YSC2221 PE 2018

You can use the package numpy in this PE and other built-in functions, e.g. "`sort()`". (And probably you do not need other packages.) Problem 1c is independent of 1a and 1b, meaning, you could start Problem 1c without doing 1a and 1b.

Problem 1a Print Total Scores (20 marks)

Each of the students in class has a name and three scores for English, math and science. Write a function "`print_total_scores(names,Eng_s,Math_s,Sci_s)`" to **print** their total scores, in which, names, Eng_s, Math_s, Sci_s are lists of strings and integers. You can see the example lists in the skeleton file. When you call the functions, here is the sample output:

```
>>> print_total_scores(names, Eng_scores, Math_scores, Sci_scores)
Peter 201
John 211
Mary 164
Paul 216
Richard 289
Anne 212
Zoe 211
Carl 289
Kelvin 206
```

Problem 1b Print Total Scores Sorted by Names (20 marks)

Similar to Problem 1a. However, your function "`print_total_scores_sorted()`" should **print** the total score list sorted in an ascending order by names:

```
>>> print_total_scores_sorted(names, Eng_scores, Math_scores, Sci_scores)
Anne 212
Carl 289
John 211
Kelvin 206
Mary 164
Paul 216
Peter 201
Richard 289
Zoe 211
```

Problem 1c List Out the Best Student(s) (20 marks)

Write a function "`show_max_scores(names, Eng_s, Math_s, Sci_s)`" to **print** out the student(s) with the maximum total score. For example, if you call the function with the given lists, you have:

```
>>> show_max_scores(names, Eng_scores, Math_scores, Sci_scores)
Richard has the maximum score of 289
Carl has the maximum score of 289
```

Problem 2 "Fly Pairs to The Moon" (40 marks)

SpaceX finally makes tourism to the Moon possible! Each spaceship can take exactly two persons to go to the Moon and return to Earth. However, each spaceship is designed in a way that it can only take EXACTLY a total weight of N kg. And there are a list of candidates and we store all of their weights in a list. Interestingly, no one has the same weight as the others. You job is to list out every **pair** of the candidates whose weights are added up to exactly N from the list. For the weight list in the skeleton file, you can run the function "`searchCouple(L,N)`" that takes in a list of candidates L and the desire total weight N kg, and it will **return** a list of tuples. Each tuple in the output is the two candidates with a total weight exactly N kg. For example:

```
>>> print(searchCouple(weights,150))
[(70, 80), (60, 90), (73, 77), (59, 91)]
>>> print(searchCouple(weights,152))
[(75, 77), (59, 93)]
```

Every number in the output must be unique, i.e. each weight appears in the output only once. You will still get partial credits if your list is not unique. If there is no couple with a total weight of N kg, the function will return an empty list. One last thing, each tuple has to be sorted in an ascending order.