CP1200 Assignment 2 - Choosem Racing

You are to plan and then code a number choosing game, as described by this information, sample output and screencast recording of the program working you have been provided with. This assignment focuses on the use of simple classes and GUI (Graphical User Interface).

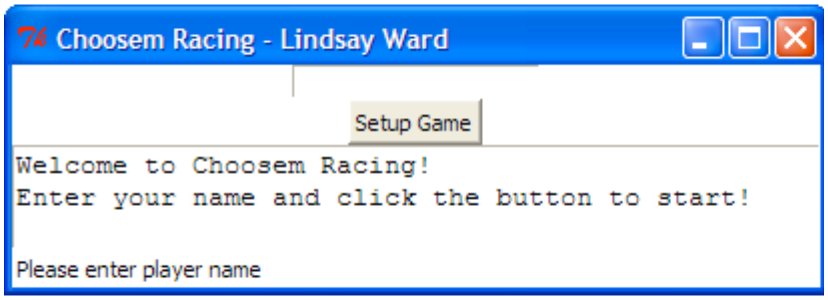
It's you versus two computer players in a race to the finish! Each round you choose a number between 1 and 3 and so do your two opponents. If a player's number is unique then they move forward that many steps. If another player also chooses the same number, neither of those players move. The first player to get to 10 or greater wins, but it is possible that 2 or 3 players could cross the line at the same time, in which case there will be multiple winners who share the glory.

You should make your program match the sample screenshots and screencast exactly (except for the names). Do not add or remove anything from the program even if you think it would make it better. You must aim to have yours look and work just like this including wording, layout – everything.

The following is a walk-through of the game, which should explain all of the details needed. Actual play (computer number and name choosing) is randomly determined.

When the program runs, the user is presented with a window containing:

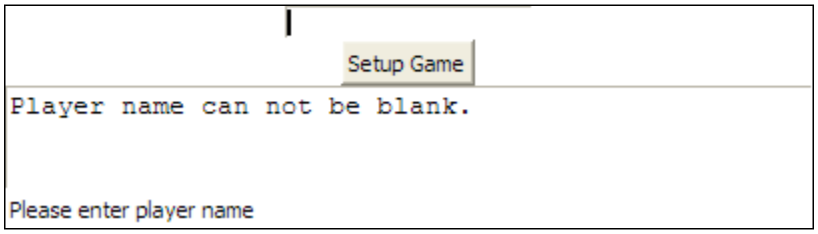
- The game and author's name in the title bar
- Text entry box for name
- Setup Game button
- 3-line text field
- Label at the bottom



```
Welcome to Choosem Racing!
Enter your name and click the button to start!
```

Please enter player name

If the user clicks Setup Game without completing the name field, the main text field shows an error message and nothing else happens.

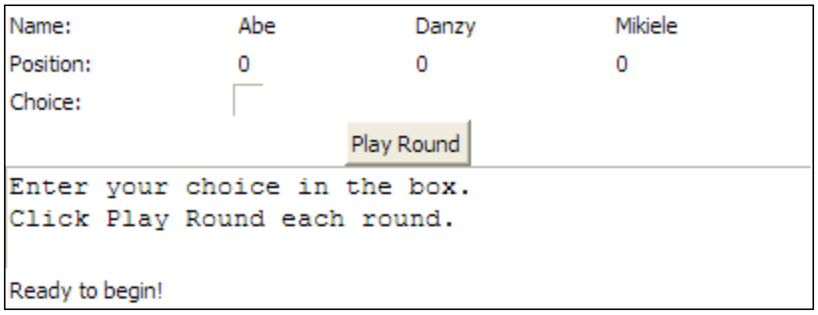


```
Player name can not be blank.
```

Please enter player name

When the user enters a name and clicks Setup Game, the GUI changes to:

- The top 3 rows have labels for player names, position (starts at 0) and choice
- The user's choice is a small text entry field
- The Setup Game button is replaced by a Play Round button
- The bottom label changes

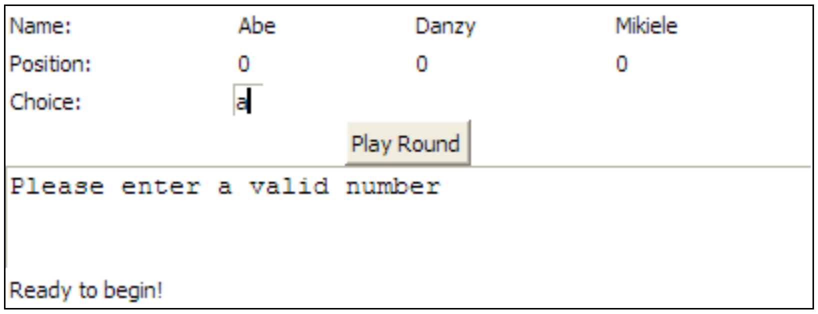


Name:	Abe	Danzy	Mikiele
Position:	0	0	0
Choice:			

```
Enter your choice in the box.
Click Play Round each round.
```

Ready to begin!

The user's choice must be a valid integer between 1 and 3 or an error message will appear when they press the Play Round button.
Check the screencast for all error messages.

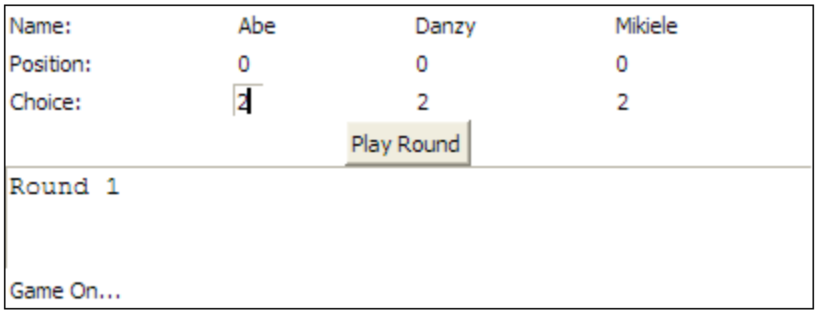


Name:	Abe	Danzy	Mikiele
Position:	0	0	0
Choice:	a		

```
Please enter a valid number
```

Ready to begin!

When a valid choice is entered and Play Round is pressed, anyone with a unique number (a number that no one else chooses) moves that much. In this example, everyone guessed 2 (what are the chances of that?*)

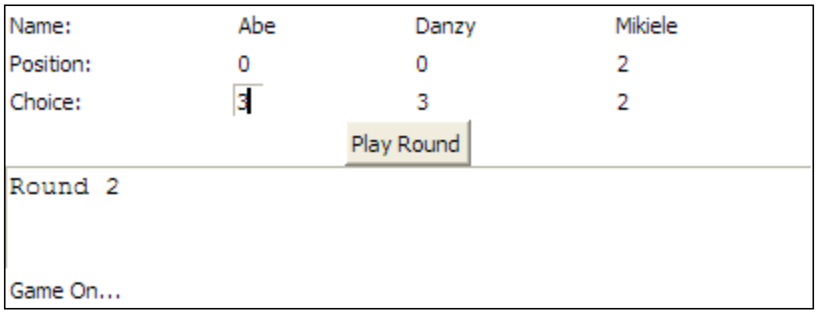


Name:	Abe	Danzy	Mikiele
Position:	0	0	0
Choice:	2	2	2

```
Round 1
```

Game On...

This example shows player 3 choosing the only unique number and moving forward 2.

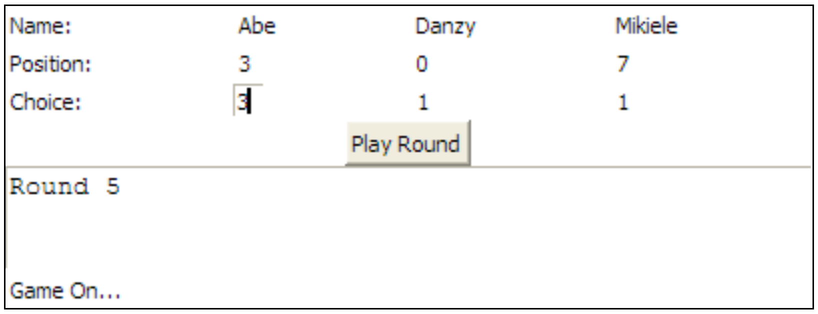


Name:	Abe	Danzy	Mikiele
Position:	0	0	2
Choice:	3	3	2

```
Round 2
```

Game On...

Yay, we finally got one (round 5)!

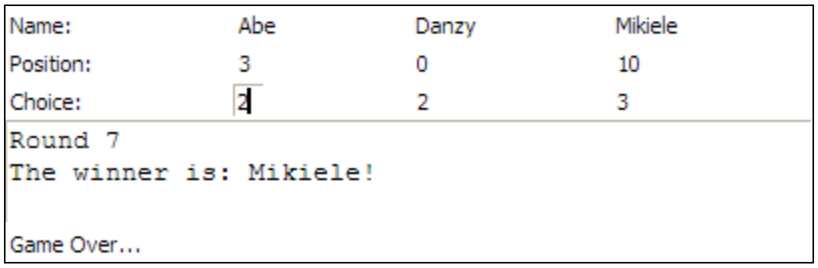


Name:	Abe	Danzy	Mikiele
Position:	3	0	7
Choice:	3	1	1

```
Round 5
```

Game On...

When someone "crosses the line" (10) first, they are celebrated with a special message in the main message box and the Game Over… appears in the status label.

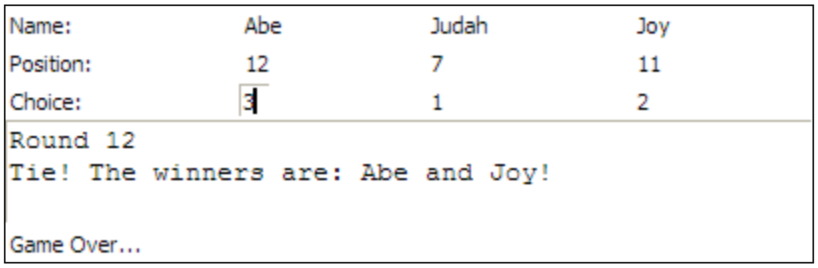


Name:	Abe	Danzy	Mikiele
Position:	3	0	10
Choice:	2	2	3

```
Round 7
The winner is: Mikiele!
```

Game Over...

This screenshot shows multiple people winning at once. If all 3 people cross the line together, the message is "Everyone wins!".



Name:	Abe	Judah	Joy
Position:	12	7	11
Choice:	3	1	2

```
Round 12
Tie! The winners are: Abe and Joy!
```

Game Over...

Implementation Details:

You must have two classes in your code (you are given starter files for each one):

Player

- This is the model and should contain data (name and position) for the player. A constructor header would look like:

```
def __init__(self, name="", position=0):
```

- Methods must include:
  - `get` and `set` for each variable,
  - `__str__` which would print like:
    ```
    Danzy   Position: 7
    ```
  - `__lt__` which should compare two players so that a player with a higher position amount is greater than another (this need not be used anywhere).
  - `choose` which randomly chooses and returns a number between 1 and 3.
  - `move` take a number as a parameter and increase the position by that. If this takes the player's position over the finish line, the function should return True, otherwise False.
  - This class file must have a `test()` function that only runs when the class is run directly. This should run through a simple (not full featured, no error-checking) text-only playing of the game. You may include multiple functions for this.
    Sample output for this follows (**bold** for user input).
  - For both the test program and the main GUI one, the computer player names are randomly chosen from a class constant list of (6 or more) names and should be unique.

```
Me   Position: 0
Gabriel   Position: 0
Danzy     Position: 0
Choice: 2
Me :  2
Gabriel :  2
Danzy :  3
Me   Position: 0
Gabriel   Position: 0
Danzy     Position: 3
Choice: 3
Me :  3
Gabriel :  2
Danzy :  3
Me   Position: 0
Gabriel   Position: 2
Danzy     Position: 3
Choice: 1
Me :  1
Gabriel :  1
Danzy :  2
Me   Position: 0
Gabriel   Position: 2
Danzy     Position: 5
Choice: 3
Me :  3
```

```
Gabriel :  2
Danzy :  2
Me   Position: 3
Gabriel   Position: 2
Danzy     Position: 5
Choice: 3
Me :  3
Gabriel :  1
Danzy :  3
Me   Position: 3
Gabriel   Position: 3
Danzy     Position: 5
Choice: 2
Me :  2
Gabriel :  1
Danzy :  1
Me   Position: 5
Gabriel   Position: 3
Danzy     Position: 5
Choice: 3
Me :  3
Gabriel :  1
Danzy :  2
Me   Position: 8
Gabriel   Position: 4
Danzy     Position: 7
Choice: 2
Me :  2
Gabriel :  3
Danzy :  2
Me   Position: 8
Gabriel   Position: 7
Danzy     Position: 7
Choice: 2
Me :  2
Gabriel :  2
Danzy :  3
Me   Position: 8
Gabriel   Position: 7
Danzy     Position: 10
The winner is: Danzy
```

GUIApplication

- This is the view/controller and contains the code to create the GUI widgets and respond to events in order to play the game.

You have been provided with skeleton code files for each class to help get you started.
You should include proper docstrings for all of your classes and functions.

Planning:

Write up the algorithms in pseudocode in comments at the top of each file. For each function, you need separate pseudocode as well as a function header that shows any parameters. Follow the style of the answers provided for tutorial 5.
You may show this part of the assignment to your tutor during practical time to get comments or suggestions. Algorithms must be concise, complete and well-formatted (e.g. indent control structures) as discussed and demonstrated in class, tutorial answers and the guide to good pseudocode.

Program Code:

You need to hand in a zip file containing two complete, functional Python (.py) code files containing appropriate comments (docstring with name, date, description at the top, planning docstring, and inline comments as appropriate).
Please name these **Player.py** and **GUIApplication.py** (these will be the names of your classes).
You can only get help from your tutor(s) in practical time after your prac work is finished, and only if you show your planning. **No help will be given on code without seeing your planning**.

Submission:

Your files need to be zipped up (so that the filenames remain intact for your importing to work).
This zip file should be submitted by uploading it in LearnJCU.
Please name your zip file **LastnameFirstnameA2.zip**

Due:

The assignment is to be submitted by the date and time specified on LearnJCU.
Submissions received after this date will incur late penalties as described in the subject guide.

*Post your answer to this seemingly rhetorical question on the LearnJCU discussion board.

Marking scheme follows on the next page…

Marking Scheme

Take note of the following marking criteria so that you know what needs to be done for marks.
Don't just try and get your program to work, but instead do proper planning followed by careful coding.
This marking scheme is subject to change.

Requirement	Marks	Out Of
Algorithm – Pseudocode – for each function (for both classes) (clear, complete, well-formatted, consistent, accurate)		**5**
Program Execution Player class: constructor, gets/sets		**2**
Player class: __str__ and __lt__		**2**
Player class: choose, move		**2**
Player class: test program		**2**
GUI Application: GUI widgets and layout		**1**
GUI Application: Error checking (name and choice)		**2**
GUI Application: Play round		**2**
GUI Application: Multiple winners		**2**
Randomly chosen unique names (for text and GUI versions)		**1**
Similarity to sample output/screenshots (including all formatting)		**1**
Quality of Code Use of appropriate, meaningful identifiers		**1**
Code readability: formatting, indentation, line spacing, etc.		**1**
Useful, descriptive comments (amount & quality – docstrings for classes and methods, at top and inline comments)		**3**
Use of own functions/methods		**2**
Class variables/constants		**1**
Deductions Inappropriate or sloppy code, literals used instead of constants, global variables, accessing "private" object variables directly		**-2**
Incorrect submission (must be 1 zip file containing 2 code files, properly named)		**-1**
Total		**30**