COMP 2034
Software Development

Python Programming Assignment 2

UniSA STEM
The University of South Australia
May 2020

Contents

Introduction

This document describes the second programming assignment for COMP 2034 Software Development course.

The assignment is intended to provide you with the opportunity to put into practice what you have learnt in the course by applying your knowledge and skills in Object-Oriented Programming with Python programming language. The task is to develop a program with multiple modules which allows users to play the game of Rock-Paper-Scissors or the game of In-between against the computer and maintains information on players. Player information will be stored in a text file that will be read in when the program commences. Once the application has read the initial player data, it should allow the user to interactively query and manipulate the player information as well as play games against the computer.

This assignment is an **individual task** that will require an **individual submission**. Each student is required to **submit your work via learnonline before Monday 15 June 2020, 10am (swot-vac week).**

You will also be required to present your work to your supervisor during your allocated Zoom session held in the swot-vac week of the study period. Important: You must attend your allocated session (schedule to be announced on week 12) in order to have your assignment marked.

This document is a specification of the required end product that will be generated by implementing the assignment. Like many specifications, it is written in English and hence will contain some imperfectly specified parts. Please make sure you seek clarification if you are not clear on any aspect of this assignment.

Graduate Qualities

By undertaking this assessment, you will progress in developing the qualities of a University of South Australia graduate. The Graduate qualities being assessed by this assignment are:

- The ability to demonstrate and apply a body of knowledge (GQ1) gained from the lectures, practicals, and readings. This is demonstrated in your ability to apply programming theory to a practical situation.
- The ability to effectively problem solve (GQ3) using Python and Object-Oriented Programming concepts to complete the programming problem. Effective problem solving is demonstrated by the ability to understand what is required, utilise the relevant information from lectures, the text book and practical work, write Python code, and evaluate the effectiveness of the code by testing it.
- The use of communication skills (GQ6) by producing source code that has been properly formatted; and by writing adequate, concise and clear comments. Communicating with others through demonstrating and explaining the program code to the instructor.

Specifications and Requirements

Your solution MUST adhere to the specifications and requirements described in this document.

It is recommended that you develop this assignment in stages and make back-ups of your code regularly not only for development purpose, but also as an evidence of original work.

Your program must be developed using multiple Python modules, with the number and names of all the files strictly adhering to the specifications below.

Your program must be developed with six Python files, three of them provided, and three of them newly written by yourself. These files must be:

- `assignment2.py` - This file contains the main part of your program to import the other modules and run. It allows the user to interactively query and manipulate the player information and play games. (PROVIDED)
- `player.py` - This file contains `Player` class definition to store a single player information.
- `leaderboard.py` - This file contains `LeaderBoard` class definition to manage players on a leader board.
- `game.py` - This file contains `Game` class definition which is the base class of other game classes. (PROVIDED)
- `rpsgame.py` - This file contains `RockPaperScissors` class definition which implements a Rock-Paper-Scissors game. (PROVIDED)
- `inbetweengame.py` - This file contains `Inbetween` class definition which implements an In-between game.

Three files (`assignment2.py`, `game.py`, `rpsgame.py`) **will be provided on the course website** along with this document and they must be **used without any modification**. To solve this assignment, you must create and write three additional files `player.py`, `leaderboard.py`, and `inbetweengame.py`.

Program Behaviour

When your run the program `assignment2.py`, it will create an instance object of the `LeaderBoard` class you define and call the `load()` method which should load in player information from a file called `players.txt` (provided on the course website). If the program has loaded the data successfully, the following message will be displayed:
`Players info successfully loaded.`
Or if the program fails to load the players info, the following message will be displayed:
`ERROR: Cannot load players info.`

Your program will enter the command mode after the player information has been loaded from the file. In the command mode, the program reads user input for a command with the following prompt shown on the screen:

```
Please enter a command [list, add, remove, play, winner,
quit]:
```

The program will allow the user to enter commands and process these commands until the 'quit' command is entered.

The following commands must be supported:

Command	Description
list	Displays the leader board (a list of all players and their details) by calling the `display()` method of the `LeaderBoard` object.
add	Prompts to input a name for a new player to add, and calls the `addPlayer()` method of the `LeaderBoard` object to add the player with the provided name. Depending on the return value from the `addPlayer()` method, the following messages are displayed on the screen with the player's *NAME* : - returned `True`: "Successfully added player *NAME*. " - returned `False`: "Player *NAME* already exists. "
remove	Prompts to input the player's name to remove then calls the `removePlayer()` method of the `LeaderBoard` object to remove the player with the given name. Depending on the return value, the following messages are displayed on the screen with the player's *NAME* : - returned `True`: "Successfully removed player *NAME*. " - returned `False`: "No such player found."
play	Asks to input the player's name who will play the game, and checks the player's points by calling the `getPlayerPoints()` method of the `LeaderBoard` object. If the player is not found (i.e. the return value is negative), an error message "No such player found." is displayed to the screen, otherwise asks for the amount of points to bid, then asks to choose which game to play. The chosen game will be played by creating an instance of either `RockPaperScissors` or `Inbetween` class, then calling its `play()` method which will return the result of the game (`1`: win, `0`: tie, `-1`: lose). After playing the game, the returned results and the points bid will be passed on to calling the `recordGamePlay()` method of the `LeaderBoard` object to update the player information.
winner	Prints the details of the winning player who has the highest number of points by calling the `getWinner()` method of the `LeaderBoard` object.

quit	Causes the program to quit, displaying a message "Thank you for playing!". Upon quitting, the program will call the `save()` method of the `LeaderBoard` object which will save the player information to a file named `output.txt` which is in the same format as the input `players.txt` file. Upon successfully saving the information the program will show a message, "Players info successfully saved." If it fails saving the information, the program will show a message, "ERROR: Cannot save players info."

After performing each command, the program returns to command mode, prompting the user to input next command.

Class Specifications

Below are detailed specifications of the three classes you must define in three files: `player.py`, `leaderboard.py`, and `inbetweengame.py`.

- `Player` class

The `Player` class stores the information for each player. This class must be defined in the `player.py` file, and must have the following *public* data attributes:

Data Attributes
A string, name of the player which may include white spaces.
An integer, number of games played
An integer, number of games won
An integer, number of games lost
An integer, number of games tied
An integer, current points

The `Player` class must have an initialiser method that takes the name of the player as a parameter, and initialises the name data attribute, as well as sets the rest of the attributes to `0`, except the current points which must be initialised to `100`.

The `Player` class must also have a string conversion method that returns a string describing the player object including the name, the number of games played, winning rate, and the current points remaining. For example, for a player named `"John Doe"` who has won 4 games out of 7 games and has 75 remaining points, it must return a string in the following format:

```
"John Doe has 75 points and a winning rate of 57.1%."
```

Note the winning rate should show 1 digit under the decimal point. If the player did not play any games, the string must be in the following format:

```
"John Doe has 100 points, and never played a game."
```

• `LeaderBoard` class

The `LeaderBoard` class manages a list of players (i.e., instance objects of the `Player` class). The `LeaderBoard` class must be defined in the `leaderboard.py` file, and must have only one *private* (i.e. hidden) data attribute which is the list of players.

The `LeaderBoard` class must define eight public methods, `load()`, `save()`, `display()`, `addPlayer()`, `removePlayer()`, `getPlayerPoints()`, `getWinner()`, and `recordGamePlay()`, as well as two private methods, `__findPlayer()` and `__sortPlayers()`. Below are detailed specifications of each method:

- `load()`

This method loads in the player information from a file named `players.txt` (provided on the course website together with this document). It reads in the file and creates a list of instances of the `Player` class you defined, and updates the list of players data attribute with the loaded information. In the `players.txt` file, each player is described in two lines of text. The name of the player (which may include white spaces) is stored in the first line. The very next line contains the number of games played, games won, games lost, games tied, and the current remaining points, all stored in one line and separated by the space character. Below is a sample content of the `players.txt` file with three players:

```
John Doe
5 4 0 1 70
Lisa Smith
16 13 2 1 105
Andrew Whittaker
7 0 7 0 55
```

You must use the input file `players.txt` provided on the course website. You are not supposed to create it yourself or edit the provided input file. You may assume that all data in this file is in the correct format. Note there could be empty lines at the end of the file which should be ignored.

After loading the data, the `load()` method must call the `__sortPlayers()` method to keep the player list in the order of their points. It should not print any message onto the screen, but must return `True` if the data has been successfully loaded, or `False` if there is any error.

- `save()`

This method saves the player information to the `output.txt` file which should have the same format as the input `players.txt` file. It should not print any message onto the screen, but must return `True` if the data has been successfully saved, or `False` if there is any error.

- `display()`

This method shows the leader board onto the screen in the format as described below:

- The player name field is displayed under the “Player Name” heading, and should be 30 characters wide, left justified.
- The number of games played is displayed under the “P” heading, and should be 2 characters wide, right justified.
- The number of games won is displayed under the “W” heading, and should be 2 characters wide, right justified.
- The number of games lost is displayed under the “L” heading, and should be 2 characters wide, right justified.
- The number of games tied is displayed under the “T” heading, and should be 2 characters wide, right justified.
- The winning rate is displayed under the “W-Rate” heading, and should be 6 characters wide, right justified, showing 1 digit under the decimal point with a percent sign (%) at the end (Hint: 100.0% has 6 characters). Note that the winning rate is not part of the information stored in the Player structure, but should be calculated based on number of games played and number of games won. If the number of games played is zero, the winning rate is treated as zero percent.
- The points is displayed under the “Points” heading, and should be 6 characters wide, right justified.
- Each of the field should be separated by a single empty space character.

Below is an example of how the output should look like:

```
|-------------------------------------------------------------|
| Player Name                     P  W  L  T  W-Rate Points |
|-------------------------------------------------------------|
| John Doe                        5  4  0  1   80.0%     70 |
| Lisa Smith                     16 13  2  1   81.3%    105 |
| Andrew Whittaker                7  0  7  0    0.0%     55 |
|-------------------------------------------------------------|
```

When there is no player (i.e. all players removed), it should show a “No player to display.” message as below:

```
|-------------------------------------------------------------|
| Player Name                     P  W  L  T  W-Rate Points |
|-------------------------------------------------------------|
|                   No player to display.                     |
|-------------------------------------------------------------|
```

- `addPlayer()`

This method takes the name of a new player to add as a parameter, creates an instance object of the `Player` class with the provided name, and adds that object to the player list

data attribute. Before creating a new player, the method should call the `__findPlayer()` method to check if there is already a player with the same name, in which case the method should not add a new player but simply return `False`. After adding a new player, it should also call the `__sortPlayers()` method to sort the list of players. The method must return `True` after successfully adding a player.

- `removePlayer()`
This method takes the name of a player as a parameter, and removes the player with the provided name. It should use the `__findPlayer()` to first identify the `Player` object to remove from the player list data attribute. If there is no player with the provided name, the method must return `False`, or otherwise return `True` after the player was successfully removed.

- `getPlayerPoints()`
This method takes the name of a player as a parameter, and returns the current points of the player with the provided name. If there is no such player with the provided name, it must return `-1`.

- `getWinner()`
This method returns a `Player` object that has the highest number of points. If there is no players in the player list, it must return `None`.

- `recordGamePlay()`
This method updates the player information based on the provided game play results. It takes three parameters: the name of a player, number of points bid, and the result of a game play. The result of a game play will be `1` in case the player won the game, `-1` if the player lost, and `0` if it was a tie. The number of games played, won, lost, tied, and the current point data attributes of the player must be updated accordingly. For winning, the points will be increased by the amount of bid, while the points will be reduced by the amount of bid if lost. If it was a tie game, no changes to the points are made. It must call the `__sortPlayers()` method to keep the player list in the order of their points.

- `__findPlayer()`
This method takes the name of a player as a parameter, finds the player with the provided name from the list of players, and returns the `Player` object. If there is not such a player with the provided name, it must return `None`.

- `__sortPlayers()`
This method updates the list of players by sorting it in the order of the remaining points.

• `Inbetween` class
The `Inbetween` class implements the game of In-between defined in `inbetweengame.py` module, extending (i.e. inheriting) the base class `Game` defined in the `game.py` module provided on the course website together with this document. The `Inbetween` class must override the `play()` method of the base class.

`-play()`
This method lets the user play a single round of the game of In-between.

First of all, it must print the following message on the screen:
```
Let's play In-between!
```

Next, two random numbers between 1 to 10 (inclusive) are generated. If the `debugMode` property inherited from the superclass is set to `True`, it should print the two numbers that are generated.

Then the user is asked to choose a number between 1 to 10 (inclusive) with a prompt `"Choose a number (1-10): "`. If the user input is not within the range, the program must ask again.

If the user chose the number that is in-between the two numbers (exclusive) that computer generated, the user wins. If the number that user chose is equal to one of the two numbers computer generated, the game is a tie. Otherwise, the payer loses the game. The program must show the result of the game as one of the followings:
```
Computer chose 7 and 8, you chose 3.
You lose!

Computer chose 3 and 8, you chose 4.
You win!

Computer chose 2 and 9, you chose 2.
Tie!
```

Finally, the method must return `1` if the user won the game, `-1` if the user lost, and `0` if tied.

Hint: Please refer to the `RockPaperScissors` class defined in `rpsgame.py` module provided on the course website together with this document to learn what is expected as a subclass of the `Game` class.

Screen Format

Please refer to the 'Sample Output' section at the end of this document to ensure that your program is behaving correctly and that you have the correct output messages.

General and Structural Requirements

Your solution must adhere to the following requirements:

- Your solution must be based on the Python modules and other files provided on the course website together with this document.
- Supplied files `assignment2.py`, `game.py`, `rpsgame.py`, and `players.txt` must not be modified.

- Each class you define must be organised into appropriate modules as described in the specification, including `player.py`, `leaderboard.py`, and `inbetweengame.py`.
- Use appropriate and well-constructed `while`, `for` and `if` statements as necessary. (You may lose marks if you unnecessarily embed if statements in another if statement. Rather use logical operators (i.e., `and`, `or`, `not`) or `elif` statements for combing multiple conditions.)
- User inputs should be validated and asked again if invalid (out of range or not among the provided options) with messages displayed as shown in sample outputs.
- Define and use classes and objects appropriately. All of the classes outlined in the specification must be implemented and used as specified. If more classes, functions, or methods are added, they should be fully implemented and used in your solution.
- Output must strictly adhere to the assignment specifications. If you are not sure about these details, you should check with the 'Sample Output' section at the end of this document or post a message to the discussion forum.
- Make appropriate comments. You are to provide comments to describe: your personal details at the beginning of each .py file you wrote, all variable and data attribute definitions, all functions and methods definitions and every significant section of code.
- Use meaningful variable, data attribute, function, method, and class names (no single letter identifier names, except for index variable accessing array elements).
- Your code must run on Python 3.8 IDLE environment.

Submission Requirements

You are required to do the following in order to submit your work and have it marked.

This assignment is an **individual task** that will require an **individual submission**. You are required to submit an electronic copy of your program source code (.py files) via **learnonline before Monday 15 June 2020, 10am (swot-vac week)**. No further changes can be made to your assignment passed the deadline.

You are **also required to present your assignment to a supervisor for marking during your allocated Zoom session held during the swot-vac week of the study period (details to be announced on Week 12)**. The supervisor will mark your work using the marking criteria included in this document.

Assignments submitted to learnonline, but not demonstrated during your allocated session, will NOT be marked. Likewise, assignments that have been demonstrated during the practical session, but have not been submitted via learnonline, will NOT be marked. Assignments are submitted to learnonline in order to check for plagiarism.

All students must follow the submission instructions below:
You must submit **three .py files** of your work with exactly following the names as required:

```
player.py
leaderboard.py
inbetweengame.py
```

The files you submit must include the following comments including your own student details at the very beginning of each file:

```
#
# File: filename.py
# Author: Steve Jobs
# Student ID: 12345678
# Email ID: jobst007
# This is my own work as defined by
#  the University's Academic Misconduct Policy.
#
```

Assignments that do not contain these details may not be marked, unless the student can prove the file is his/her own work through presenting the progress of development with a group of backup files. It is expected that students will make copies and back-ups of all assignments in various stages and be able to provide these if required.

Extensions and Late Submissions

There will be no extensions/late submissions for this course without one of the following exceptions:

1. A medical certificate is provided that has the timing and duration of the illness and an opinion on how much the student's ability to perform has been compromised by the illness. Please note if this information is not provided the medical certificate WILL NOT BE ACCEPTED. Late assessment items will not be accepted unless a medical certificate is presented to the Course Coordinator. The certificate must be produced as soon as possible and must cover the dates during which the assessment was to be attempted. In the case where you have a valid medical certificate, the due date will be extended by the number of days stated on the certificate up to five working days.
2. A Learning and Teaching Unit councillor contacts the Course Coordinator on your behalf requesting an extension. Normally you would use this if you have events outside your control adversely affecting your course work.
3. Unexpected work commitments. In this case, you will need to attach a letter from your work supervisor with your application stating the impact on your ability to complete your assessment.
4. Military obligations with proof.

Applications for extensions must be lodged with the Course Coordinator before the due date of the assignment.

Note: Equipment failure, loss of data, 'Heavy work commitments' or late starting of the course are not sufficient grounds for an extension.

Academic Misconduct

Students are reminded that they should be aware of the academic misconduct guidelines available from the University of South Australia website.

Deliberate academic misconduct such as plagiarism is subject to penalties. Information about Academic integrity can be found in Section 9 of the Assessment policies and procedures manual at:
https://i.unisa.edu.au/policies-and-procedures/codes/assessment-policies/

Sample Output

Sample Output 1:

```
Players info successfully loaded.

Please enter a command [list, add, remove, play, winner,
quit]: list
|-------------------------------------------------------------|
| Player Name                    P  W  L  T  W-Rate Points    |
|-------------------------------------------------------------|
| Lisa Smith                    16 13  2  1   81.2%    105   |
| John Doe                       5  4  0  1   80.0%     70   |
| Andrew Whittaker               7  0  7  0    0.0%     55   |
|-------------------------------------------------------------|

Please enter a command [list, add, remove, play, winner,
quit]: add
Name: Steve Jobs
Successfully added player Steve Jobs.

Please enter a command [list, add, remove, play, winner,
quit]: list
|-------------------------------------------------------------|
| Player Name                    P  W  L  T  W-Rate Points    |
|-------------------------------------------------------------|
| Lisa Smith                    16 13  2  1   81.2%    105   |
| Steve Jobs                     0  0  0  0    0.0%    100   |
| John Doe                       5  4  0  1   80.0%     70   |
| Andrew Whittaker               7  0  7  0    0.0%     55   |
|-------------------------------------------------------------|

Please enter a command [list, add, remove, play, winner,
quit]: add
Name: John Doe
Player John Doe already exists.

Please enter a command [list, add, remove, play, winner,
quit]: remove
Name: Bill Gates
No such player found.

Please enter a command [list, add, remove, play, winner,
quit]: remove
Name: John Doe
Successfully removed player John Doe.

Please enter a command [list, add, remove, play, winner,
quit]: list
```

```
|-------------------------------------------------------------|
| Player Name                    P  W  L  T  W-Rate Points    |
|-------------------------------------------------------------|
| Lisa Smith                    16 13  2  1   81.2%    105   |
| Steve Jobs                     0  0  0  0    0.0%    100   |
| Andrew Whittaker               7  0  7  0    0.0%     55   |
|-------------------------------------------------------------|

Please enter a command [list, add, remove, play, winner,
quit]: play
Name: John Doe
No such player found.

Please enter a command [list, add, remove, play, winner,
quit]: play
Name: Steve Jobs
How many points to bid (1-100)? 0
How many points to bid (1-100)? 30
Which game ([r]Rock-Paper-Scissors, [i]In-between)? i
Let's play In-between!
DEBUG: 5 - 7
Choose a number (1-10): 0
Choose a number (1-10): 12
Choose a number (1-10): 6
Computer chose 5 and 7, you chose 6.
You win!
You now have 130 points.

Please enter a command [list, add, remove, play, winner,
quit]: list
|-------------------------------------------------------------|
| Player Name                    P  W  L  T  W-Rate Points    |
|-------------------------------------------------------------|
| Steve Jobs                     1  1  0  0  100.0%    130   |
| Lisa Smith                    16 13  2  1   81.2%    105   |
| Andrew Whittaker               7  0  7  0    0.0%     55   |
|-------------------------------------------------------------|

Please enter a command [list, add, remove, play, winner,
quit]: winner
Steve Jobs has 130 points and a winning rate of 100.0%.

Please enter a command [list, add, remove, play, winner,
quit]: play
Name: Lisa Smith
How many points to bid (1-105)? 120
How many points to bid (1-105)? 50
Which game ([r]Rock-Paper-Scissors, [i]In-between)? i
Let's play In-between!
DEBUG: 1 - 3
Choose a number (1-10): 3
```

```
Computer chose 1 and 3, you chose 3.
Tie!
No changes to your points.

Please enter a command [list, add, remove, play, winner,
quit]: list
|-------------------------------------------------------------|
| Player Name                    P  W  L  T  W-Rate Points    |
|-------------------------------------------------------------|
| Steve Jobs                     1  1  0  0  100.0%    130    |
| Lisa Smith                    17 13  2  2   76.5%    105    |
| Andrew Whittaker               7  0  7  0    0.0%     55    |
|-------------------------------------------------------------|

Please enter a command [list, add, remove, play, winner,
quit]: play
Name: Steve Jobs
How many points to bid (1-130)? 130
Which game ([r]Rock-Paper-Scissors, [i]In-between)? i
Let's play In-between!
DEBUG: 2 - 6
Choose a number (1-10): 9
Computer chose 2 and 6, you chose 9.
You lose!
Oh no! You ran out of points!

Please enter a command [list, add, remove, play, winner,
quit]: list
|-------------------------------------------------------------|
| Player Name                    P  W  L  T  W-Rate Points    |
|-------------------------------------------------------------|
| Lisa Smith                    17 13  2  2   76.5%    105    |
| Andrew Whittaker               7  0  7  0    0.0%     55    |
| Steve Jobs                     2  1  1  0   50.0%      0    |
|-------------------------------------------------------------|

Please enter a command [list, add, remove, play, winner,
quit]: play
Name: Steve Jobs
Not enough points to play.

Please enter a command [list, add, remove, play, winner,
quit]: winner
Lisa Smith has 105 points and a winning rate of 76.5%.

Please enter a command [list, add, remove, play, winner,
quit]: quit
Thank you for playing!

Players info successfully saved.
```

Sample Output 2 (in case players.txt is not available):

```
ERROR: Cannot load players info.

Please enter a command [list, add, remove, play, winner,
quit]: list
|-------------------------------------------------------------|
| Player Name                    P  W  L  T  W-Rate Points |
|-------------------------------------------------------------|
|                  No player to display.                      |
|-------------------------------------------------------------|

Please enter a command [list, add, remove, play, winner,
quit]: winner
There is no winner.

Please enter a command [list, add, remove, play, winner,
quit]: add
Name: Bill Gates
Successfully added player Bill Gates.

Please enter a command [list, add, remove, play, winner,
quit]: list
|-------------------------------------------------------------|
| Player Name                    P  W  L  T  W-Rate Points |
|-------------------------------------------------------------|
| Bill Gates                     0  0  0  0    0.0%    100 |
|-------------------------------------------------------------|

Please enter a command [list, add, remove, play, winner,
quit]: winner
Bill Gates has 100 points, and never played a game.

Please enter a command [list, add, remove, play, winner,
quit]: quit
Thank you for playing!

Players info successfully saved.
```

Marking Criteria

This assignment is worth 20% of your grade. Marking will follow a demo similar to the scenarios shown in the Sample Output section.

Category	Mark
Produces correct results	
Proper error message shown when cannot load data from `player.txt`.	/ 0.5
Proper success message shown after loading data from `player.txt`.	/ 0.5
`list` command prints a leader board.	/ 0.5
The leader board is printed in a correct format as specified.	/ 0.5
`add` command successfully adds a new player initialised with 100 points.	/ 0.5
`add` command properly fails if the player already exists.	/ 0.5
`remove` command successfully removes an existing player.	/ 0.5
`remove` command properly fails if the player does not exist.	/ 0.5
`play` command properly fails if the player does not exist.	/ 0.5
`play` command properly fails if the player has no points to play.	/ 0.5
`play` command asks for bidding with correct range of points available.	/ 0.5
Game of In-between correctly behaves as expected with the game logic.	/ 0.5
Game of In-between asks again if the user's choice is not between 1 to 10.	/ 0.5
Game of In-between prints output in a correct format as specified.	/ 0.5
The leader board is properly updated and sorted based on the results.	/ 0.5
`winner` command prints correct output in proper format.	/ 0.5
`winner` command prints no winner message when no players available.	/ 0.5
The leader board displays correct message when there is no player.	/ 0.5
`quit` command shows output of confirming the player info being saved.	/ 0.5
The player information is saved into `output.txt` in correct format.	/ 0.5
Sub-Total	**/ 10**
Adheres to specifications	
Uses the provided files without any modification.	/ 1
Class definitions organised into python modules as specified: `player.py`, `leaderboard.py`, and `inbetweengame.py`.	/ 1
`Player` class defines all of the specified methods.	/ 0.5
`LeaderBoard` class defines all of the specified methods.	/ 1
`Inbetween` class defines all of the specified methods.	/ 0.5
Proper use of classes and objects practising data encapsulation, inheritance and polymorphism, as specified.	/ 1
Appropriate and well-constructed `while`, `for`, and `if` statements.	/ 1
Sub-Total	**/ 6**
Coding style	
Comment at the beginning of each file with student details.	/ 0.5
Comment on code with sufficient details. (e.g., when defining a variable, function, or method, and for each significant section of code)	/ 0.5

Naming identifiers must use camelCase or under_score style.	/ 0.5
Meaningful names. (e.g., no single letter variable names, except for index)	/ 0.5
Sub-Total	**/ 2**
Demonstration - Explain the code answering questions asked by the supervisor.	
Question 1	/ 1
Question 2	/ 1
Sub-Total	**/ 2**
Total	**/ 20**

- The end -