

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:** This method takes a **`name`** as input and returns a string. It checks if the **`name`** is already in the **`name_list`** and if so returns "I already have that name!", otherwise it returns the string returned from the **`predict`** method.

- ***predict* method**: Takes a name as input and returns a string. It randomly picks an index from 0 to the number of possible predictions minus one (hint: use the random module). It adds the index of the prediction to the end of the ***prediction_history_list***. It also picks a random index from 0 to the number of names in the ***name_list*** minus one and adds that index to the end of ***name_history_list***. Next, it adds the input name to the ***name_list***. It returns a string of the selected prediction and selected name as "*selected prediction selected name*".

- ***print_history* method**: Loops through the contents of the ***prediction_history_list*** and ***name_hist_list*** and prints "[prediction index] prediction at index - [name index] name at index". Each string prints 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". It calls the ***check_name*** method, and prints "input name - returned string from check_name" as shown below.

```
Give a name or type quit: Muna
Muna - I already have that name!
Give a name or type quit: Sam
Sam - Is going to have a snowball fight with Hannah
```

Sample output from the 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:

```
Testing Crystal Ball:
Testing the __str__ method
['Is going to take a class with ', 'Will fall in love with ', 'Will spill on ', 'Must go on a walk with ', 'Is going to have a snowball fight with ']

Printing the history when no predictions have been generated yet
None yet

Giving the name: Muna
Will spill on Ewelina

Giving the name: Muna again
I already have that name!

Giving the name: Mike
Must go on a walk with Nik

Printing the history
[2] Will spill on - [4] Ewelina
[3] Must go on a walk with - [5] Nik

Printing name_list
['Yasmeen', 'Xinghui', 'Elaina', 'Anna', 'Ewelina', 'Nik', 'Muna', 'Mike']
```

Grading Rubric - Total of 60 points

- 10 points - the `__init__` method sets the **object's prediction_list** correctly to the passed **prediction_list**, sets 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 "
- 10 points - the **check_name** method returns "I already have that name" if the name is already in the **name_list** and otherwise returns the result from calling the **predict** method.
- 10 points - the **predict** method returns a string with a random prediction and random 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**. It also saves the input name in the **name_list**.
- 5 points - the **print_history** function prints "None Yet" when there are no items in **prediction_history_list**.
- 10 points - **print_history** prints "[prediction index] prediction at index - [name index] name at index" 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 from check_name".

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. It resets the ***prediction_history_list*** instance variable to the empty list, executes ***prediction n*** times, prints how many times each index occurred, and prints the most frequently occurring index. Choose any one of the top most common indices if there is a tie.

```
Testing most_frequent method
0: 198
1: 186
2: 211
3: 222
The most frequent prediction after 1000 was: 3, Must go on a walk with
```