

Important Notes:

1. Please ensure that you have read and understood the university's policies on plagiarism and collusion available at `http://www.monash.edu.au/students/policies/academic-integrity.html`. You will be required to agree to these policies when you submit your assignment. The assignments will be checked for plagiarism using an advanced plagiarism detector.
2. You must not use the built-in functions to sort (e.g., `sort()` or `sorted()`), search (e.g., `x in list`) or to find the maximum/minimum element in the list (e.g., `max()` or `min()`): please write your own Python code for these if required. You are not allowed to import any Python library or module.
3. Your program will be checked against a number of test cases. Do not forget to include comments in your code explaining your algorithm. If your implementations have bugs, you may still get some marks based on how close your algorithm is to the correct algorithm.
4. For each task, you need to write a program that properly decomposes the problem. You will learn functions and decomposition in Week 6.

Marks: This assignment has a total of 30 marks and contributes to 15% of your final mark. Late submission will have 10% off the total assignment marks per day (including weekends) deducted from your assignment mark, i.e., 3 marks per day. So, if you are 1 day late, you will lose 3 marks. Assignments submitted 7 days after the due date will normally not be accepted.

Marking Guide:

Task 1: 8 marks

(a) Code readability (Non-trivial comments where necessary and meaningful variable names) – 1 mark

(b) Code decomposition – 1 mark

(c) Correctly determining the minimum and maximum score given by a judge to a participant – 2 marks

(d) Final scores correctly computed for each contestant – 2 marks

(e) Creating the "Scoreboard" table and displaying the final score for each participant – 2 marks

Task 2: 10 marks

(a) Code readability (Non-trivial comments where necessary and meaningful variable names) – 1 marks

(b) Code decomposition – 1 mark

(c) Correctly determining if a palindrome can be formed using all letters or not – 2 marks

(d) Correctly determining the lexicographically smallest palindrome – 6 marks

Task 3: 12 marks

(a) Code readability (Non-trivial comments where necessary and meaningful variable names) – 1 mark

(b) Proper decomposition of the problem – 1 mark

(c) Correctly writing `isGood(s1,s2)` function – 2 marks

(d) Correctly writing `isAlmostGood(s1,s2)` function – 6 marks

(e) Correctly determining if the string is a bad string using – 2 marks

Task 1: Australian Idol 8 Marks

You are hired by Australian Idol to write a program to create a scoreboard for them. Each participant is judged by 5 different judges where each judge gives a score between 0 to 10. You are given an input file named `raw_scores.txt` which contains 10 lines (one for each participant). Each line contains 6 strings separated by spaces. The first string in each line is the name of the participant. The remaining 5 strings record the scores given by the 5 judges to the participant. Below is an example showing scores for three participants Mark, Alice and Alan.

```
Mark 1 10 0 2 9
Alice 8 10 8 0 10
Alan 10 10 10 10 10
```

The final score of each participant is computed as follows. To handle potentially biased judges, the highest score and lowest score received for the participant are ignored and the average of remaining three scores is the final score. E.g., for Mark, the highest score 10 and the lowest score 0 are ignored and his final score is $\frac{1+2+9}{3} = 4$. Alice's final score is $\frac{8+8+10}{3} = 8.67$ (ignoring lowest score 0 and one of the two 10 scores) and Alan's final score is $\frac{10+10+10}{3} = 10$ (ignoring the lowest score 10 and the highest score 10). Note that the final score is rounded to two decimal points (e.g., Alice's score is rounded to 8.67). You can use `round()` function `https://docs.python.org/3/library/functions.html#round`.

Your program should read the `raw_scores.txt` file and compute the final score for each participant. The results must be stored in a table called "Scoreboard" that consists of two columns recording the names of the participants and their final scores. Your program must also print the final score for each participant. Below is a sample output for the above input.

```
Mark: 4
Alice: 8.67
Alan: 10
```

Important: Your program will be checked on an input file that will contain data different than the data provided in `raw_scores.txt` file. Therefore, do not tweak your program to work only for the provided file (e.g., by manually computing scores). However, you can assume that the input file will contain exactly 10 participants and each participant will be judged by exactly 5 judges. You can also assume that the name of the input file will be the same, i.e., `raw_scores.txt`.

Task 2: Palindromes 10 Marks

You are given a sequence of letters (all lowercase). Your goal is to rearrange the letters to form a palindrome. The palindrome must contain **all** the given letters. The palindrome does not need to be an English word, i.e., it is any sequence of letters that is the same when reversed. If there is more than one palindromes that can be formed using all of the given letters, your program must output the lexicographically smallest palindrome string (i.e., the palindrome that appears first in the alphabetical order `https://en.wikipedia.org/wiki/`

`Alphabetical_order`). For example, if the input sequence of letters is `x y x z y`, then `xyzyx` and `yxzxy` both are palindrome strings formed using all the letters in the sequence. However, `xyzyx` is the lexicographically smallest. Thus, the output must be `xyzyx`. If no palindrome can be formed by the letters in the input sequence then you must output `impossible`. For example, if the input sequence is `a l g o r i t h m`, your program must output `impossible`.

Your program must read from the input file named `letters.txt` that contains a single line containing a sequence of letters separated by spaces. Below is a sample input file.

```
x y x z y
```

There may be up to 1000 letters in the input line. Your program must create a list containing all these letters. Then, the program must print the lexicographically smallest palindrome that can be formed by the letters in the list. If no palindrome can be formed using all these letters, your program must output `impossible`

Task 3: (Almost) Good Strings 12 Marks

Suppose you are given two strings named `s1` and `s2` (containing only lowercase letters). The string `s1` is called a "good" string if it contains `s2`, i.e., `s2` is a substring of `s1`. For example, if `s1` is `smart` and `s2` is `art`, then `s1` is a good string because `smart` contains `art`. The string `s1` is called "almost good" if it is not a "good" string but inserting one character into it would make it a "good" string. For example, if `s1` is `smart` and `s2` is `mark`, then `s1` is not a good string because `mark` is not a substring of `smart`. But `s1` is an "almost good" string because inserting the letter `k` in `s1` (between `r` and `t`) makes it `smarkt` which is a "good" string because it contains `mark`. The string `s1` is called a "bad" string if it is neither a "good" nor an "almost good" string.

You need to write two functions called `isGood(s1, s2)` and `isAlmostGood(s1, s2)` that return `true` if and only if `s1` is "good" and "almost good", respectively.

Your program must ask the user to enter two input strings: first input string will be `s1` and the second input string will be `s2`. Both strings must be converted to lowercase letters. Then, the program must call the two functions to determine whether `s1` is a "good", "almost good" or "bad" string. The result must be printed.