CSCI 136 Assignment – Mind Reader

February 8th, 2023

Due Date: Sunday, 2/12/2023 11:59PM

Points: 35

Topic: Dictionaries, Hash Maps, Associative Arrays

In this assignment, you will use a hash map (dictionary) to implement a game you can play against the computer. The interesting part of this is that the computer typically wins.

Your assignment is to write a program called `MindReader.py` that implements the game.

After following all instructions below, please submit your Python script (`MindReader.py`) and a writeup file (`MindReader.docx`) in Moodle. Upload these as individual files – do not zip them. Be sure the script file has header sections that include the file name, your name, credits, a description of the file, the class and other useful information. Make sure your classes and all methods are thoroughly commented.

- `MindReader.py`
- `MindReader.docx`

Grading:

This assignment is worth 35 points. You will be graded according to the following criteria:

Grade Item	Points
Program Compiles	2
Program Runs	2
Comments on All Classes and Methods	6
Dictionary Stores Keys/Values Correctly	6
Computer Prediction is Based on Patterns	4
Game Operates Correctly	5
Write Up of Results Covering 10 Test Runs	10
Total	35

The Mind Reader Game

Prior to user (player) input in each iteration, the computer (program) will predict whether a player selects “heads” or “tails”, as if a coin has been tossed. (A coin is not tossed; the player makes the choice using whatever strategy or intuition they think works.) Without being told what the program predicted/guessed, the player will then choose “heads” or “tails”. The program’s guess and the player’s input will then be compared. If the program guessed correctly, it gets a point. If not, the player gets a point. The program continues until either the program or the player reaches 25 points.

The program will keep track of past behavior of the player, and over time, build a "history" of what the player has input. Suppose the player has made and entered the following 10 choices in this order:

```
H H T H T T T H T T
1 2 3 4 5 6 7 8 9 10
```

The program tabulates the choices made after each sequence of four consecutive picks within the overall sequence of picks and subsequently uses this to predict player behavior. With the example sequence of 10 choices shown above, the program will have encountered the following sequences of four. Notice that the seventh sequence of four is the same as the third sequence of four.

```
1. HHTH
2. HTHT
3. THTT
4. HTTT
5. TTTH
6. TTHT
7. THTT
```

For each unique sequence of four entries, the program tabulates over the choice the player makes immediately after that sequence, either heads or tails, and how many times the player made each choice after that sequence. You could use a 2-element (integer) list with the first element representing the number of times "H" was chosen and the second representing the number of times "T" was chosen for this. Then the tabulation would be as follows. Note that the tabulation of THTT will be updated with the choice.

```
HHTH -> 0 1
HTHT -> 0 1
THTT -> 0 1
HTTT -> 1 0
TTTH -> 0 1
TTHT -> 1 0
```

Using this tabulation, the one and only time the player chose the series of `THTT`, the next choice was `T` (corresponding to the 0 1 tabulation). So, the program will guess that the player will input a `T` next. If the player actually does choose `T` next, then the tabulation will be updated to:

```
THTT -> 0 2
```

If the player enters this same pattern again, but this time their next choice is `H`, the tabulated values would be update to:

```
THTT -> 1 2
```

If a sequence is not yet in the dictionary, the program should make a random guess. The program should also make a random guess if the number of times `H` or `T` was chosen is equal in the tabulation dictionary.

You should use a (Python built-in) dictionary to store the tabulations: the input patterns are the keys, and the value is a list with two components used to keep track of the number of times H and T were chosen by the player after that key pattern was used. You should also think about what kind of structure would work best for keeping track of player choices as they are input. In this program, it is up to you on how to define your methods. You are not given a specification to follow.

Example Run and Output

```
(Instructions for playing …)

...

Your Turn.
Please enter H for heads or T for tails: H
The computer predicted H and the player chose H.
One point for the computer!
Computer: 1 , Player: 0

Your Turn.
Please enter H for heads or T for tails: T
The computer predicted H and the player chose T.
One point for the player.
Computer: 1 , Player: 1

...

Your Turn.
Please enter H for heads or T for tails: T
The computer predicted T and the player chose T.
One point for the player.
Computer: 25 , Player: 3

Computer WINS!!!
```

MindReader.docx (writeup)

Run your program 10 times, and report the outcomes. You may even have a friend run your program. Make sure though that you are consistent in who is running the program for the writeup. How many times did the program win and how many times did the player win? Is the result what you expected? Why?

Write up your approach to the problem, the results you obtained, and your conclusions in a short report.

Helpful hints:

Work incrementally. Build and test incrementally, starting with easier aspects. Add functionality incrementally, testing with each change.

No test script is provided for this lab. The game is dependent on user input.

There is no specification for the Mind Reader program. You may define the methods and variables you need. Be sure to document them.

You should use Python's built-in Dictionary object for this assignment.

Test and test again. Test thoroughly. Have someone else try to use your program and observe their usage. Fix accordingly.

Submit Your Work

Your assignment is to write a program called `MindReader.py` that implements the game.

After following all instructions above, please submit your Python script (`MindReader.py`) and a writeup file (`MindReader.docx`) in Moodle. Upload these as individual files – do not zip them. Be sure the script file has header sections that include the file name, your name, credits, a description of the file, the class and other useful information. Make sure your classes and all methods are thoroughly commented.

- `MindReader.py`
- `MindReader.docx`

EXTENSIONS

You will not submit the answers to these extensions as part of your assignment submission. You should work through and understand them to learn more and practice. You may be held responsible for the content within, so you really should work through them. With that said, this are outside the scope of what will be submitted for this lab. Do not change your original submittal to address these extensions.

1. In addition to or in the absence of an exact pattern match, the program could use other patterns that are "similar" to make predictions. Loosely, this approach might be considered as case-based reasoning. How would you implement this?

2. There is a relatively simple way we could have hashed (encoded) sequences of four H/T values in our program that results in unique integer values for each length four sequence. The integer values could then be used as indexes for a list that stores the counts. How would this be done?

3. Could the same approach be used to encode sequences of varying and virtually unlimited length? How would that be accomplished?

4. What is special about sequences of length four? Could sequences of other lengths be used?

5. Can you beat the program? Not just once or twice, but regularly? What is the best you can hope to do?

6. Investigate how Python implements dictionaries. (It uses hash tables to do this.) How is the hash function handled? How are collisions handled? How is storage done?