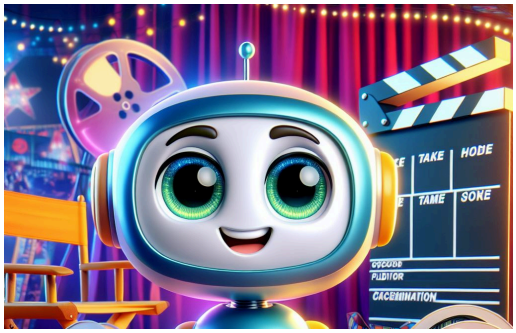
ICS 31 - Project 1

Movie Chatbot

Due: Sunday May 18, 2025 11:59pm

Project Assessment: part of quiz 5



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

The key to our chatbot will be a dataset of over 300K spoken lines from 617 movies[1]. 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.

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

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

- Open up VSCode and open the folder that you placed your project 1 files in.
- Make a new Python file called **project1.py**. Add your name and the project number in comments to the top of this file.
- Download the **project1.zip** file from the project 1 directory and unzip it
- Copy the files from that unzipped starter into the folder containing your project1.py file. To find that folder, you can right click your file in the explorer window and use the Reveal in Finder menu item , or you can just navigate there in your file browser.

– Delete the zip file. Your project folder should now have three files in it: **project1.py**, **project1_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 **project1.py** file below the header comments:

from project1_quotes import get_quotes, get_practice_quotes

Run your program by clicking the play button or in the terminal and then test and make sure everything is loaded correctly. There are two functions in the **project1_quotes** module in the project1_quotes.py file: **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). In each tuple/pair, the first element represents the first quote and the second element the response quotes. 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 got
it!")]
```

We can look at the first pair of real quotes:

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

The first speaker said "they do to!" 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 **project1_quotes** module, python will create a directory called __pycache__ in your assignment directory. Just ignore this directory.

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 provide a censored version of the data.

Movie Quotes Analysis

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
```

1. Write a function called **`is_question`** that takes as input a string and return `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. 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. 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. 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 your 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. 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

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 may not be identical if you run the same questions.

```
>>>
chatbot(0)

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 decompose the problem into smaller subproblems. The program will be constructed by building 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. 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 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 got it!"]
>>> get_responses(simple_quotes, "what is your name?") []
```

7. 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.

```
>>> get_random_from_list([1, 2, 3, 4, 5])
2
>>> 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. 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."

```
>>> 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. **Chatbot version 0:** Write a function called **chatbot** that takes one parameter **version.** The function should implement 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 are all lowercase.

   In version 0, the chatbot looks for exact matches between the user question and the movie questions. To execute your chatbot program for testing, insert the following code at the

bottom of the project1.py file.

Note: Change the argument to the version number when testing other versions.

```
if __name__ == '__main__':
    chatbot(0)
```

10. **Chatbot version 1:** Increase the number of questions your chat bot will respond to. Instead of matching the user question exactly, consider a question to be similar if its Jaccard similarity with a movie question is at least 0.6 (60%). To compute the Jaccard similarity, remove punctuations and split each question into words. Given 2 sets of words, calculate the jaccard similarity as (size of intersection) / (size of union).

    For example, the similarity would be 0.75 with a user question of "Who are you?" and a movie question of "Who are you exactly?". The intersection has size 3, and the union has size 4.

    Randomly select one of the responses from all similar movie questions.

    Note: In Python, **string.punctuation** is a predefined string constant that provides a collection of common punctuation characters. You will need to import the **string** module. You can use **string.punctuation** to remove the punctuation characters. Then, split the sentence into words to check for word similarity.

11. **Chatbot version 2:** Improve the selection process of the answer when multiple quotes match the question. Version 2 requires the Jaccard similarity functionality described for version 1. Instead of random selection, select a quote with the highest % of matching words. Only use random selection when there's a tie on the similarity %.

    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 intro information and repeatedly
        - 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.
        - 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.
    - 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.

When you're done

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.
- 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 too long

When you're done

Submit your project1.py file (do NOT submit any starter code that you downloaded) to the Pensieve autograder. There will be a project assessment question in the quiz after the project's due date to show your understanding.