

CS Scholars - Programming Hw4 [Optional]

Due Date: Thursday 07/27 EOD

This assignment is **optional**, for students who want to challenge themselves by building a full interactive game in Python. For full credit, complete #1, #2, or #3.

There is no starter file for this assignment. You can start from a blank Python file for Hangman, or the interaction starter code for Memory Game or Snake.

[#1 - Hangman](#)

[#2 - Memory Game](#)

[#3 - Snake](#)

#1 - Hangman

Can attempt after the Text-Based Interaction lecture

Hangman is a simple two-player game where one player comes up with a mystery word and the other player guesses letters in the word. If the letter is in the word, the first player writes it in all the places where it appears; if the letter is not in the word, the first player draws part of a stick figure. If the second player guesses all the letters in the word, they win; if they guess too many wrong letters and the full stick figure is drawn, they lose.

Write a function `playHangman(word)` which takes a string (the word to guess) and lets the user play an interactive game of Hangman. The game should implement everything described above, apart from drawing a stick figure; instead, start the user at 6 incorrect guesses available and remove one for each incorrect guess.

Hint: this is a complex problem. You'll want to use **top-down design** to make it easier!

Note: to give you an idea of what you might want to include, here's an example of a full game of Hangman played in Python:

```
Current word: _ _ _ _ _ _ _ _ _ _
Incorrect guesses left: 6
Letters guessed:
Which letter do you want to guess: e
Not quite...
-----
Current word: _ _ _ _ _ _ _ _ _ _
Incorrect guesses left: 5
Letters guessed: e
Which letter do you want to guess: o
Good guess!
-----
Current word: _ _ o _ _ _ _ _ _ _
Incorrect guesses left: 5
Letters guessed: e, o
Which letter do you want to guess: i
Good guess!
-----
Current word: _ _ o _ _ _ _ _ i _ _
Incorrect guesses left: 5
Letters guessed: e, o, i
Which letter do you want to guess: n
Good guess!
-----
```

Current word: _ _ o _ _ _ _ i n _

Incorrect guesses left: 5

Letters guessed: e, o, i, n

Which letter do you want to guess: g

Good guess!

Current word: _ _ o g _ _ _ _ i n g

Incorrect guesses left: 5

Letters guessed: e, o, i, n, g

Which letter do you want to guess: y

Not quite...

Current word: _ _ o g _ _ _ _ i n g

Incorrect guesses left: 4

Letters guessed: e, o, i, n, g, y

Which letter do you want to guess: as

Please enter only one letter.

Which letter do you want to guess: a

Good guess!

Current word: _ _ o g _ a _ _ i n g

Incorrect guesses left: 4

Letters guessed: e, o, i, n, g, y, a

Which letter do you want to guess: m

Good guess!

Current word: _ _ o g _ a m m i n g

Incorrect guesses left: 4

Letters guessed: e, o, i, n, g, y, a, m

Which letter do you want to guess: i

You already guessed that letter! Pick a different letter.

Which letter do you want to guess: t

Not quite...

Current word: _ _ o g _ a m m i n g

Incorrect guesses left: 3

Letters guessed: e, o, i, n, g, y, a, m, t

Which letter do you want to guess: r

Good guess!

Current word: _ r o g r a m m i n g

Incorrect guesses left: 3

Letters guessed: e, o, i, n, g, y, a, m, t, r

Which letter do you want to guess: p

Good guess!

Final word: p r o g r a m m i n g

You won in 11 turns.

#2 - Memory Game

Can attempt after the Event-Based Interaction lecture

Memory is a simple one-player game where the player starts with a small deck of cards (where each card appears twice in the deck), shuffles the deck, and places all the cards face-down. The player then takes turns, where in each turn they flip two cards face-up. If the cards match, they are left face-up; if they do not match, they are flipped face-down again. The goal is to flip all cards face-up in as few turns as possible.

Using the interaction framework from class, implement a graphical version of the Memory Game. The game should be played using 16 cards where the 'face' of the card shows a word (so 8 unique words overall). The cards should be displayed in a 4-by-4 grid, and the user should be able to 'flip' a card by clicking on it.

Cards should automatically flip face-down again when two cards have been flipped up which don't match. However, this should not happen immediately - let the user see the value of the second card first, then anywhere to flip them face-down again.

Hint: you may want to represent cards with two values: the word on the face of the card and whether or not it is currently face-up.

Another Hint: it may help to distinguish which cards are 'temporarily' face-up as opposed to permanently face-up.

Here is an example of what a Memory Game could look like mid-game. In this game, the user just flipped the 'bird' and 'rabbit' cards; they will flip face-down again on the next click.

tk			
		pig	
bird		snake	
	pig		
	rabbit		snake

#3 - Snake

Can attempt after the Time-Based Interaction lecture

Using the interaction framework from class, create a simple version of the classic arcade game Snake!

If you've never played Snake before, start by learning how the game works. Google has a simple version you can play: <https://www.google.com/search?q=snake+game>

Your Snake game should be played on an 8 x 8 grid. The snake can be represented as a connected sequence of green cells; the food can be represented as a red circle in a cell.

Your game should do the following:

- Display the grid, food, and snake.
- Automatically move the snake one cell in the 'forward direction' over a regular time interval (we recommend setting the time rate to 0.4 for a playable game).
- Allow the user to change the 'forward direction' by pressing the up, down, left, and right arrows.
- Allow the snake to 'eat' the food when the head of the snake overlaps the food (by extending the snake body by one length and placing the food in a randomly-chosen empty cell on the grid).
- End the game when the head of the snake goes off the board or intersects with the body of the snake. The user should no longer be able to play the game at this point.

Hint: the hardest part of Snake is determining how to move the body. Here's a handy trick! Represent the snake as a list of body cell positions ([row, col] lists). To move the snake forward, just add a new cell (the new head) to the front of the list, and remove the cell at the back of the list! If you need to 'grow' the snake, just add the head without removing the last cell.

Here's an example of what the board might look like at the middle of the game:

