Homework #3 - Make a Crystal Ball Predictor

For this assignment, you will be writing a Crystal Ball class with the following:

- **A constructor (__init__) method:** The constructor will initialize a new crystal_ball object.
  - Set ***prediction_list*** to a passed list of possible predictions.
  - Set ***name_list*** to a passed list of possible names.
  - Set ***prediction_history_list*** to an empty list. This will hold the indices of all of the predictions that have been selected.
  - Set ***name_history_list*** to an empty list. This will hold the indices of all of the names that have been selected.
- ***__str__*** **method**: It should return a string with all of the predictions in **prediction_list** separated by commas, for example : " will eat lunch with, will fall in love with , must apologize to ." Hint: you can convert a list to a string using the str function.

```
Testing the __str__ method
['Is going to take a class with ', 'Will fall in love with ', 'Will spill on ']
```

- ***check_name*** **method:** The main function will prompt the user to type in a string name. This method takes as input the user inputted name and checks if it is already in the **name_list** and if so returns “I already have that name!”, otherwise it returns the prediction from the **predict** method**.**

- ***predict*** **method**: Takes as input the user inputted string name. Returns a

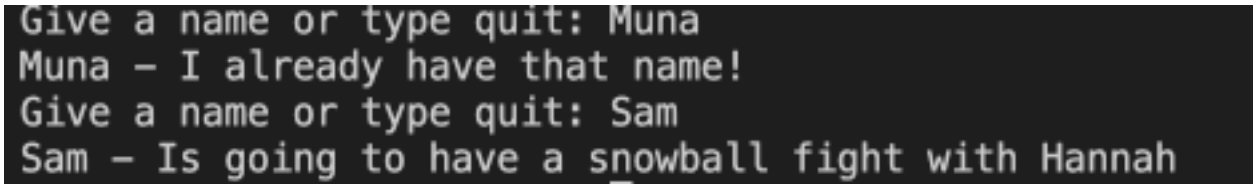
random prediction from the **prediction_list** and a random name from **name_list**. It randomly picks an index from 0 to the number of possible predictions minus one (hint: use the random module). It adds the index to the end of the **prediction_history_list**. Pick a random index from 0 to number of names in the **name_list** minus one and add the index to the end of **name_history_list**. Then add the inputted name to the **name_list.** It returns a string containing the prediction at the prediction index followed by the name at the name index.

● ***print_history*** **method**: Loops through the contents of the **prediction_history_list** and name_hist_list and prints each prediction and name with the prediction index in [ ] followed by the prediction at that index then "-" followed by the name index in [ ] and then the name at that index. Each prediction and name print on a separate line as shown below. It does not return anything. If there are no items in **prediction_history_list** it should print "None yet".

```
Printing the history
[2] Will spill on  - [0] Yasmeen
[4] Is going to have a snowball fight with  - [5] Muna
```

```
Printing the history when no answers have been generated yet
None yet
```

● ***main()*** **function**: Prompts the user to type in a name ex: “Hannah” or type “quit”. Loops until the user types “quit”. Getting a name from the user calls the **check_name** method, and prints the name, prediction, and a name from **name_list** prints the response as "input name - prediction other name" as shown below.

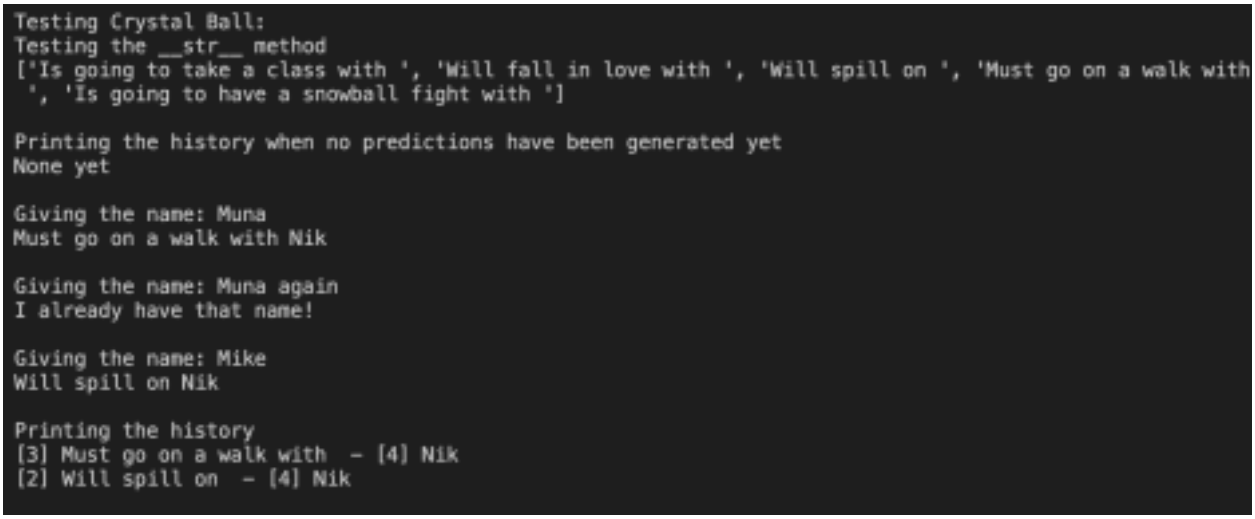


- Example output from HW.3

Sample output from main method:

```
Give a name or type quit: Muna
Muna - Is going to take a class with Ewelina
Give a name or type quit: Jack
Jack - Will spill on Muna
Give a name or type quit: Xinghui
Xinghui - I already have that name!
Give a name or type quit: Harry
Harry - Must go on a walk with Ewelina
Give a name or type quit: Knife
Knife - Will fall in love with Xinghui
Give a name or type quit: Maggie
Maggie - Will fall in love with Knife
Give a name or type quit: Maggie
Maggie - I already have that name!
Give a name or type quit: Hope
Hope - Is going to have a snowball fight with Anna
Give a name or type quit: quit
```

Sample output from the test method:



Grading Rubric - Total of 60 points

- 5 points - the __init__ method sets the object's prediction_list correctly to the passed prediction_list, set's the object's name_list correctly to the passed name_list and sets both the object's prediction_history_list and name_history_list to an empty list
- 5 points - the __str__ method returns a string with all predictions in prediction_list separated by commas : " will eat lunch with, will fall in love with , must apologize to "
- 5 points - the check_name method returns "I already have that name" if the name is already in the name_list

- 10 points - the predict method returns a random prediction and name and saves the index of the prediction at the end of the prediction_history_list and the index of the name at the end of the name_history_list
- 5 points - the predict_history function prints "None Yet" when there are no items in prediction_history_list.

- 10 points - print_history prints "[index] prediction - [index] name" for each of the prediction indices in the prediction_history_list and name_history_list in order and on a separate line.
- 10 points - the main() function loops until the user enters "quit" and each time asks the user for a name and prints the "name - response".

Extra Credit - 6 points

Implement the ***most_frequent*** method. It needs to find the most frequently chosen prediction index after telling n fortunes. It takes a number as an input: n, Ex: 200. Reset the **prediction_history_list** instance variable to the empty list, execute **prediction** n times, print how many times each index occurred, and print the most frequently occurring index. Choose any one of the top most common indices if there is a tie.