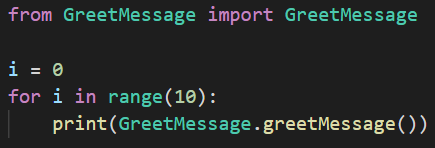
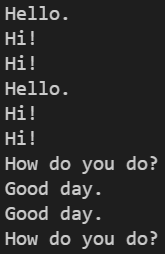
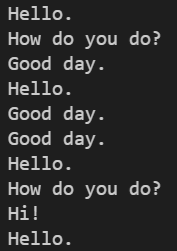
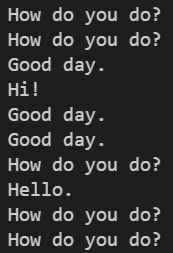
Names of methods or variables may not match.

Testing was done in the following order:

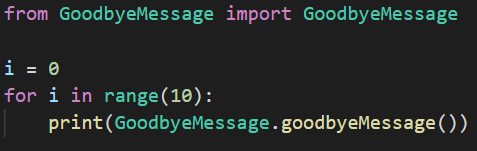
* GreetMessage:
  + Test: Create a new class. Import the class GreetMessage, and call the function, greet\_message(), 10 times. The function is working correctly if (1) the greeting messages show up correctly, and (2) randomly. Rinse and repeat 3 times. If all three runs work, then the function has passed the test..
    - Code used to test:



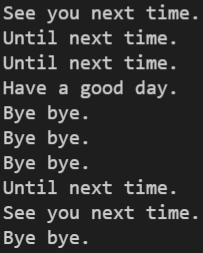
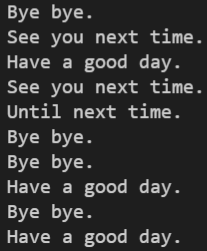
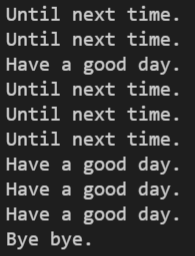
* + - Outcome of the test:



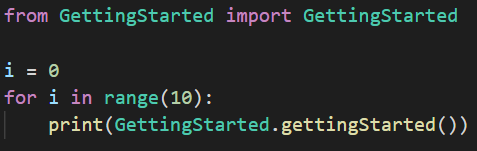
* + - Test success: Passed
* GoodbyeMessage:
  + Test: Create a new class. Import the class GoodbyeMessage, and call the function, goodbye\_message(), 10 times. The function is working correctly if (1) the goodbye messages show up correctly, and (2) randomly. Rinse and repeat 3 times. If all three runs work, then the function has passed the test.
    - Code used to test:



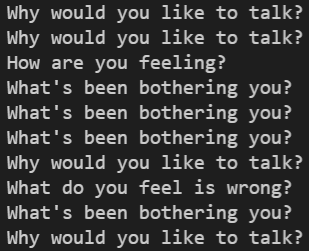
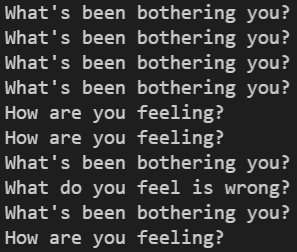
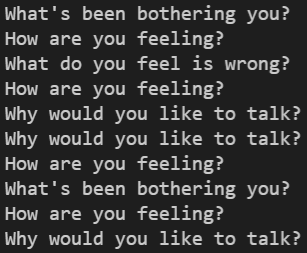
* + - Outcome of the test:



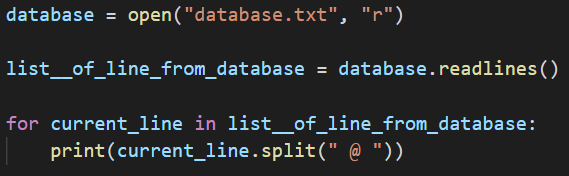
* + - Test success: Passed
* GettingStarted:
  + Test: Create a new class. Import the class GettingStarted, and call the function, getting\_started(), 10 times. The function is working correctly if (1) the getting started messages show up correctly, and (2) randomly. Rinse and repeat 3 times. If all three runs work, then the function has passed the test.
    - Code used to test:



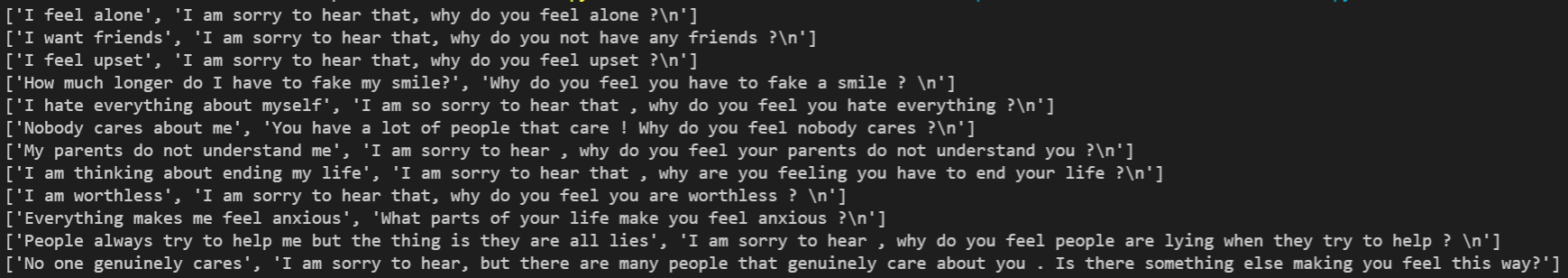
* + - Outcome of the test:



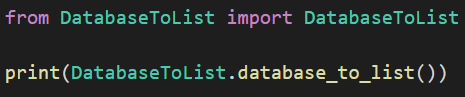
* + - Test success: Passed
* Database:
  + Test: Create a new class. Import the class and in a loop that goes through each line, and storing, temporarily, said line into a variable, current\_line. After storing the line in said current\_line, use the method .split(" @ ") on current\_line. Print said 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.
    - Code used to test:



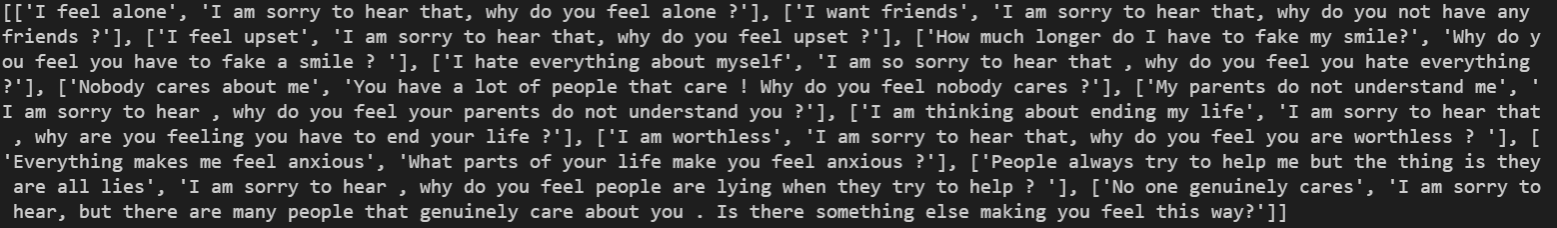
* + - Outcome of the test:



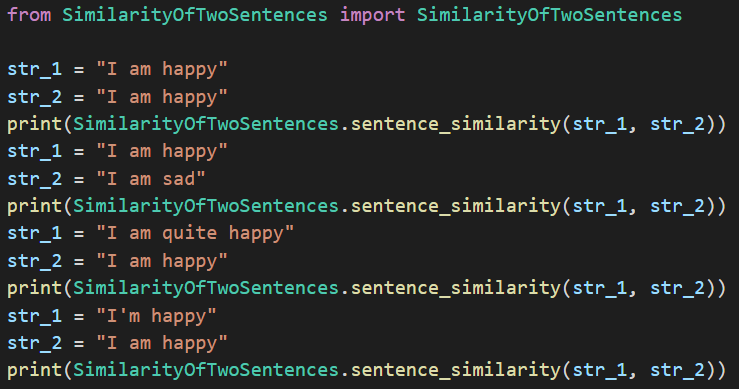
* + - Test success: Passed
* DatabaseToList:
  + Test: Create a new class. Import the DatabaseToList, and call the function, database\_to\_list(). Make sure you have implemented class Database first. Because it will return a list, print said list. Scan over the printed list, and make sure (i) that each "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, (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 test.
    - Code used to test:



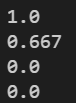
* + - Outcome of the test:



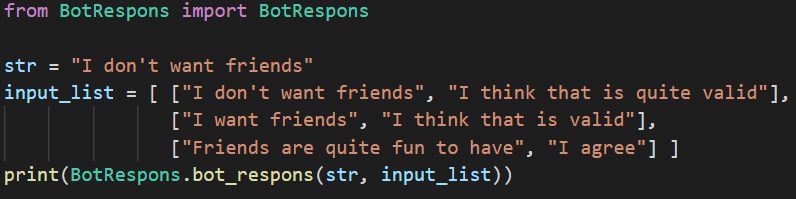
* + - Test success: Passed
* SimilarityOfTwoSentences:
  + Test: Create a new class. Import the class SimilarityOfTwoSentences, and call the function, sentence\_similarity(str\_1, str\_2); where the four cases are tested: (i) str\_1 = "I am happy" and str\_2 = "I am happy". Output expected: 1 (ii) str\_1 = "I am happy" and str\_2 = "I am sad" Output expected: 0.666… (iii) str\_1 = "I am quite happy" and str\_2 = "I am happy". Output expected: 0 (iv) str\_1 = "I'm happy" and str\_2 = "I am happy". Output expected: 0. If outputs come out correctly, then the class SimilarityOfTwoSentences has passed the test
    - Code used to test:



* + - Outcome of the test:



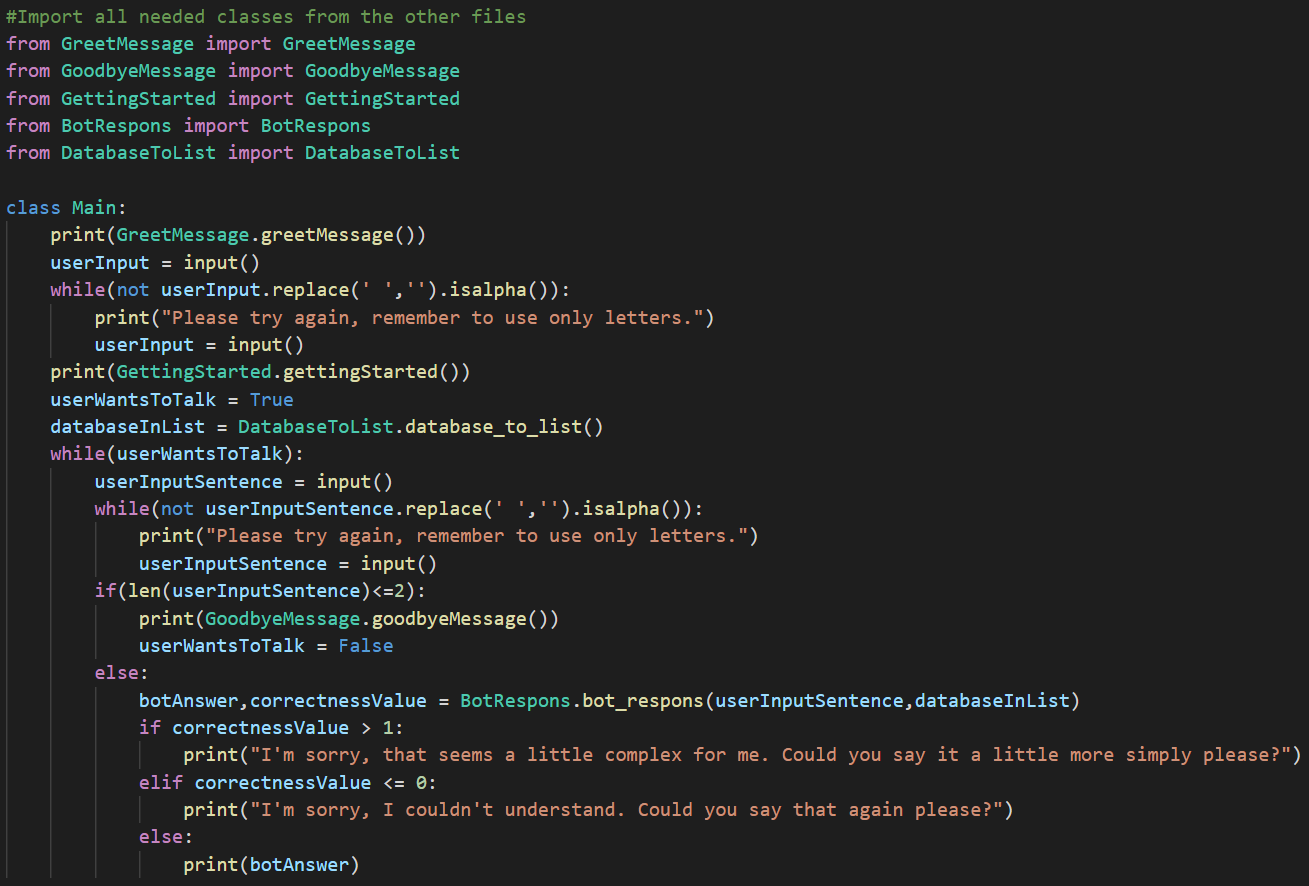
* + - Test success: Passed
* BotsRespons:
  + Test: Create a new class. Import the class BotsRespons, and call the function, bot\_respons (str, input\_list); where a case is tested: str = "I don't want friends" input\_list = [ ["I don't want friends", "I think that is quite valid"], ["I want friends", "I think that is valid"], ["Friends are quite fun to have", "I agree"] ] Output expected: "I think that is quite valid", and 1. If the output comes out as expected, then BotsRespons has passed the test
    - Code used to test:



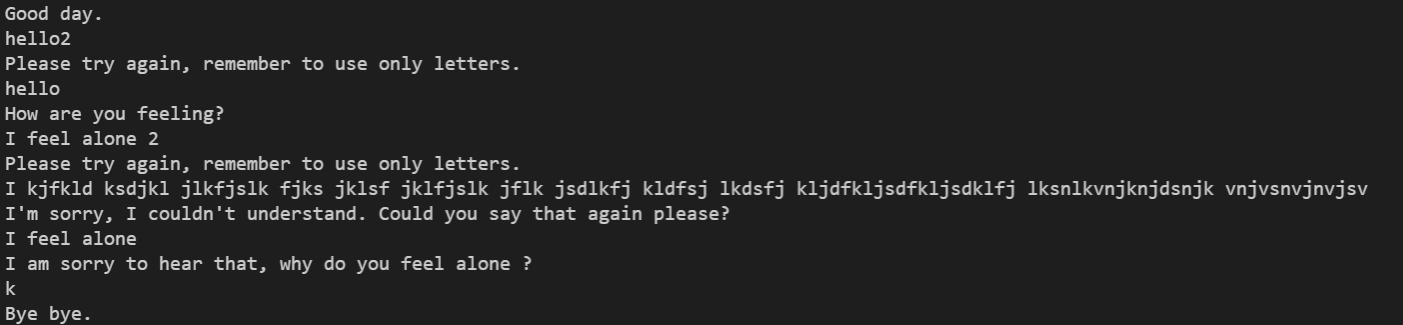
* + - Outcome of the test:



* + - Test success: Passed
* Main:
  + 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: a message informing the user, that the bot could not understand. 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 goodbye\_message() and closers of the program. If the code passed all the above tests, then the code has passed testing.
    - Code used to test:



* + - Outcome of the test:



* + - Test success: Passed