**Scoreboard Evaluation**

**How well my program meets the quiz requirements**

My scoreboard program fully meets the requirements of the task.

The program is decomposed into six subprograms. Each subprogram (known as a function in Python) can be called as many times as necessary without having to repeat code, making the program more efficient and easier to read.

The ‘menu’ function correctly displays the three option choices, enables users to select an option and to quit the program. I have used an IF, ELIF, ELSE function in a WHILE loop to identify and validate the option choice input by the player. This allows the program to continue with a prompt for the option choice if invalid data is entered, therefore meeting the requirement for validation. I have used the BREAK command to ensure that there is a way of exiting the menu and enabling the program to end. All three options produce the required information.

I have used escape sequences for tab and new line (\t and \n), eliminating the need to use multiple print commands and therefore reducing the amount of code in the program.

I have used a global list (levelList) for the level numbers used to display the information for menu options A and B, which enabled me to simply refer to the list items in the subprograms without having to type out each list item each time

At the beginning of the program, I have included code to read from the playerScores text file and store valid records in a list. This was far more efficient than the first way I chose to do this, whereby I read the data from the text file every time an option was selected.

I created two new versions of the playerScores files (playerScores2.txt and playerScores3.txt) with additional data so that I could test that the scoreboard program works for any number of records and that if more than one player in option B or option C has the same highest score, each player with that score is displayed.

I have made my program easy to read by using meaningful variable names throughout. I have followed the conventions for writing program code by using underscores between each word of function names and camelCase for variable names. This makes it easier to distinguish between subprograms and variables. I have also included comments to explain the code.

**Improvements I could make**

To improve the program, I could have used more escape sequences so that the information could be displayed using just one print function, whereas I chose to display the information using several print functions. I did not have time to do this but this would make my code more efficient.

If I had more time I would also try to improve the output of options B and C so that they match the format given in the task.

**How the program finds the correct information for each of the three options**

When reading in the data file at the start of the program I have used an IF, OR statement to identify invalid scores and levels. If the conditions of the IF statement are met, the scores and levels for that player will be written to the error log. If the data is within the correct range, then it will be added to the list called playerData, using an append function. I initially used several IF statements to do this, but found it more effective to use an IF, OR statement.

I have carried out thorough testing to ensure that invalid records are excluded and correctly written to the error log and that only valid data is used in the scoreboard.

**Option A**

Firstly, I created and declared a variable called selectedPlayer to store the name input by the player. I then declared a variable called nextRecord, which was equal to the playerData list that was extracted from the text file. I used an IF statement to check if the first item (item 0) in nextRecord was equal to the player name stored in the variable selectedPlayer. I set the level scores to a default value of 0 using local variables to store this data, which is stated in the requirements of the task. I declared the variables level and score as integers so that they would be read as numbers. Using a FOR loop, each record is read by the program, and if the player name matches the selectedPlayer, then the score for each level is stored in the relevant variable, i.e. level1Score for the score for level one, and so on. I used a Boolean variable found so that this would only happen if the player name exists in the list; otherwise a print function displays the text “Invalid player name” and option A is run again. Using the IF statement that allows this to happen, if the name exists in the data then the scores for the selected player will be displayed, using print functions, along with the level number as a list item from the list ‘levelList’. Once complete, the return\_to\_menu function is run.

**Option B**

For option B, I declared local variables (level1HighestScore, level2HighestScore and so on) to store the high score for each level and set these to a default value of 0. I then created a set of lists to hold the names of the players with the highest score for each level. At the beginning of the program, I declared a global variable called numberOfItems, which calculates how many records are in the playerData list, using the length function. A FOR loop, in the Option B subprogram, looks through the items in the list playerData and runs for the number of times set in the variable numberOfItems. I set local variables for player, level and score to store items from the playerData list (player holding the record at 0, level at 1 and score at 2), and set level and score again to integers, so that they can be used in calculations later. Within this FOR loop, I used an IF, ELIF, ELSE statement to find the highest score for each level and store it in the variable for highest score (level1HighestScore and so on). I then used another FOR loop to loop through the records in playerData to find the names of the highest scorers and append them into the list level1TopScorers, level2TopScorers and so on. The IF statement used here checks if the level equals the correct number, i.e. 1, and the score equals the highest score for that level stored in level[number]HighestScore, if these conditions are true, then the player name will be appended to the list of top scorers.

I used a list here rather than a variable so that it could store more than one name, using a variable would only store one highest scorer for each level and would therefore not fully meet the requirements. Once the player name and score are stored in the lists for top scorer and the variable for highest score, the table will be displayed showing the relevant data. I did this using several print functions, escape sequences and by calling the labels from the levelList list, the value stored in the variable for highest score and then using FOR loops, I ran through each of the lists holding the names of the top scorers and displayed them in the table. I changed the format of the table to allow for several players with the same score. Once highest scorers for each level have been displayed, I ran the return\_to\_menu option to allow the player to access the main menu again.

**Option C**

To find the highest scorer overall, I used a FOR loop to run through the records in the playerData list, setting variables to store the player name and score. For this to work the list needed to be sorted into alphabetical order of player name. I used the .sort method to do this at the start of the program.

An IF statement is used so that if the current player name matches the previous player name in the list, it will take the scores and add them together and store them in a variable called currentTotal. If there are no other records with the same name, the player name, along with their total score (currentTotal) are appended to a new list called totalForEachPlayer.

Another IF statement is used to find out the highest score overall; if the value stored in currentTotal is greater than the highest score, then highest\_Score is set to currentTotal..

Once the loop is complete the print function will output a string in the correct format as set in the task, stating the highest score along with the name of the player. The FOR loop looks through the totalForEachPlayer list, finds the player with the highest score and prints their name. Using a list allows for more than one player to be the highest scorer, as stated in the requirements of the task. I changed the output format for this option so that it displays correctly when there are more than one player with the highest score.