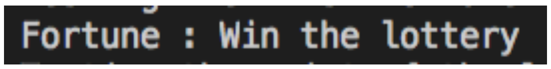
# Homework 3 - Make a fortune-teller

For this assignment, you will be writing a *Fortune_Teller* class with the following:

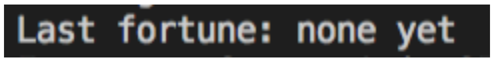
- A constructor (__init__) method: The constructor will initialize a new fortune-teller object from the passed list of all possible fortunes.
  - Set **fortune_list** to the passed list of all possible fortunes
  - Set **history_list** to an empty list. This will hold the indices of all of the fortunes that have been told.

- *tell* method: Returns a random fortune from the **fortune_list** by randomly picking an index from 0 to the number of possible fortunes minus one (hint: use the random module). Add the index for the fortune to the end the **history_lis**t. Return the fortune for that index (not the index).

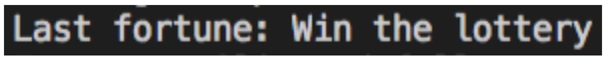


```
Fortune : Win the lottery
```

- *__str__* method: If no fortunes have been told yet, it should return "Last fortune: none yet" otherwise it should return "Last fortune: *the last Fortune*" (not the last index) as shown below.

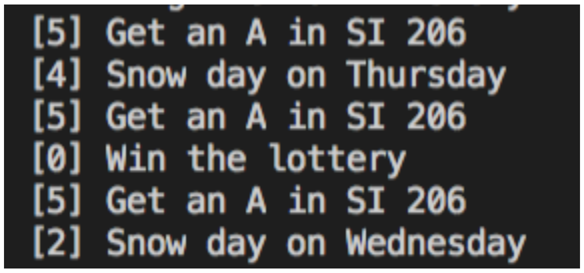


```
Last fortune: none yet
```



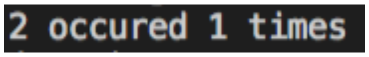
```
Last fortune: Win the lottery
```

- *print_history* method: Prints the content of the history list with the index number in [] and each fortune on a separate line



```
 [5] Get an A in SI 206
 [4] Snow day on Thursday
 [5] Get an A in SI 206
 [0] Win the lottery
 [5] Get an A in SI 206
 [2] Snow day on Wednesday
```

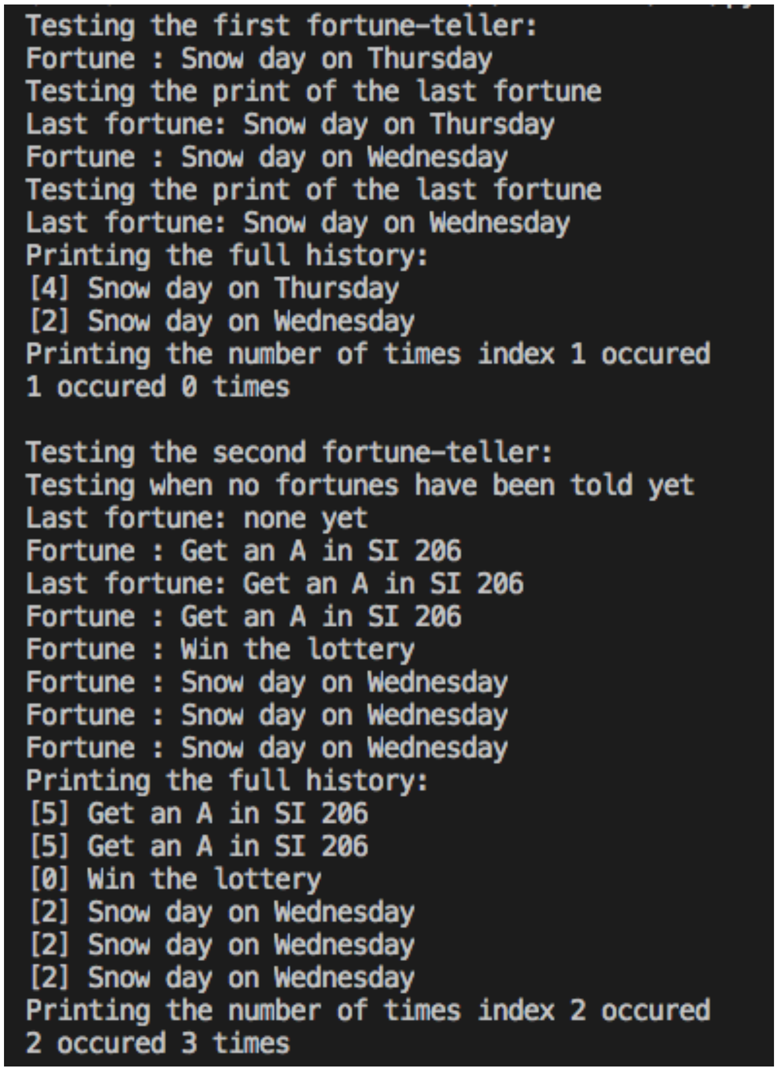
- *print_count_for_answer* method: Takes in a parameter *num* which specifies which index to look for. Prints the number of times that index occurs in the history_list.



```
2 occured 1 times
```

## Example Output From HW3.py

NOTE: Your output will not look *exactly* like this because we are using *random* and can't predict what it will return.



```
Testing the first fortune-teller:
Fortune : Snow day on Thursday
Testing the print of the last fortune
Last fortune: Snow day on Thursday
Fortune : Snow day on Wednesday
Testing the print of the last fortune
Last fortune: Snow day on Wednesday
Printing the full history:
[4] Snow day on Thursday
[2] Snow day on Wednesday
Printing the number of times index 1 occured
1 occured 0 times

Testing the second fortune-teller:
Testing when no fortunes have been told yet
Last fortune: none yet
Fortune : Get an A in SI 206
Last fortune: Get an A in SI 206
Fortune : Get an A in SI 206
Fortune : Win the lottery
Fortune : Snow day on Wednesday
Fortune : Snow day on Wednesday
Fortune : Snow day on Wednesday
Printing the full history:
[5] Get an A in SI 206
[5] Get an A in SI 206
[0] Win the lottery
[2] Snow day on Wednesday
[2] Snow day on Wednesday
[2] Snow day on Wednesday
Printing the number of times index 2 occured
2 occured 3 times
```

## Grading Rubric - total of 60 points

5 points - the __init__ method sets the object's **fortune_list** correctly (the instance variable)
5 points - the __init__ method sets the object's **history_list** to an empty list
10 points - the *tell* method correctly picks a random index between 0 and the number of fortunes in the fortune_list minus one
5 points - the *tell* method saves the picked index at the end of the **history_list**
5 points - the *tell* method returns the fortune at the picked index in the **fortune_list**
5 points - the __*str*__ method returns a string "Last fortune" with the text of the last fortune
5 points - the __*str*__ method returns a string telling the user "Last fortune: none yet" if there haven't been any calls to tell yet
10 points - *print_count_for_num* correctly prints the number of times an index occurs in the **history_list**
10 points - *print_history* prints "[index] fortune" for each of the fortunes in the **history_list** in order and on a separate line.

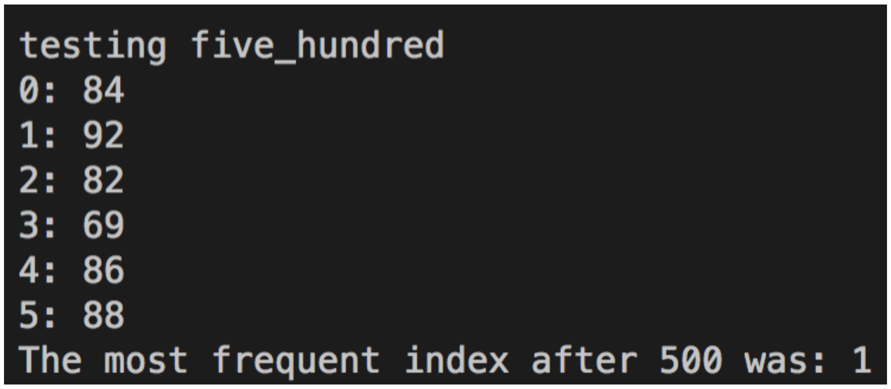
This grading rubric shows how you will gain points, but not all the ways you could lose points.

## Extra Credit - 6 points

Implement the following method:

*five_hundred* method: Finds the most frequently chosen index after telling 500 fortunes. In this method, reset the **history_list** instance variable to the empty list, execute tell 500 times, print how many times each index occured, and print the most frequently occurring index. Choose any one of the top most common indices if there is a tie.

## Extra Credit Example Output:



```
testing five_hundred
0: 84
1: 92
2: 82
3: 69
4: 86
5: 88
The most frequent index after 500 was: 1
```