aStart and Last Modification Dates

Start Date: 27.04.2018

Last Modification Date: 03.05.2018

Contents

1 Introduction

In this assignment it is asked to build a code structure in Python to translate each letter, word, sentence from Morse Code format to written format. There are 5 different steps to create this program and every step is defined by different tasks. User is guided for usage of every task one by one.

2 Tasks

2.1 Task1

In this task, it is asked to define a class that decodes the Morse code. This class has one instance variable which stores the dictionary structure for 26 letters, 10 digits and 3 punctuations and their corresponding Morse codes.

In this task there is a magic method to modify the str() function. After this modification, str() function is capable of getting Decoder class instances as arguments to return it as a formatted string.

And decode method which accept Morse code sequence as an argument then splits it word by word using "***". After this process the method splits words using "*" letter by letter and checks whether all letters are defined in the dictionary. If not, returns an error message as "Invalid input, it is not defined in Morse code dictionary." If so, decodes it using dictionary and stores it in a variable called decoded.

After decoding process finishes, module checks whether the last entry of Morse code is a punctuation or not. If not, it returns an error message as "Invalid input, it cannot finish without a punctuation." If so, returns the decoded sequence which is a string type sequence.

Decoder module accepts consequent punctuations and decodes them.

Decoder module doesn't accept sequences starting with "*" and returns an error message as "Invalid input, it is not defined in Morse code dictionary."

Decoder module doesn't accept sequences including two or more than three consequent "*" and returns an error message as "Invalid input, it is not defined in Morse code dictionary."

Decoder module doesn't accept sequences ending with "*" or "***" and returns an error as "Invalid input, it is not defined in Morse code dictionary."

Decoder module accepts sequences starting with any of the punctuations, decodes it and then returns to decoded sequence.

2.2 Task 2

In this task CharacterAnalyser class is created with an instance variable which is a dictionary structure storing occurrences of 26 letters and 10digits (even if the character isn't decoded, dictionary stores it as zero occurrence)

In this task there is also one more magic method to modify the str() function. After this modification, str() function is capable of getting CharacterAnalyser class instances as arguments to return it as a formatted string.

There is a analyse_characters method that performs analysis on decoded sequences. It counts the occurrences of every character and saves them into dictionary structure of CharacterAnalyser class's instance variable. This method takes every element of decoded sequence and checks whether it is in dictionary structure or not. If so adds 1 to value of character in dictionary, if not just passes it.

This module doesn't make input validation, it takes all inputs as valid and counts them. Validation are done while both inputting and decoding.

2.3 Task 3

In this task WordAnalyser class is created with an instance variable which is a dictionary structure storing occurrences of every word (every string sequence drawn apart with a single space or a punctuation)

The dictionary structure's default state is an empty dictionary.

In this task there is also one more magic method to modify the str() function. After this modification, str() function is capable of getting WordAnalyser class instances as arguments to return it as a formatted string that presents number of occurrences of each word in a readable format.

Task 3 includes a method called analyse_words which takes decoded sequence as argument and counts the number of occurrences of each word and stores them in the dictionary structure of WordAnalyser class. This module firstly replaces punctuations in the decoded sequence with a space to distinguish the words using spaces between them then splits decoded sequence using the spaces in the sequence and stores split words in a list. Checks whether there is an empty sequence in the created list or not. If so removes it. (These empty sequences can exist due to change of punctuations into spaces which may create consequent spaces in the sequence that will end up with empty strings after splitting of string.). Then module changes list into set using set() command in order to find out the words that will be saved in the dictionary structure. After that point, module gets every element of list as key in dictionary structure and adds one to their corresponding value.

This module doesn't make input validation, it takes all inputs as valid and counts them. Validation are done while both inputting and decoding.

2.4 Task 4

In Task 4 SenteceAnalyser class which have a dictionary structure as instance variable, is defined. This instance variable has 3 entries as ‘Complete sentences’, ‘Clauses’ and “Questions” starting from value of zero as default.

There is a method to re-defines the str() function. This method takes every key from dictionary structure and their corresponding value then turns them into a formatted string. It is capable of getting SentenceAnalyser class’s instances as argument.

There is an analyse_sentence method that takes decoded string as an argument and counts the number of occurrences of punctuations to decide how many sentences, clauses and questions are there in the decoded sequence. Method first deletes the spaces in order to recognize consequent punctuations that have just a space between them then puts all the elements of decoded string into a list in order to make them mutable. After this point method uses a while loop to delete consequent punctuations in the list and leaves just one of them which is the first one in the queue. After these processes, method counts the occurrences of punctuations and store them in the dictionary structure.

Due to the input rules user can enter consequent punctuations but method counts consequent punctuations as 1 punctuation and that counted is the first occurred one in the string.

This module doesn’t make input validation, it takes all inputs as valid and counts them. Validation are done while both inputting and decoding.

2.5 Task 5

For this task firstly, classes created in the previous tasks are imported then a function is defined to get the input from user. This input function also makes validation for the input as well as decode method. get_input function checks whether the entered sequence finish with a punctuation or not. If not, it prints and error message telling ‘Invalid input, every input must end with a punctuation’ and checks that is there at least one sequence of three asterisks (which is a input rule for get_input function) or not. If not, it prints an error message as ‘Invalid input, there should be at least one set of three consecutive ‘*’’

This function then divides the input according to places of ‘***’ in order to specify the coded words in the sequence then divides specified words according to places of ‘*’ and ends up with codded letters. After that point function makes one more check to decide whether the input is valid or not and searches for the dictionary of Decoder for all decoded letters entered by the user. If all of them matches function returns encoded sequence which is entered by the user, if not, prints an error as ‘Invalid input it is not defined in the dictionary’

get_input function doesn’t accept sequences that doesn’t include at least one set of three consecutive ‘*’.

get_input doesn’t accept sequences starting with “*” and returns an error message as “Invalid input, it is not defined in Morse code dictionary.”

get_input doesn't accept sequences including two or more than three consequent "*" and returns an error message as "Invalid input, it is not defined in Morse code dictionary."

get_input function accepts consequent punctuations.

There is a function called choice function in this task. It is defined to create a menu for the user in order to make user decide which level of analysis he/she wants. User can choose character analysis, word analysis, sentence analysis or all of them to see the results of analysis for the last entered sequence. If user doesn't choose any of them and enters any other character(s), he/she doesn't see the result of any analysis but entered sequence is counted for the total numbers that will be showed at the termination of the programs. Therefore, if user enters a valid input, regardless of analysis choice, the input will be counted for the total numbers.

Task 5 includes a main function to drive the flow of execution of the whole program. Firstly 4 instances of 4 classes are created in the main function. Then a string is created to store all entered valid inputs' decoded state.

The a while loop is set to make user give input until he or she terminates. In the loop get_input() function is invoked. Next to inputting, there is an if statement to check whether get_input() function returns False or not, if it returns False (means invalid input is entered by the user) ignores the if statement and starts the while loop again forcing user to enter one more sequence until user enters a valid sequence it is not possible to stop program without error.

If user enters a valid input, Decoder class' decode method is invoked and decodes the Morse code and adds the decoded sequence to the other sequence which will store all of the decoded strings. At this point choice() function is called to make user choose one of the options in the menu. After it, user is prompted with a sentence as 'if you want to terminate please press t and enter, if not just enter' that allows user to terminate the program.(if user enters any other character(s) than 't' program won't terminate) If user doesn't terminate, will enter one more sequence of Morse code until termination and will be able to see the analysis of every other sequence explicitly. At the termination total occurrences of all valid inputs' characters, words and sentences are printed on the screen.

At the end of the task 5 there is an if statement to execute the program only if the file runs directly and not imported.