Memory Game

Write a program that has a user play a memory game using lists.

This Python programming assignment will guide you through the exciting process of creating your own memory game, a classic and engaging exercise that tests your ability to remember and match pairs of items. In this game the cards will be represented as numbers in a list and their location will be indices of the list. By completing this project, you will practice critical list concepts. Not only will this enhance your problem-solving skills, but you will also have a lot of fun as you watch the game come to life and challenge yourself or your friends to find all the pairs.



*This game is inspired by the Memory Card Game, pictured above.*

**IMPORTANT**

* Like Khansole Academy, **the autograder expects you to match the Expected Output exactly.** If you're stuck, [SEE COMMON BUGS HERE](/cip5/info/memory-game-tips).
* **NOTE:** The autograder stops as soon as it sees an incorrect output. So, the Expected and Observed outputs **will only show lines up to where you failed. Use the demo to see *everything* you should print.**
* Here is a demo of a fully implemented solution. Play it so you understand how the game works![Memory Game Demo](/cip5/share/JIGHUopbwtmf2CyQGU8s)
* Use the **Demo** and the **Example (at the bottom of this page)** as references for **exactly how your output should look.**

**Milestone #1**: Create the truth list.

Use a for loop to create a list that has the values 0 through NUM\_PAIRS - 1 twice. So for example if NUM\_PAIRS = 3 as in the example, your list should have this value:

A white rectangular object with a black border

AI-generated content may be incorrect.

**Milestone #2**: Shuffle the list.

The python random library has a special function called shuffle which will randomly order the values in a list. Call that function on your list and then print out the result. Print out the list to make sure it looks shuffled! Assuming that your list from the previous milestone was named truth

A close-up of a computer screen

AI-generated content may be incorrect.

Note: Do not keep the print statement in your final program! It's just for testing.

**Milestone #3:** Create a displayed list

In this program you will need to keep track of a separate list, which is the one to display to the user. We use the '\*' symbol to denote that a value is hidden to the user (otherwise we display the value). Initially displayed should be a list of '\*' values which is the same length as truth. If NUM\_PAIRS = 3:

A group of black and blue symbols

AI-generated content may be incorrect.

**Milestone #4**: Get two valid indices from the user

Write logic to get two valid indices from the user. A valid index satisfies the following:

* A valid number, greater than or equal to 0 and less than or equal to the index of the last element in the list.
* The value of the **displayed** list at that index must be '\*' (i.e. it has not yet been uncovered/matched).
* The second index the user supplies in the pair must not be equal to the first (i.e. the two indices supplied must be distinct)

If the user does not enter a valid index, repeatedly prompt them until they do. We recommend that you define a new function get\_valid\_index which takes in the displayed list and returns the valid index the user entered.

Here are specific strings we are expecting for you to print out when the user doesn't enter a valid index:

* **Entering the same index twice**
  + "You entered the same index twice. Try again."
* **Number is out-of-bounds**
  + "Invalid index. Try again."
* **Non-number.**
  + "Not a number. Try again."
* **Inputting an index that has already been revealed**
  + "This number has already been matched. Try again."

**Milestone #5:** Check correct.

Get two valid inputs from the user and check if there is a match (the values of the indices in the truth lists are equal).

* If the values **don't match:**
  + Do not update the displayed list.
  + Print out the values at the two guessed indices.
  + Print "No match. Try again."
  + Use input() to wait for the user to press the "Enter" key before continuing
* If the values **do match:**
  + Update the displayed list
  + Print "Match!"
  + **Do not** print the values at each guessed index or wait for the user to press "Enter" to continue, as is done if there is not a match

Imagine the current value of displayed and truth lists are as follows:

A group of text on a white background

AI-generated content may be incorrect.

Here is an example turn where the user does not get a match (user input is in blue):

A screen shot of a computer

AI-generated content may be incorrect.

Here is an example turn where the user does get a match (user input is in blue):

A black screen with blue and white letters

AI-generated content may be incorrect.

Since the user got a match the display list will be updated to [2, '\*', '\*', 2, 0, 0]

**Milestone #6:** Play multiple turns.

The user should play until all the pairs have been correctly located. Between turns make sure to call clear\_terminal. At the end of the game, tell the user that they have won by printing 'Congratulations! You won!'

Each turn looks like this:

1. Print out the display array
2. Get two valid indices from the user
3. Determine if they indices have matching values in truth\_list
4. If not a match, print the values, "No match. Try again.", and wait for the user to Press Enter. Once they do, **call** clear\_terminal()
5. If it is a match, print "Match!" and **call** clear\_terminal(). (No waiting for the Enter key).

When the game is won:

1. Print out the display array (which will now be fully uncovered)
2. Print 'Congratulations! You won!'

**Example Game**

Here is a (very convenient) full game. User input is in (blue). **Remember to match this output format exactly**

Imagine that we have:

A screenshot of a computer program

AI-generated content may be incorrect.