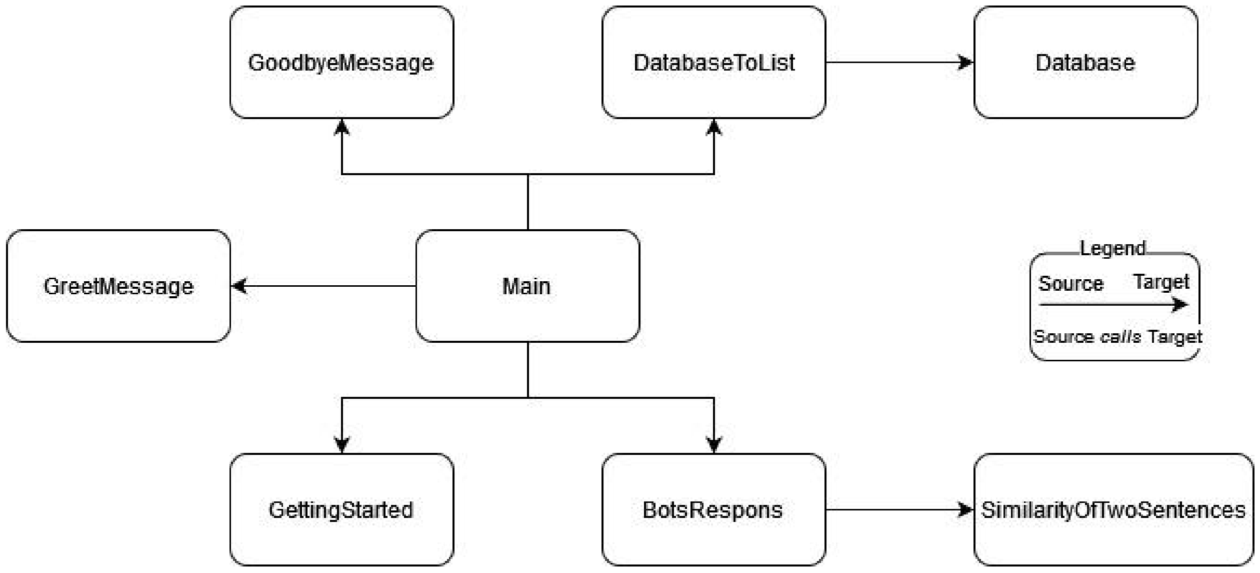
Specifications Increment 2



Stakeholder requirements and expectations:

The main requirement and expectation of Increment 2 is revamping the database. The overhauling of the database is essential to the program's functionality. For example, if the database does not contain a particular pair, prompt and answer, on a given topic, the software system will not respond to said topic. Alongside revising the database, the class "main" will be changed to achieve a more clean and lean code. Finally, the contents of the lists in "GreetMessage", "GoodbyeMessage", and "GettingStarted" will be reconsidered.

Database:

The database is comprised of numerous pairs of sentences on the topic of loneliness. A pair is made up of a "prompt" and a corresponding "answer". The "answers" are designed in such a way as to provoke a specific input from a user. In addition, pairs should be designed holistically. Finally, the pairs should lead the user to enter an input that will result in the program saying something to the effect: "I would suggest for you to consider X. What else are you thinking about?"

Test:

Create a new class. Import the class Run a for-loop that goes through each line, and storing, temporarily, said line into a variable, current_line. After storing the line in said current_line, use the method .split(" @ ") on current_line. Print said application, i.e., print(current_line.split(" @ ")). After running the loop, scan over the printed lines, and make sure (i) that each "prompt" and answer are in a size 2 list, (ii) the "prompt" comes before the "answer", (iii) there is no "@", (iv) the "prompt", in the list, does not have a space at the end, (v) the "answer", in the list, does not have a space at the beginning and end (vi) the "prompt" has at least 3 words. If all the sentences have passed the six metrics, then the Database has passed the test.

Because of the changes to the Database, DatabaseToList needs to be tested as well.

Create a new class. Import the class DatabaseToList, and call the function, database_to_list(). Make sure you have implemented class Database first. Because it will return a list, print said list. Scan over the printed list, and make sure (i) that each "prompt" and "answer" pair are in a size 2 list, (ii) the "prompt" comes before the "answer", (iii) there is no "@", (iv) the "prompt", in the list, does not have a space at the end, (v) the "answer", in the list, does not have a space at the beginning, end (vi) the "prompt" has at least 3 words, (vii) the printed list contains list(s) of size 2. If printed list has passed the seven metrics, then the DatabaseToList has passed the

Main:

To refine the logic and structure of the code. For example, there is an if-statement, "if correctnessValue > 1:" that is never executed. Finally, add a feature that can help distinguish the user and bot's sentences; and a more robust error handling.

Test: First enter an input that does contain a char that is not a letter, expected output should be: an appropriate output informing the user that the input is invalid. Then, input something appropriate. Next, enter an input that does contain a char that is not a letter, expected output should be: print an appropriate output informing the user that the input is invalid. Next, input a sentence that is larger than any "prompt" in the database; the expected output should be: "Bot: I'm sorry, I cannot understand that sentence. Could you say it a little more simply please?". Next, input a sentence that is at least 3 words long, but is smaller or equal to the largest "prompt" in the database. The expected output should be an answer that is appropriate. To determine an appropriate answer, apply the method bot_respons (str, list). After doing so, if the output is appropriate and matches the manually calculated answer, then the case is cleared. Next, enter in a sentence that is less, in terms of length, than or equal to 2; expected output is an output from the method goodbyeMessage() and closers of the program. If the code passed all the above tests, then the code has passed testing.

GreetMessage:

Reconsider the contents of the list, and make sure they are appropriate greet messages.

Test:

Create a new class. Import the class GreetMessage, and call the function, greetMessage(), 10 times. The function is working correctly if (1) the greeting messages showup correctly, and (2) randomly. Rinse and repeat 3 times. If all three runs work, then the function has passed the test.

GoodbyeMessage:

Reconsider the contents of the list, and make sure they are appropriate goodbye messages.

Test:

Create a new class. Import the class GoodbyeMessage, and call the function, goodbyeMessage(), 10 times. The function is working correctly if (1) the goodbye messages showup correctly, and (2) randomly. Rinse and repeat 3 times. If all three runs work, then the function has passed the test.

GettingStarted:

Reconsider the contents of the list, and make sure they are appropriate messages. An appropriate message is one that prompts the user to input a sentence on the topic of loneliness.

Test:

Create a new class. Import the class GettingStarted, and call the function, gettingStarted(), 10 times. The function is working correctly if (1) the getting started messages showup correctly, and (2) randomly. Rinse and repeat 3 times. If all three runs work, then the function has passed the test.