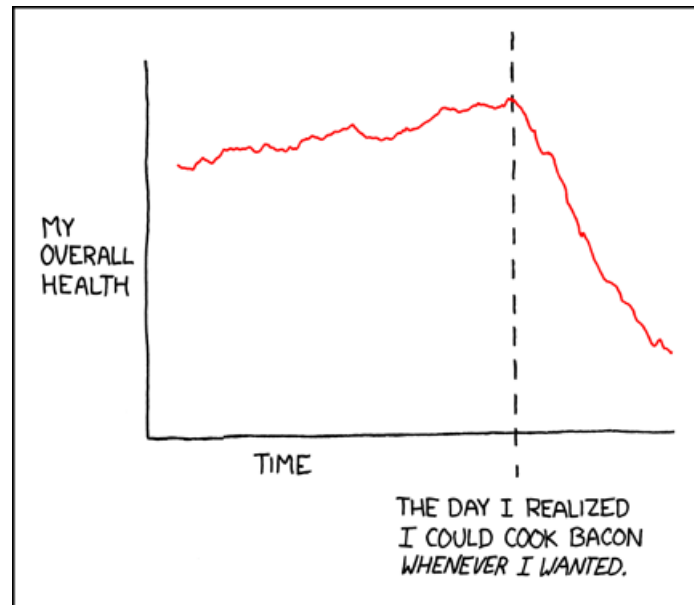


CS50—Assignment 4

Chatbot

Due: Monday, Sept. 28, at 11:59am



<https://xkcd.com/418/>

For this assignment, we're going to be developing a basic chatbot, which will respond to user questions. Making an actual chatbot, particularly one that is able to make good responses in a range of situations, is hard, so don't expect ours to be perfect!

Introduction to our Movie Quotes (Group)

The key to our chatbot will be a dataset of over 300K spoken lines from 617 movies¹. I've preprocessed this dataset to create a collection of 244K pairs of quotes from movies that contains a first quote by one character and then the response from a second character.

First, let's get access to the movies quotes:

- Take the folder containing this PDF and put it into your CS50 folder. Open the **assignment4** folder in VS Code.

¹The quotes are a processed version of the data from: <https://www.kaggle.com/Cornell-University/movie-dialog-corpus>

- Add your name and the assignment number in comments to the top of `assign4.py`.
- Double check that your project folder has three files in it: `assign4.py`, `assign4_quotes.py`, and `movie_quotes.txt`. If you're curious, you can open this last file and you can see the pairs of quotes.
- Finally, let's access the quotes in our program. Import the helper functions for this assignment by adding the following into your `assign4.py` file below the header comments:

```
from assign4_quotes import *
```

There are two functions in the `assign4_quotes` module: `get_quotes()` and `get_practice_quotes()`. The first returns the real quote data and the second returns some sample data that may be useful for debugging your programs.

In both cases, the functions return the quotes data as a *list of tuples* (specifically, a list of pairs) of *strings*. In each tuple/pair the first element is the first line of a dialog and the second element is the response. For example, the practice data has five pairs:

```
>>> get_practice_quotes()
[('quote1', 'quote2'), ('first', 'second'), ('first they said this', 'then this'),
 ('what?', "that's what"), ('what?', "now you've it!")]
```

We can look at the first pair of real quotes:

```
>>> quotes = get_quotes()
>>> quotes[0]
('they do too!', 'they do not!')
```

The first speaker said “they do too!” and then the second speaker responded “they do not!”. Take a look at a few of the other pairs to get a feeling for what the data looks like. Make sure that you understand exactly how this data is being stored and what it represents.

Note: when you import the `assign4_quotes` module, python will create a directory called `__pycache__` in your assignment directory. You can safely ignore this directory; in particular, don't submit it to gradescope.

A word of warning: This data comes from real movies and some of those movies have adult (i.e., R-rated) content. If you think this will be problematic for you, I'm happy to work with you to find other content since the particular data isn't critical for the assignment.

Intermission: Midterm Prep (Group)

Make a file called `midterm.txt` in your `assignment4` folder.

Phase 1 (30 minutes): In your LC, go over the study guide for the midterm (<https://cs.pomona.edu/classes/cs50/midterm-guide.pdf>). Split into pairs (or triples). Each pair should repeatedly pick one question from each section in turn and together come up with a hypothetical midterm question in any format you like (short answer, true or false, multiple choice, fix-the-bug, reorder-the-lines, fill-in-the-blank, etc; try a variety of forms). Write your question in `midterm.txt`. Be careful to design each question so that it specifically tests the ability targeted by the study guide prompt. If you finish before 30 minutes is up, work on practice questions for another prompt from each section. Try to emphasize concepts you are shakier on. Feel free to ask your TA for help or to double check that the question targets the prompt well.

Phase 2 (30 minutes): Swap questions with another pair in your LC. See if you can each answer each other's questions correctly. Discuss them if they're confusing or if their connection to the prompt is unclear; in `midterm.txt`, write down your favorite of the other group's questions. If you get through them all, swap with a different pair and repeat for the rest of the meeting.

Movie Quotes Analysis (Individual)

Before we write our chatbot, we're first going to do some analysis of the quotes.

To make grading easier, add the following to your file to delimit all of your work for this section:

```
# -----  
# Movie Quotes Analysis Section
```

Under this header, write the following functions:

1. **[2 points]** Write a function called `is_question` that takes as input a string and returns `True` if that string is a question (i.e., ends in a question mark) or `False` otherwise.

```
>>> is_question("do you want some pie?")  
True  
>>> is_question("of, course!")  
False
```

2. **[3 points]** Write a function called `get_first_quotes` that takes as input a list of tuples (i.e., our quotes data format) and returns a list of all of the first items in the tuples, i.e. the first quotes. Hint: you'll need to build your answer from scratch by iterating through all of the quote pairs and grabbing what you need as you go.

```
>>> simple_quotes = get_practice_quotes()
>>> get_first_quotes(simple_quotes)
['quote1', 'first', 'first they said this', 'what?', 'what?']
```

3. [2 points] Write a function called `get_first_questions` that takes as input a list of tuples (i.e., quotes data) and returns a list of all of the first quotes that are questions.

```
>>> simple_quotes = get_practice_quotes()
>>> get_first_questions(simple_quotes)
['what?', 'what?']
```

4. [1 points] Write a function called `count_question_quotes` that takes as input a list of tuples (i.e., quotes data) and returns the number of *first* quotes that are questions. *Hint:* this function can be written very simply using our previous functions.

```
>>> simple_quotes = get_practice_quotes()
>>> count_question_quotes(simple_quotes)
2
```

In a comment right below your function, write how many first quotes there are in the real data that are questions.

5. [3 points] Write a function called `get_average_question_length` that takes as input a list of tuples (i.e., quotes) and returns the average length (in characters) of all of the first quotes that are questions.

```
>>> simple_quotes = get_practice_quotes()
>>> get_average_question_length(simple_quotes)
5.0
```

Building a Chatbot (Individual)

To make grading easier, add the following to your file to delimit all of your work for this section:

```
# -----
# Chatbot Section
```

Chatbot Overview

Now that you're familiar with our quotes data, we're going to use it to build a very basic chatbot that will have a conversation by responding to questions. The key to the "intelligence" of our chatbot will be the quotes. In particular, the chatbot will try and find the question that the user asks as a first entry in the quotes data and then respond with the corresponding second entry.

The chatbot will have three types of responses:

- If the user enters something that isn't a question, then the chatbot will respond with `'I only respond to questions!'`.
- If the user enters a question, but that question never occurs as a first quote in our quotes data, then the chatbot will respond with `'I don't know.'`
- If the user enters a question and that question does occur one or more times as a first quote in our quotes data, then the chatbot will respond by *randomly* picking a response from the list of all second entries that had the question as the first entry.

Here is an example transcript. Note that because of the randomness, the responses will not be identical if you run the same questions.

```
>>> chatbot()
Welcome!
Ask me anything. When you're done, just type 'bye'
- who are you?
I am the Borg.
- what is your name?
My name is Sir Launcelot.
- where did you come from?
Do you believe in time travel, Donnie?
- I don't understand.
I only respond to questions!
- is that true?
Well, truth is for suckers, isn't it?.
- come on now
I only respond to questions!
- are you a robot?
I don't know.
- are you a person?
I don't know.
- what are you?
You know.
- bye
```

Implementing the Chatbot

When writing a program like this, it is important that you build up the functionality by writing smaller functions that do some of the work and then combining them to get the final program. For this assignment, we're going to help guide you through this process.

6. **[3 points]** Write a function called `get_responses` that has two parameters: our quotes data (list of tuples) and a string representing a question. The function should return (not print!) a list containing all of the second entries in the quotes data where the question exactly matches the first entry.

```
>>> simple_quotes = get_practice_quotes()
>>> get_responses(simple_quotes, "what?")
["that's what", "now you've it!"]
>>> get_responses(simple_quotes, "what is your name?")
[]
```

7. **[2 point]** Write a function called `get_random_from_list` that takes as input a list and returns a random element from that list. Don't forget that to access the random functions, you'll need to import the `random` module. You can get your random element either using `randint` and indexing or by using another appropriate function from `random`.

```
>>> get_random_from_list([1, 2, 3, 4, 5])
4
>>> get_random_from_list([1, 2, 3, 4, 5])
3
>>> get_random_from_list(["apples", "bananas", "cranberries"])
'bananas'
>>> get_random_from_list(["apples", "bananas", "cranberries"])
'cranberries'
```

8. **[3 points]** Write a function called `respond` that has two parameters: our quotes data (list of tuples) and a string representing a question. If the question matches any first entry in the quotes data, then the function should randomly pick a response from the list of all second entries that had the question as the first entry. If there were no occurrences of the question in the quotes data, then the function should return "I don't know." Note that like all the other functions so far, this should return rather than print the response.

```
>>> simple_quotes = get_practice_quotes()
>>> respond(simple_quotes, "what?")
"that's what"
>>> respond(simple_quotes, "what?")
```

```

"now you've it!"
>>> respond(simple_quotes, "what?")
"now you've it!"
>>> respond(simple_quotes, "what is your name?")
"I don't know."
>>> respond(simple_quotes, "what is your name")
"I don't know."
>>> respond(simple_quotes, "banana")
"I don't know."

```

Hint: You've already done a lot of the work for this function with the previous two functions. Use them in this function!

9. [5 points] Write a function called `chatbot` that has no parameters and implements the chatbot behavior described above in “Chatbot Overview”. The function should first print out the user instructions. The function should then continue to prompt the user for a question (see transcript above) and respond appropriately until the user enters “bye”. The response should be one of the three responses described above. Questions can be entered as any capitalization variants, even though the quotes data is all lowercased.

To get textual input from the user at the console, you can use the built-in `input` function. It returns a string, but it also takes an optional parameter for the *prompt*: a bit of text that shows just before the area where the user types. For example, the python interpreter uses `>>>` followed by a space, which we could imitate using `command = input('>>> ')`.

Advice:

- Build the behavior of this function incrementally, testing it as you go. There are many ways you might do this, but here's one approach:
 - Write a version of the function that prints out the introduction and instructions, then repeatedly prompts for input until you enter “bye”, but doesn't respond at all. You probably want to use a loop for this; which kind of loop iterates *until* something becomes true?
 - Add in checking for whether or not it's a question. You can just print out some generic response if it is a question for now. Use the functions you defined earlier!
 - Add in code to actually get an appropriate response and print it out.
 - Add in any additional functionality that you're still missing.
- All of the first quotes are lowercased, so make sure that you lowercase the user question before searching, otherwise, you won't get a match (see if there's some useful method in `help(str)`).
- When you're first testing it, use the practice quotes rather than the real quotes since it will allow you to test all of the three different response cases very easily. Once you're sure it's working, you can switch the code to read the real data.

It is especially important to *remove all `print` or `input` calls* that are not part of the `chatbot` loop. They *will* break the autograder!

When you're done (Individual)

Make sure that your program is properly commented:

- You should have comments at the very beginning of the file stating your name, course, assignment number and the date.
- You should have comments delimiting the two sections.
- Each function should have an appropriate docstring.
- Include other miscellaneous comments to make things clear.

In addition, make sure that you've used good *style*. This includes:

- Following naming conventions, e.g. all variables and functions should be lowercase.
- Avoiding repetition—don't have direct copies of code anywhere in your file; instead you should be using your helper functions wherever they can come in handy.
- Using good variable names.
- Good use of booleans. You should NOT have anything like:

```
if boolean_expression == True:
```

or

```
if boolean_expression == False:
```

instead use:

```
if boolean_expression:
```

or

```
if not (boolean_expression): # or some other way of negating the expression
```

- Proper use of whitespace, including indenting and use of blank lines to separate chunks of code that belong together.
- Make sure that none of the lines are overly long.

Submitting

Submit only your `assign4.py` and `midterm.txt` files on Gradescope.

Grading

	points
Quotes analysis	11
Chatbot	13
Comments, style	3
total	27