
University of the West of England

| | |
|---|---|
| **Name:** | Reece Benson |
| **Student No.:** | 16021424 |
| **Module:** | Design and Analysis of Data Structures and Algorithms |
| **Assignment No.:** | Assignment 2 |
| **Due Date:** | Thursday 22nd March, 2018 |

# Table of Contents



# Brief

1. Design a solution to allow the processing of scores for both input types
2. Implement the solution designed in Task 1 in Python.
3. Design a solution for showing player information
4. Implement the design shown in Task 3
5. Evaluate the efficiency of your software
6. A second season will be introduced
7. Implement the design for Task 6

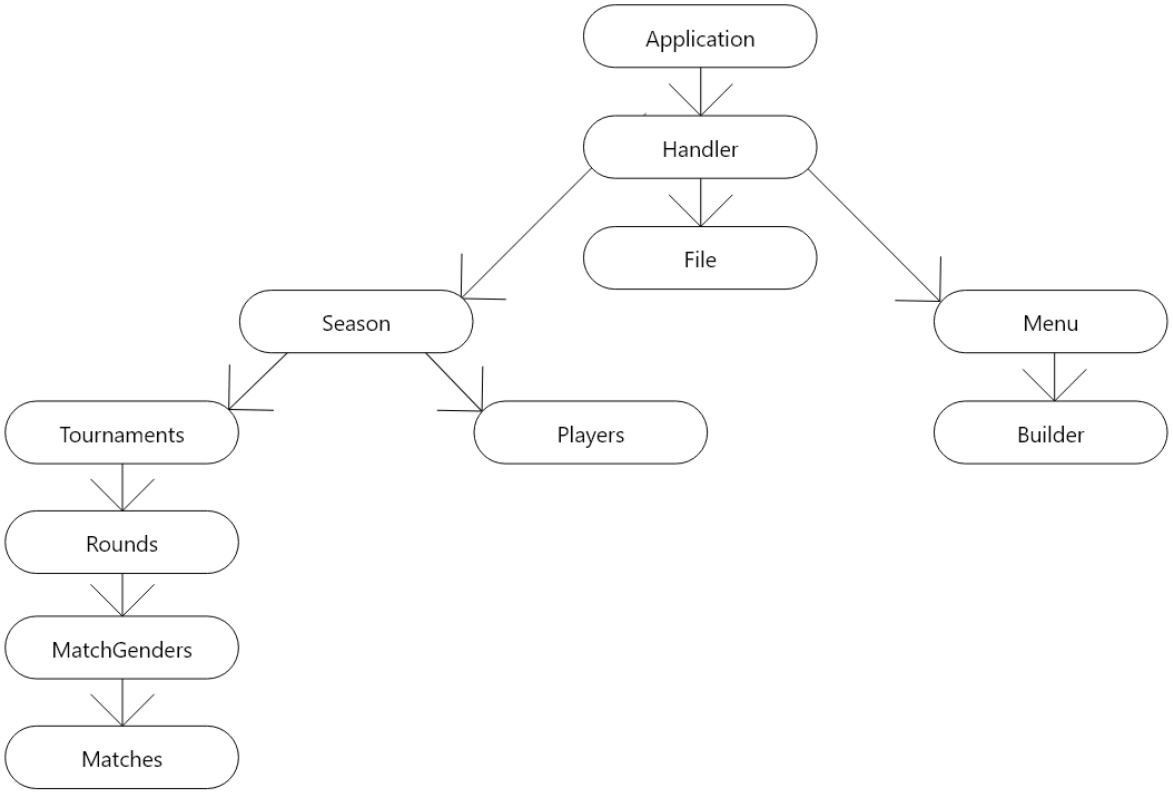
## Design a solution to allow the processing of scores for both input types

For this task, we are to design a solution in order to allowing the follow features:

- Allow the processing of scores for both input types (manual input/file input)
- Upon the end of each round, provide the user with the winners and options to go to the next round or save and exit the application.
- Upon choosing the next round, provide the user with options to manually input data or read from the file. In my application, reading from file is disabled when the previous round data is modified, and no longer fits in line with the file data.
- Once all rounds have been completed, the winner will be declared and options to view the tournaments rankings will be shown. In my application, you can access the current tournament rankings at any round, to see the progress of each player.

```
Please select an option: (Viewing: Season 1, TAC1, Round 1, Male)
1. View Round (Not Available)
2. View Prize Money (Not Available)
3. View Ranking Points (Not Available)
4. Input using file data
5. Input data manually
6. Go to Next Round (Not Available)
x. Save and Return
>>>
```

In order to implement my solution, I visualised my application into a tree-like manner and followed the flow for my development process, for example:



## Pseudo Code for Task 1

"Allow the processing of scores for both input types":

```
Print "Input using file data"
Print "Input data manually"
GetUserInput = response

if response is "file data" then
        ReadDataFile for <current round> = currentRoundData
        AddMatches(currentRoundData)
        Print currentRoundData
else if response is "manual input" then
        MatchCount = 0
        while (MatchCount is not the maximum matches for this round)
                Print "Enter Player 1 Name,Player 1 Score,Player 2 Name,Player 2 Score:"
                GetUserInput = matchData
                AddMatchToMatches(matchData)
                MatchCount + 1
        end while
end if
```

"Provide the user with the winners and options":

```
RoundWinners = <list of winners from the current round>
foreach winner from RoundWinners do
        Print winner


Print "Go to next round"
Print "Save and Exit"
GetUserInput = response

if response is "go to next round" then
        GoToNextRound()
else if response is "save and exit" then
        SaveAndExit()
end if
```

"Once all rounds have been completed, the winner will be declared and options to view the tournaments rankings will be shown":

```
if AllRoundsAreCompleted() is True then
        Print <the final round>.Winner
        Print <new line>
        Print "View Ranking Points Leaderboard"
        Print "View Prize Money Leaderboard"
        Print "View Overall Season Ranking"
        Print "Go to Next Tournament"
        Print "Save and Exit"

        GetUserInput = response
        if response is "view ranking points leaderboard" then
                DisplayRankingPointsLeaderboard(<tournament name>)
        else if response is "view prize money leaderboard" then
                DisplayPrizeMoneyLeaderboard(<tournament name>)
        else if response is "view overall season ranking" then
                DisplayOverallSeasonRanking(<season id>)
        else if response is "go to next tournament" then
                GoToNextTournament(<next tournament name>)
        else if response is "save and exit" then
                SaveAndExit()
        end if
end if
```

## Design a solution for showing player information (pseudo code)

```
Print “Select an option for viewing player information”
Print “1. The number of wins for a player with a particular score”
Print “2. The percentage of wins for a player”
Print “3. Player with most wins”
Print “4. Player with most loses”
GetUserInput = option

# Number of wins for a player with a particular score
if option is 1 then
        Print “Specific Tournament”
        Print “Overall Season”
        GetUserInput = type

        Print “Enter a player name, i.e. 3-0”
        GetUserInput = player_name

        Print “Enter a particular score, i.e. 3-0”
        GetUserInput = score_to_look_for
        if type is “specific” then
                MatchingWins = list()
                foreach match in <specified tournament>
                        if match matches score_to_look_for then
                                Add match To MatchingWins

                foreach match in MatchingWins
                        Print match
        else if type is “overall” then
                MatchingWins = list()
                foreach tournament in <this season>
                        foreach match in tournament
                                if match matches score_to_look_for then
                                        Add match To MatchingWins

                foreach match in MatchingWins
                        Print match
        end if

# Percentage of wins for a player
else if option is 2 then
        Print “Specific Tournament”
        Print “Overall Season”
        GetUserInput = type

        Print “Enter a player name, i.e. 3-0”
        GetUserInput = player_name
```

```
        if type is “specific” then
                player_wins = 0
                total_rounds = 0
                foreach match in <specified tournament>
                        if match.winner is player_name
                                player_wins + 1
                        total_rounds + 1

                Print player_name + “ has “ + player_wins + “ out of “ + total_rounds + “ (“ +
(player_wins / total_rounds) + “% overall)”

        else if type is “overall” then
                player_wins = 0
                total_rounds = 0
                foreach tournament in <this season>
                        foreach match in <specified tournament>
                                if match.winner is player_name
                                        player_wins + 1
                                total_rounds + 1

                Print player_name + “ has “ + player_wins + “ out of “ + total_rounds + “ (“ +
(player_wins / total_rounds) + “% overall)”
        end if

# Player with most wins
else if option is 3 then
        highest_players = list(<first player of player list>)

        foreach this_player in <season players>
                foreach highest_player in highest_players
                        if this_player.wins > highest_player.wins then
                                empty highest_players list
                                add this_player to highest_players
                        else if this_player.wins is highest_player.wins then
                                add this_player to highest_players

        foreach highest_player in highest_players
                Print highest_player.name
```

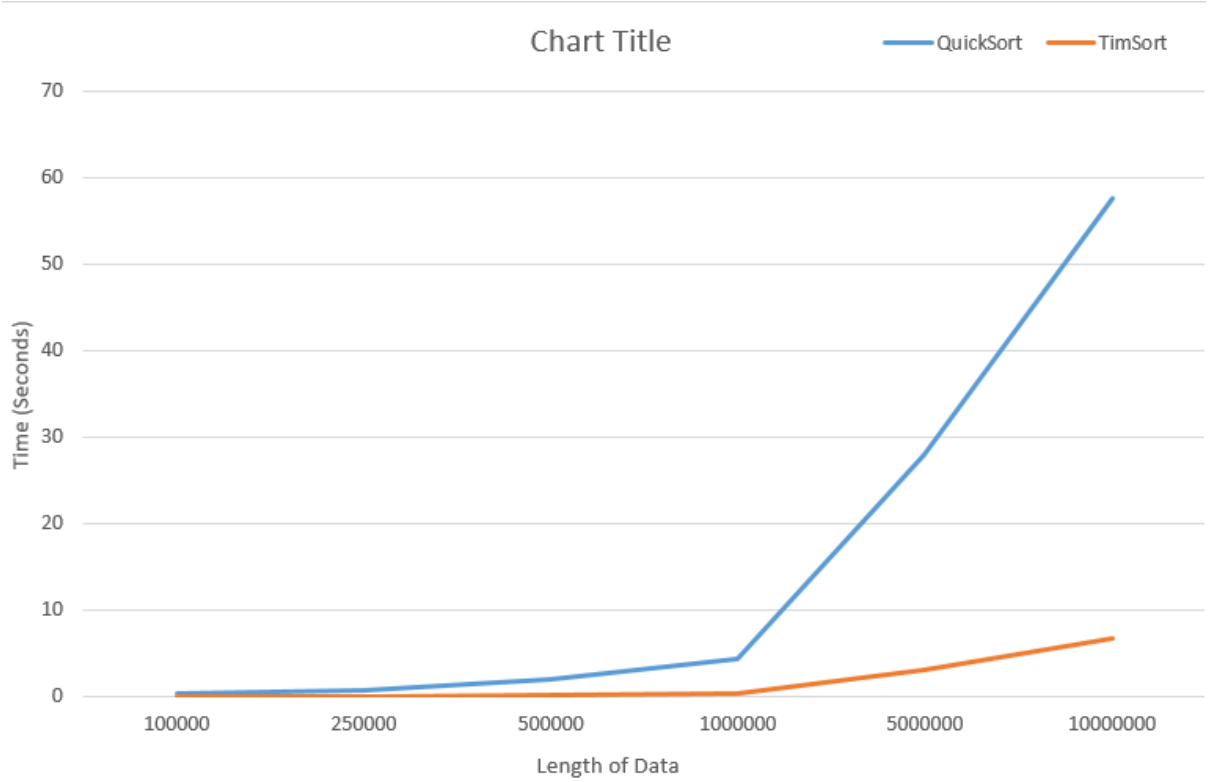
```
# Player with most loses
else if option is 4 then
        lowest_players = list(<first player of player list>)

        foreach this_player in <season players>
                foreach lowest_player in highest_players
                        if this_player.loses > lowest_player.loses then
                                empty lowest_players list
                                add this_player to lowest_players
                        else if this_player.loses is highest_player.loses then
                                add this_player to lowest_players

        foreach lowest_player in lowest_players
                Print lowest_player.name

end if
```

## Evaluate the efficiency of your software

For this section, I will be evaluating the efficiency of my software in terms of the size of code, speed of running and the efficient use of functions and specialist algorithms. Here, I implemented the use of **cProfile** to track the performance of functions I ran and the algorithms I used, and included some benchmarks of the comparisons between the algorithm I am using (Quick Sort), and Python's in-built **sorted** algorithm (Tim Sort), however I stuck with Quick Sort as we were advised to use our own algorithm.

Here are the benchmarks of the differences between Quick Sort and Tim Sort; had I the ability to use Python's in-built algorithm, I would've due to its amazing efficiency and the fact that in Python, Tim Sort is implemented in C. Having Tim Sort executed via C allows it to have a slight boost in performance in comparison to other sorting algorithms written in Python.

| | **100,000** | **250,000** | **500,000** | **1,000,000** | **5,000,000** | **10,000,000** |
|---|---|---|---|---|---|---|
| **Quick Sort** | 0.401s | 0.824s | 1.968s | 4.429s | 27.965s | 57.691s |
| **Tim Sort** | 0.03s | 0.09s | 0.199s | 0.461s | 3.06s | 6.752s |




We can see here, that Tim Sort (BigO notation of O(n)), it represents that in the graph quite accurately. If I was to do the tests with a standard iteration (i.e. iterations of 1,000,000), the graph would represent that including with Quick Sort (BigO notation of O(n log n)), it also represents that in the above graph.

I then moved on to using **cProfile** that is a Python in-built library that is used to evaluate the time taken to execute a partition of code. The code I had written in order to view the ranking points and prize money includes the use of the Quick Sort algorithm I had implemented:

**Pseudo Code:**

```
function quick_sort(data)
        if length of array is equal to or below 1
                return data
        else
                pivot = <first element of data>

                greater_half = <get all elements of data that are greater than the pivot>
                less_half = <get all elements of data that are less than or equal to the pivot>

                # concatenate all of our arrays via function recursion
                # and our pivot element should be cast into an array
                return quick_sort(less_half) + array(pivot) + quick_sort(greater_half)
        end if
end function
```

**Python Code:**

```python
def quick_sort_score(arr, attr="score"):
    if(len(arr) <= 1): return arr
    else:
        piv = arr[0]
        gt  = [ e for e in arr[1:] if e[attr] > piv[attr] ]
        lt  = [ e for e in arr[1:] if e[attr] <= piv[attr] ]
        return quick_sort_score(lt, attr) + [piv] + quick_sort_score(gt, attr)
```

***[ The next page contains the cProfile Statistics ]***

**cProfile Statistics for "view_prize_money()" of a specific tournament:**

```
         2605 function calls (2549 primitive calls) in 0.149 seconds
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.142    0.142 Game.py:118(clear_screen)
        1    0.000    0.000    0.000    0.000 MatchGender.py:52(get_gender)
        5    0.000    0.000    0.000    0.000 MatchGender.py:55(is_complete)
     2260    0.000    0.000    0.000    0.000 Player.py:27(get_name)
       28    0.000    0.000    0.000    0.000 QuickSort.py:10(<listcomp>)
     57/1    0.000    0.000    0.000    0.000 QuickSort.py:5(quick_sort_score)
       28    0.000    0.000    0.000    0.000 QuickSort.py:9(<listcomp>)
        5    0.000    0.000    0.000    0.000 Round.py:51(get_gender)
        1    0.000    0.000    0.000    0.000 Round.py:57(get_id)
        1    0.000    0.000    0.000    0.000 Season.py:54(get_id)
       58    0.000    0.000    0.001    0.000 Season.py:94(get_player)
        1    0.000    0.000    0.000    0.000 Tournament.py:41(get_name)
        1    0.000    0.000    0.000    0.000 Tournament.py:47(get_rounds)
        1    0.000    0.000    0.000    0.000 Tournament.py:48(<listcomp>)
       57    0.000    0.000    0.000    0.000 {built-in method builtins.len}
       33    0.005    0.000    0.005    0.000 {built-in method builtins.print}
        1    0.142    0.142    0.142    0.142 {built-in method nt.system}
       32    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
       33    0.000    0.000    0.000    0.000 {method 'format' of 'str' objects}
```

**cProfile Statistics for "view_ranking_points()" of a specific tournament:**

```
         3559 function calls (3507 primitive calls) in 0.150 seconds
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.139    0.139 Game.py:118(clear_screen)
      171    0.000    0.000    0.000    0.000 Match.py:62(get_player_one)
       57    0.000    0.000    0.000    0.000 Match.py:65(get_player_two)
      201    0.000    0.000    0.000    0.000 Match.py:68(get_winner)
       57    0.000    0.000    0.000    0.000 Match.py:71(get_player_winner)
       57    0.000    0.000    0.000    0.000 Match.py:74(get_player_loser)
       31    0.000    0.000    0.000    0.000 Match.py:80(get_match_bonuses)
        5    0.000    0.000    0.000    0.000 MatchGender.py:160(get_matches)
        5    0.000    0.000    0.000    0.000 MatchGender.py:161(<listcomp>)
        5    0.001    0.000    0.005    0.001 MatchGender.py:198(finalise)
        5    0.000    0.000    0.000    0.000 MatchGender.py:206(<listcomp>)
        5    0.000    0.000    0.000    0.000 MatchGender.py:207(<listcomp>)
        1    0.000    0.000    0.000    0.000 MatchGender.py:52(get_gender)
        5    0.000    0.000    0.000    0.000 MatchGender.py:55(is_complete)
     2260    0.000    0.000    0.000    0.000 Player.py:27(get_name)
       26    0.000    0.000    0.000    0.000 QuickSort.py:10(<listcomp>)
     53/1    0.000    0.000    0.000    0.000 QuickSort.py:5(quick_sort_score)
       26    0.000    0.000    0.000    0.000 QuickSort.py:9(<listcomp>)
        9    0.000    0.000    0.000    0.000 Round.py:51(get_gender)
      159    0.000    0.000    0.000    0.000 Round.py:57(get_id)
        1    0.000    0.000    0.000    0.000 Season.py:54(get_id)
       58    0.001    0.000    0.001    0.000 Season.py:94(get_player)
        1    0.000    0.000    0.000    0.000 Tournament.py:41(get_name)
        1    0.000    0.000    0.000    0.000 Tournament.py:47(get_rounds)
        1    0.000    0.000    0.000    0.000 Tournament.py:48(<listcomp>)
        4    0.000    0.000    0.000    0.000 Tournament.py:50(get_round)
       32    0.000    0.000    0.000    0.000 Tournament.py:53(get_difficulty)
       57    0.000    0.000    0.000    0.000 {built-in method builtins.len}
       69    0.007    0.000    0.007    0.000 {built-in method builtins.print}
        1    0.139    0.139    0.139    0.139 {built-in method nt.system}
       90    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
       99    0.000    0.000    0.000    0.000 {method 'format' of 'str' objects}
        5    0.000    0.000    0.000    0.000 {method 'keys' of 'dict' objects}
```

**cProfile Statistics for Viewing Matches with Particular Score of All Tournaments within a Season:**

```
         657 function calls in 0.002 seconds
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      136    0.000    0.000    0.000    0.000 Match.py:62(get_player_one)
      136    0.000    0.000    0.000    0.000 Match.py:65(get_player_two)
        6    0.000    0.000    0.000    0.000 Match.py:68(get_winner)
        6    0.000    0.000    0.000    0.000 Match.py:77(get_match_text)
       20    0.000    0.000    0.000    0.000 MatchGender.py:159(get_matches)
       20    0.000    0.000    0.000    0.000 MatchGender.py:160(<listcomp>)
       20    0.000    0.000    0.000    0.000 MatchGender.py:54(is_complete)
      244    0.000    0.000    0.000    0.000 Player.py:27(get_name)
       20    0.000    0.000    0.000    0.000 Round.py:51(get_gender)
        6    0.000    0.000    0.000    0.000 Round.py:57(get_id)
        1    0.000    0.000    0.000    0.000 Season.py:57(get_tournaments)
        1    0.000    0.000    0.000    0.000 Season.py:58(<listcomp>)
        4    0.000    0.000    0.000    0.000 Tournament.py:41(get_name)
        4    0.000    0.000    0.000    0.000 Tournament.py:47(get_rounds)
        4    0.000    0.000    0.000    0.000 Tournament.py:48(<listcomp>)
       12    0.001    0.000    0.001    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
       16    0.000    0.000    0.000    0.000 {method 'format' of 'str' objects}
```

**cProfile Statistics for Viewing Player with Win Percentage of All Tournaments within a Season:**

```
         424 function calls in 0.004 seconds
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      124    0.000    0.000    0.000    0.000 Match.py:68(get_winner)
       20    0.000    0.000    0.000    0.000 MatchGender.py:159(get_matches)
       20    0.000    0.000    0.000    0.000 MatchGender.py:160(<listcomp>)
       20    0.000    0.000    0.000    0.000 MatchGender.py:54(is_complete)
      128    0.000    0.000    0.000    0.000 Player.py:27(get_name)
       20    0.000    0.000    0.000    0.000 Round.py:51(get_gender)
       20    0.000    0.000    0.000    0.000 Round.py:57(get_id)
        1    0.000    0.000    0.000    0.000 Season.py:57(get_tournaments)
        1    0.000    0.000    0.000    0.000 Season.py:58(<listcomp>)
        4    0.000    0.000    0.000    0.000 Tournament.py:41(get_name)
        4    0.000    0.000    0.000    0.000 Tournament.py:47(get_rounds)
        4    0.000    0.000    0.000    0.000 Tournament.py:48(<listcomp>)
       29    0.003    0.000    0.003    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
       28    0.000    0.000    0.000    0.000 {method 'format' of 'str' objects}
```

**cProfile Statistics for Finding Player with Highest Amount of Wins of All Tournaments within a Season:**

```
         258 function calls in 0.002 seconds
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       20    0.000    0.000    0.000    0.000 MatchGender.py:54(is_complete)
        1    0.000    0.000    0.000    0.000 Player.py:27(get_name)
      125    0.000    0.000    0.000    0.000 Player.py:63(get_total_wins)
       20    0.000    0.000    0.000    0.000 Round.py:51(get_gender)
       20    0.000    0.000    0.000    0.000 Round.py:57(get_id)
        1    0.000    0.000    0.000    0.000 Season.py:57(get_tournaments)
        1    0.000    0.000    0.000    0.000 Season.py:58(<listcomp>)
        2    0.000    0.000    0.000    0.000 Season.py:91(get_players)
        4    0.000    0.000    0.000    0.000 Tournament.py:41(get_name)
        4    0.000    0.000    0.000    0.000 Tournament.py:47(get_rounds)
        4    0.000    0.000    0.000    0.000 Tournament.py:48(<listcomp>)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
       26    0.002    0.000    0.002    0.000 {built-in method builtins.print}
        2    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
       26    0.000    0.000    0.000    0.000 {method 'format' of 'str' objects}
```

**cProfile Statistics for Finding Player with Highest Amount of Loses of All Tournaments within a Season:**

```
         2142 function calls in 0.005 seconds
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       20    0.000    0.000    0.000    0.000 MatchGender.py:54(is_complete)
       28    0.000    0.000    0.000    0.000 Player.py:27(get_name)
     1856    0.000    0.000    0.000    0.000 Player.py:66(get_total_lost)
       20    0.000    0.000    0.000    0.000 Round.py:51(get_gender)
       20    0.000    0.000    0.000    0.000 Round.py:57(get_id)
        1    0.000    0.000    0.000    0.000 Season.py:57(get_tournaments)
        1    0.000    0.000    0.000    0.000 Season.py:58(<listcomp>)
        2    0.000    0.000    0.000    0.000 Season.py:91(get_players)
        4    0.000    0.000    0.000    0.000 Tournament.py:41(get_name)
        4    0.000    0.000    0.000    0.000 Tournament.py:47(get_rounds)
        4    0.000    0.000    0.000    0.000 Tournament.py:48(<listcomp>)
       21    0.000    0.000    0.000    0.000 {built-in method builtins.len}
       53    0.005    0.000    0.005    0.000 {built-in method builtins.print}
       27    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
       80    0.000    0.000    0.000    0.000 {method 'format' of 'str' objects}
```

## Design a solution for the second season (pseudo code)

Due to this task being very confusing to read at first, I have separated and broken down the specification of this task into sub categories, and this is how I will list the pseudo code for this task.

**First Round Pairs (Manual Input Checks & Validation)**

```
top_half_players = <top half of round 1, season 1, tournament X>
bottom_half_players = <bottom half of round 1, season 1, tournament X>

player_names = list
current_matches = list
while (True)
        Print <current created matches>
        Print <available players to pair>

        GetUserInput = first_player_in_pair

        if round is first or 1 then
                Print <available players to pair with first_player_in_pair>

                if not valid pair then
                        continue # restart loop
                else
                        Pop selected players from top_half_players, bottom_half_players
                        GetUserInput = scores_for_pair
                        Append match to current_matches
                end if
        end if
end while
```

**Second Round and Above Pairs (Manual Input Checks & Validation)**

```
player_names = list
current_matches = list
while (True)
        Print <current created matches>
        Print <available players to pair>

        GetUserInput = first_player_in_pair

        if round is first or 1 then
                Print <available players to pair with first_player_in_pair>

                if not valid pair then
                        continue # restart loop
                else
                        Pop selected players from available_players, player_names
                        GetUserInput = scores_for_pair
                        Append match to current_matches
                end if
        end if
end while
```

**Points Difficulty Factor For Achievement**

```
season_id = GetCurrentSeasonId()

tournament_difficulty = GetCurrentTournamentDifficulty()

if season_id is above 1 then

        if ((player is in <previous season round>.winners_list or

          player is in <current season round>.winners_list) AND

          player.win_count >= GetCurrentRoundId()) then

                # tournament difficulty stays the same

        else

                # set tournament difficulty to 1 so we don't multiply by zero

                tournament_difficulty = 1

        end if

end if
```

**Combination of Season Overall Leaderboards**

Prize Money

```
player_money = list
foreach season in GetSeasons()
        foreach player in season.GetPlayers()
                foreach tournament in season.GetTournaments()
                        if player in player_money then
                                update player element in player_money
                        else
                                add player element to player_money
                        end if
                end foreach
        end foreach
end foreach

Sort(player_money) = sorted_player_money
foreach player in sorted_player_money
        Print player + ", " + player.money
end foreach
```

Ranking Points

```
player_score = list
foreach season in GetSeasons()
        foreach player in season.GetPlayers()
                foreach tournament in season.GetTournaments()
                        if player in player_score then
                                update player element in player_score
                        else
                                add player element to player_score
                        end if
                end foreach
        end foreach
end foreach

Sort(player_score) = sorted_player_score
foreach player in sorted_player_score
        Print player + ", " + player.score
end foreach
```