Coursework Assignment Specification

(40 % of the module assessment)

**Squash League Management**

Submission Deadline: 3pm, 20th May 2016.

# Problem Description

Your task is to write a program which could be used to maintain the University staff squash league. The squash league is divided into six divisions, the highest division being division one. The number of players in a division is normally six. At the beginning of a ”round” (which lasts a month) the league organiser emails all the players telling them who is in what division and posts a notice on the squash noticeboards where players can write in the points to be awarded in each match; an example notice is available from the module web page. The players in a division then contact each other and play one match against each other; a match consists of up to five games and the winner is the first to win three games; league points are awarded as described in the notice.

When a match has finished, the players write the result on the noticeboard. Unfortunately, some matches do not get played. Furthermore, after a round has started, someone wishing to join a division can simply write his/her name in (on the division listing on the noticeboard) and play matches against the other members of the division.

After the finishing date for a round, the league organiser removes the notice from the noticeboard and totals up the points for each player and, according to the points gained, produces a new set of divisions for the following round. Ideally this is a straightforward matter: in each division, the top two are supposed to be promoted and the bottom two are supposed to be relegated (although, of course, there is no promotion from division one and no relegation from the bottom division). In practice, it’s more complicated than that: some players need to be removed (having given notice that they will be away or having apparently disappeared), some need to be added (normally, but not always, to the bottom division), and the organiser is free to promote and relegate fewer than the normal ration of two players (e.g. sometimes players ask not to be promoted).

# Your Task

Your aim is to write a program to make it as easy as possible for the league organiser to perform his job. The program should:

* 1. Maintain a text file, called players.txt, containing at least the following information about each player:
     + Forename
     + Surname
     + Email address
     + Phone number
     + Division in current round
     + Points in current round
     + Division in previous round
     + Points in previous round

To indicate a player who is unavailable for the current round it may be useful to set the division number to a nonsensical value, e.g. 0 for a player who is unavailable for the current round but available for the next, -1 for a player who is away until further notice.

The text file should be read in automatically when the program starts, the data then being held in a list in memory which can be manipulated by the program. The user should be able to save to disk (i.e. the text file) at any stage, and should be invited to do so on exit from the program if the file is not up to date.

* 1. Permit the maintainer to add new players, delete players, and edit the information for any player.
  2. Enable the maintainer to produce all the necessary listings at the start of a round:
     + the listing to be put on the noticeboard,
     + the list of email addresses of all of the players,
     + a list of the players in each division and their contact details (suitable for emailing so that players can start contacting each other to arrange matches).
  3. Provide a tool to help the maintainer perform the routine task of entering the points that each player has obtained in the current round.
  4. Provide a tool to help the maintainer roll the divisions forward to the next round.

The program should be as flexible and easy to use as possible, requiring as little as possible in the way of effort from the user, even when they make a mistake.

# How to Structure Your Program

The list of players should be a global variable in a module called **players.py**. All functions relating to modifying this list should be placed in this file and test code should be placed at the bottom; the test code should run when the file is run, but not when it is imported.

Another file called **squash.py** should contain all the code for the user interface and should import players.py. Running squash.py should produce a menu of options, enabling the league organiser to get his job done in the most efficient way possible.

The information for each player can be stored as any of the following: (i) a list of values (as in the marklist.py program), (ii) a dictionary of key:value pairs, or

(iii) an object of a Player class.

# What to Submit

In addition to the files mentioned you may write a short text file called README. This is to provide any special instructions or warnings to the user (or assessor!), or to draw attention to any aspects of the program that you are particularly proud of (please don’t waste time by writing an excessive amount).

All the files (including sample data files) should be compressed into a zip file and submitted electronically as directed.

# Notes on Expectations:

You will be marked according to your overall achievement, marked according to the Assessment Matrix that will be provided to you separately from this document. However below follows a qualitative description of some general expectation that may help you understand the general level of expectation associated with this piece of coursework.

**Technical mastery of Python** Your programs should show mastery of what you have been taught.

**Design** Your programs should be well structured for the task in hand so that it is as easy as possible for:

* a user to use the program for any likely purpose,
* a programmer to understand the code structure and be able to develop it further,
* a programmer to be able to re-use as much as possible of the code in a related application.

**Clarity and Self-Documentation** Given the structure of your programs, they should be as easy to read and understand as possible. *Lay your code out* so that it can be listed sensibly on a variety of devices: avoid having any lines longer than 80 characters as these may wrap (to reduce the number of “problem lines” you should use 4 spaces for indentation rather than tabs). *Sensible names* should be chosen for all variables, methods etc. *Documentation strings* should be included for each:

**Program** Fully explain what the program does and how it should be used.

Also state who wrote it and when.

**Class** Describe the objects to which the class relates. List and explain the data attributes of the objects.

**Method (or Function)** State what each method does and explain the roles of its parameters; you were given examples of useful examples of documentation strings when you wrote the marklist.py program.

In addition you should include occasional *comments* in your code; these may be (a) to introduce a new section in the code, or (b) to explain something that is not obvious. Bear in mind that pointless comments make your code harder to read, not easier.