

## 15-110 Hw5 - Programming Portion

These problems should be solved in the starter file available on the course website. Submit your code to the Gradescope assignment Hw5 - Programming for autograding.

All programming problems may also be checked by running the starter file, which calls the function `testAll()` to run test cases on all programs.

Programming is not a core part of Unit 3. Therefore, we will use this assignment to prepare you for Hw6. You will write five functions and these functions will work together to create an interactive, text-based game of **Memory**.

In the game Memory you start with a deck of matched cards. Place the cards face-down on the table. In each turn you get to flip over two face-down cards. If they match, they stay face-up; if they don't, you have to flip them face-down again. The goal is to flip all the cards face-up in as few turns as possible. If you've never played Memory before, try it out here: <https://www.helpfulgames.com/subjects/brain-training/memory.html>

The first three functions can be solved with just the skills you've learned in prior units; we encourage you to write these early. Functions #4 and #5 will be easier to solve after the Managing Large Code Projects lecture.

### #1 - `generateBoard(words)` - 3pts

First, we need to set up the initial board of cards. Each card is represented by a two item list: the first item is the card's value (a string), the second is a Boolean representing whether the card is face-down or face-up (`False` for face-down, `True` for face-up).

Write the function `generateBoard(words)` which takes a list of strings and returns a board (a 2D list holding cards, which are 1D lists). The function should create two cards for each of the words in the list, both with initial flipped values of `False` (since we want all the cards in our game to begin face-down). Once all cards have been added to the board, shuffle the board so the cards are in a random order. Use the built-in function `random.shuffle(list)` which destructively shuffles the list (returning `None`) to do this.

For example, if we call `generateBoard(["dog", "cat"])`, it might return `[ ["dog", False], ["cat", False], ["cat", False], ["dog", False] ]`, though the inner lists could also be in a different random order.

## #2 - displayBoard(board) - 4pts

Next, we need to be able to display the board to the user. We'll display each card as its value if it is face-up, or the string "???" if it is face-down.

Write the function `displayBoard(board)` which takes a board (a 2D list containing cards as described above in `generateBoard`) and prints the board, returning `None`. Each card on the board will be printed on a separate line. The line should contain the index of the card and the card's value (or `???` if it is face-down). You can format the printed line however you like as long as those two pieces of information are included and each card is printed on a separate line.

For example, if we call `displayBoard([ ["dog", False], ["cat", True], ["cat", False], ["dog", True] ])`, (the list above, but with the first "cat" and second "dog" face-up), it might print:

```
0: ???  
1: cat  
2: ???  
3: dog
```

## #3 - getFlippedCards(board) - 3pts

We'll need to determine which cards on the board have been flipped to tell when the game is over. Write the function `getFlippedCards(board)` which takes the board (a 2D list containing cards as described above) and returns a 1D list of integers. This function should go through each card on the board and add that card's **index** in the board list to a result list if the card is face-up. The list of indexes should be returned at the end.

For example, if we call `getFlippedCards([ ["dog", False], ["cat", True], ["cat", False], ["dog", True] ])` it will return `[ 1, 3 ]`, because the first and third elements are face-up. If no cards are face-up, just return an empty list.

## #4 - pickIndex(board) - 10pts

To make the game interactive we'll let the user choose which cards to flip over on each turn by typing in the indexes of the cards they want to flip into the interpreter. We'll have to make sure to handle bad user inputs without crashing the program!

Write the function `pickIndex(board)` which takes a board (a 2D list containing cards as described above) and returns the index of the card to be flipped (an int). The function should ask the user to pick a card index. If the index is not an integer, or not in the range of the board, or is the index of a card that has already been flipped, the function should print out an appropriate error message and ask the user for an index again. When the user enters a valid index, that index should be returned as an integer.

For example, if we call `pickIndex([ ["owl", False], ["dog", True], ["cat", False], ["cat", False], ["owl", False], ["dog", True] ])` we might end up with the following interaction. User input is bolded.

```
Pick a card index: test
That's not an integer!
Pick a card index: -1
Out of range. Pick a valid card.
Pick a card index: 6
Out of range. Pick a valid card.
Pick a card index: 1
You've already flipped that card. Pick a different card.
Pick a card index: 2
```

The function would then return the integer 2. Your prompts and error messages don't have to be the same as ours, but they should exist and provide relevant information.

We will manually grade this problem. To test it yourself, run `testPickIndex()` and try entering non-integers, numbers out of range, and indexes that have already been flipped, as shown above.

**Hint:** use the board to help determine what is in range and whether the chosen index has already been flipped.

**Hint 2:** if you're stuck, look at our tic-tac-toe example from the Managing Large Code Projects lecture. How did we implement taking turns?

## #5 - playMemoryGame(words) - 10pts

Now we finally have everything we need to actually implement the game! You must use the four functions written above as **helper functions** to streamline the game code.

Write the function `playMemoryGame(words)` which takes a list of words (1D list of strings), sets up a memory game based on those words, and runs the game with the user until all cards have been flipped. Let's break this down into smaller steps.

- A. First, welcome the player to the game and generate a new board based on the cards and show the starting board to the user. You'll want to call `generateBoard` and `displayBoard` for this.
- B. Second, set up a loop that continues until the game is finished. The game is done when every card in the board has been flipped. To check this, look at whether `getFlippedCards` contains every card in the board or not. When the game is over, congratulate the user on winning.
- C. Third, in each iteration of the loop, have the user pick indexes for two cards that they want to flip face-up. After a valid index is picked the card at that index on the board should have its flipped value set to `True` and the board should be displayed again. You'll need to do this **twice**- once for the first card and again for the second. You'll want to call `pickIndex` and `displayBoard` here.
- D. Fourth, check whether the two cards that were flipped match by comparing their values. If they match, leave them face-up; if they don't match, set both flipped values to face-down (`False`) again. In either case, print out an appropriate message to the user, then print some kind of line break to show that the next turn has begun.
- E. Finally, keep track of the number of turns the user takes to clear the board. A turn is a single iteration of the loop (flipping two cards and seeing if they're a match). You should print out the number of moves taken at the end of the game.

Once this function is finished, you're done! Play your game to test out how it works.

If we call `playMemoryGame([ "dog", "cat" ])`, here's an example of what a full game might look like. User inputs have been bolded. Your game prompts don't need to be identical to ours, but they should convey the same information.

Welcome to Memory!

0: ???

1: ???

2: ???

3: ???

Pick a card index: 0

0: dog

1: ???

2: ???

3: ???

Pick a card index: 2

0: dog

1: ???

2: cat

3: ???

Try again!

---

Pick a card index: 1

0: ???

1: cat

2: ???

3: ???

Pick a card index: 2

0: ???

1: cat

2: cat

3: ???

Good guess!

---

Pick a card index: 0

0: dog

1: cat

2: cat

3: ???

Pick a card index: 2

You've already flipped that card. Pick a different card.

Pick a card index: 3

0: dog

1: cat

2: cat

3: dog

Good guess!

---

Good game! You took 3 moves to clear the board.