Morse Code Translator

**Name:** Aw Shao Yang

**Class:** DAAA/FT/2B/03

**Student ID:** P2012126

File Content

The folder you have downloaded should consist of the following directories and files:

Code:

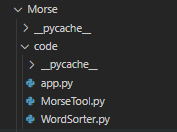
* **Morse.py**: Contains python Morse classes with various functions, such as morse decoding, encoding, file read and write requesters, attribute getters, attribute setters, as well as other methods meant for programmers to create their own programs that carry out morse translations.
* **App.py:** Contains Code for the main Morse application that makes use of Morse classes to carry out certain tasks
* **WordSorter.py:** Contains custom class datatypes and Word Sorter classes which allow the Morse class in Morse.py to do better morse sorting for its morse analysis function

Text Files

* **Morse1.txt:** Contains the Morse message in a text file meant for morse decoding as well as its morse analysis function
* **Stop-words.txt**: Contains a list of Stop words meant to be filtered out to get the essential decoded morse message.

**User Application Guidelines**

Application Set Up



In-order to set up the application in Visual Studio Code, unzip the folder that you have downloaded, the files should already be sorted in this manner as seen on the left, but do ensure that they are. This is essential especially for the python files if you want the entire program to function as created as it may cause program crashes as well as errors in starting the main program due to file class importing errors.

Application Usage

1. Change Printing Mode

To change the printing mode for your encoded morse code select 1, it will tell you your current printing mode and request for which printing mode you want. To choose the printing mode you want, just type v for vertical printing or h for horizontal printing. Then press enter. You should see this:

Text

Description automatically generatedText

Description automatically generated Horizontal to Vertical Vertical to Horizontal

Text

Description automatically generated

If you choose the print mode, that you are currently in you will get this message instead:

1. Encoding a Message

To encode a sentence, select the option 2, you will be requested for the sentence that you want to translate to morse code. Any symbol that is unrecognized by the morse translation sheet in the Morse encoder will not be translated, instead the original symbol will unchanged and placed in the encoded message. The output message depends on the printing mode that is set

Text

Description automatically generatedHorizontal Printing Vertical Printing

Text

Description automatically generated

1. Analyzing a Message from a file and writing a report

The next function requests for a text file that contains morse code messages, as well as an output file path to give a report on the analysis of the morse message given. It gives out the fully decoded message, sorted frequency, length, and text coordinates of each word in the entire message and lastly the essential message of the entire text. To analyze it, just choose option 3 and input the file that you want to read and write to. In the example below I have a file named morse1.txt that contains the morse messages to decode and I saved the report to report.txt.

Text, letter

Description automatically generatedApp.py morse1.txt

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generatedOutput report.txt

Here you see that it generated a report on the analysis on screen as well as in a text file

Text

Description automatically generated

Text, letter

Description automatically generated

1. Exiting

To Exit the app just key in 4.

**Data Structures and Algorithms Used**

**Custom Data structures:**

SortedList

Purpose: Return a sorted link list when I insert new custom data types. Used to sort different types of objects based on custom object attributes upon inserting efficiently and effectively. Methods in this class include:

* \_\_appendToHead (Appends new node to be the earliest node)
* Insert (sorts and inserts new nodes properly)
* node\_to\_list. (Completely converts the sorted list into a python list for custom Word class)

Challenges Faced: Challenge deciding how to incorporate it so that it sorts elements matching the requirements in the brief. This is due to the elements requiring multiple ways to be sorted, through (frequency, length, and alphabets) for the full message as well as (frequency and coordinates) for the essential message.

Solution: Develop custom data types as classes with these attributes, e.g., Word (word, frequency, length, placement) and conducting operator overloading, changing less (<) than and greater (>) than operators. Allows comparison of custom datatypes on their attributes and sort them using the standard Sorted Lists.

Limitations: Harder extraction of data as we can’t do it using index. To solve this, I created a method that converts the SortedlinkList into a python list with sub lists based on word frequencies

Big O Notation for inserting: O (n)

**Python Data Structures Used:**

Lists

The main data structure that I have used made by python would be the list data structure. This is due to its in-depth development, allowing quite flexible inserts, removal, sorting and list creation. Another reason would be its ability for multidimensional lists which was very good for coordinate retrieval for each word etc.

Dictionaries

In the program, another data structure used would be pythons dictionary, which was created to store the morse code reference sheet in the Morse object for encoding and decoding

**Main Classes Developed**

**Morse**

The Morse object is meant to be treated as a toolbox that allows you to use these methods to create new applications for Morse codes or new classes through inheriting. The attributes include

* \_\_codes: Contains a dictionary of English to morse code translation sheet
* \_\_valid\_morse: Contains list of values allowed to be stored in \_\_morse\_message, only values in this list are allowed in \_\_morse\_message for decoding
* letter\_message: Meant to store the sentence that is used for encoding.
* \_\_morse\_message:Stores the morse message meant to be decoded into English, input can only contain text in the \_\_valid\_morse list
* \_\_stop\_words: Meant to store list of stop words to filter out. Input can be from using a text file path or a custom list of words from filtering.

Reason for mostly private attributes is to enable data checking, allowing only certain values.

Next are the main methods of the Morse class

encodeMorse(self)

This is meant for encoding a normal English message by referencing the morse dictionary stored in the \_\_codes attribute. Instances without any letter\_ message will receive an error and prompt

* Big O notation: O ()

decodeMorse(self)

This is meant for decoding a morse message also through referencing the morse dictionary. Instances without any \_\_morse\_message will receive an error and prompt

* Big O notation: O ()

analyze(self)

This method is the most sophisticated one as it carries out the analysis of a \_\_morse\_message. It returns the decoded message, sorted list of words in the morse message, and the filtered essential message. To create this function, it was broken down to subtasks.

Text

Description automatically generatedThe first task to handle, is to decode the morse message, which was done by decodeMorse(self) of big O notation . Next was to sort the list of words decoded from the morse message. How we need to sort it was based on the highest frequency, lowest word length and lastly the earlier alphabets. What I did was create a custom word sorter using SortedList data structure to return a list of sorted elements using nodes.

To sort my data by custom attributes, I created my own custom class data type called Word. 🡪

A screenshot of a computer

Description automatically generated with medium confidence

I then conducted operator overloading to define what was greater than, in order to sort my custom objects 🡪

Text

Description automatically generatedTo create and sort my objects I needed to collect index coordinate positions of each unique word in the sentence, hence I created 2 static methods

1. To locate all the coordinates for each word in a 2-Dimensional list. 🡪Big O notation: O ()

2. Get a list of words that are unique in a message. 🡪Big O notation: O ()

Text

Description automatically generatedThen a Sorted word list was created in the analysis method with the list of words, the words were inserted after converting the string datatypes into the Word class by initializing the word as an instance of the Word class, with their respective list of coordinate positions. 🡪

Text

Description automatically generatedAfter obtaining a SortedList of Words, I need to output the frequencies of the words and the list of words with their attributes. So, I created a method that converts a sortedlist into a python list with sub lists grouped by their frequencies called node\_to\_list(). 🡪

To output the sorted words, I looped through the list and sub lists.

Text

Description automatically generatedThe last task was to obtain the essential message. How it is done is through sorting a list of filtered words based on its frequency and earliest coordinate in a message.

Text

Description automatically generatedTo do this I created Coordinate\_Word class 🡪 which inherits from Word since the attributes are the same. Then I overwritten its greater than operator to change how it compares. I also created a Coordinate object and overloaded its less than operator to help compare the coordinates of the word placement. For easy conversion from a Word to Coordinate\_Word with original attributes I created a word\_to\_coordword() method in the Word object

A screenshot of a computer

Description automatically generated with medium confidenceBefore sorting and creation of the essential message, the words must first be filtered out using this filter method created called \_\_filter\_stop\_words Big O notation: O (n). These filtered words are set either using a text file with each word on a new line or a custom list of words stored under the \_\_stop\_words attribute. 🡪

Text

Description automatically generated🡨 Lastly, I initialized another word sorter object to sort the filtered Coordinate\_Words to generate the essential message. Now through adding up all the output, we can now print it as well as return it to upload it into a file

🡪 Analyze () method Big O notation: O ()

MorseHorizontal (Morse)

A screenshot of a computer

Description automatically generated with medium confidenceDescription: MorseHorizontal is a child class of Morse that enables morse encoded message printing. Through polymorphism I overloaded the \_\_str\_\_ operator to for printing of the encoded morse message when given a \_\_letter\_message. The message will be encoded and printed out when asked to print(object)

Text

Description automatically generatedHori\_to\_verti(self): Convert the object from MorseHorizontal to MorseVertical to change its printing style in our main program

MorseVertical (Morse)

Description: MorseVertical is also a subclass of Morse that enables morse encoded message printing in the vertical manner. This was also created through polymorphism by overloading the printing operator. The printing in Morse Vertical was much harder to deal with as we needed to print it vertically. Printing vertically has a longer operation than MorseHorizontal as it includes many for loops just for printing. 🡪Big O notation: O ()

Verti\_to\_hori(self): Convert the object from MorseVertical to MorseHorizontal to change its printing style in our main code

Main Program

In the main program, the MorseHorizontal and MorseVertical classes were imported and were the classes used in the app itself. The option to switch between horizontal and vertical printing was mainly handled by the app and not the object. The app mainly makes use of the object tools and attributes by swapping, manipulating, and using the objects methods to carry out the tasks.

Class Summary (Parent Class 🡪 Child Class)

Morse 🡪 MorseHorizontal and MorseVertical

Node 🡪 Word 🡪Coordinate\_Word

No Child classes: Coordinate, WordSorter

Appendix A Code

App.py (Main Program)

####################################################################

# ST1507 DSAA: MorseCode Message Analyzer (Morse Class)            #

#------------------------------------------------------------------#

#                                                                  #

# - Name: Aw Shao Yang                                             #

# - Class: DAAA/FT/2B/03                                           #

# - Admission Number: p2012126                                     #

####################################################################

from MorseTool import MorseHorizontal

from MorseTool import MorseVertical

#General Input function for user input 1 to 4

def get\_input():

    choice = 0

    choices = [1,2,3,4]

    print("""\n\nPlease select your choice ('1','2','3','4'):

    1. Change Printing Mode

    2. Convert Morse Code To Text

    3. Analyze Morse Code Message

    4. Exit """)

    #Only possible choices are 1 to 4, else ask agian

    while choice not in choices:

        try:

            choice = int(input("Enter choice: "))

        except ValueError:

            print("Please enter a number 1 to 4")

    return choice

def start():

    choice = 0

    #application creates the MorseHorizontal Object

    morse\_obj = MorseHorizontal('../stop-words.txt')

    printing\_mode = 'horizontal'

    while choice != 4:

        print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\* ST1507 DSAA: MorseCode Message Analyzer                \*\n\*--------------------------------------------------------\*\n\*                                                        \*\n\*   - Done by: Aw Shao Yang (2012126)                    \*\n\*   - Class: DAAA/2B/03                                  \*\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        choice = get\_input()

        #Converting from h to v or v to h

        if choice == 1:

            printables = ['h','v']

            print\_state = ''

            print(f"Current print mode is {printing\_mode}")

            #Ensure proper input

            while print\_state not in printables:

                print\_state = input('Enter h for horizontal, Enter v for vertical then press enter: ').lower()

            #If requesting for convert to horizontal

            if print\_state == 'h':

                #Check current object type

                if isinstance(morse\_obj, MorseHorizontal):

                    print('Printing is already horizontal')

                else:

                    #Use convert MorseVertical to MorseHorizontal

                    morse\_obj = morse\_obj.verti\_to\_hori()

                    print('The print mode has been changed to horizontal')

                    printing\_mode = 'horizontal'

            #If requestion for convert to vertical

            if print\_state == 'v':

                #Check current object type

                if isinstance(morse\_obj, MorseVertical):

                    print('Printing is already vertical')

                else:

                    #Use convert MorseHorizontal to MorseVertical

                    morse\_obj = morse\_obj.hori\_to\_verti()

                    print('The print mode has been changed to vertical')

                    printing\_mode = 'vertical'

            input('\nPress Enter, to continue....')

        #Output message

        elif choice == 2:

            #Use object morse encoder method

            message = str(input("Please type text you want to convert to morse code:\n"))

            morse\_obj.letter\_message = message

            #Print output is different for Vert and Horizontal Morse

            print(morse\_obj)

            input('\nPress Enter, to continue....')

        elif choice == 3:

            #Request files, analyze and then write to file

            try:

                morse\_obj.request\_morse\_from\_file()

                morse\_obj.request\_write\_file()

                analyzed\_content = morse\_obj.analyze()

                morse\_obj.write\_file(analyzed\_content)

            except ValueError:

                print("Invalid Values in file. Please have a Text file with only '.' ',' '\\n' and ' '")

        else:

            continue

    print('Bye thanks for using ST1507 DSAA: MorseCode Message Analyzer')

#Testing Ground

if \_\_name\_\_ == "\_\_main\_\_":

    start()

MorseTool.py (Morse Encoder, Decoder, and analyzer)

####################################################################

# ST1507 DSAA: MorseCode Message Analyzer (Morse Class)            #

#------------------------------------------------------------------#

#                                                                  #

# - Name: Aw Shao Yang                                             #

# - Class: DAAA/FT/2B/03                                           #

# - Admission Number: p2012126                                     #

####################################################################

import os

import os.path

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

from WordSorter import Word

from WordSorter import WordSorter

#General morse class

class Morse:

    #Morse Codes transaltion sheet

    \_\_codes = { 'A':'.-', 'B':'-...',

                    'C':'-.-.', 'D':'-..', 'E':'.',

                    'F':'..-.', 'G':'--.', 'H':'....',

                    'I':'..', 'J':'.---', 'K':'-.-',

                    'L':'.-..', 'M':'--', 'N':'-.',

                    'O':'---', 'P':'.--.', 'Q':'--.-',

                    'R':'.-.', 'S':'...', 'T':'-',

                    'U':'..-', 'V':'...-', 'W':'.--',

                    'X':'-..-', 'Y':'-.--', 'Z':'--..',

                    '1':'.----', '2':'..---', '3':'...--',

                    '4':'....-', '5':'.....', '6':'-....',

                    '7':'--...', '8':'---..', '9':'----.',

                    '0':'-----'}

    #Only morse symbols that are allowed

    \_\_valid\_morse = [',', ' ', '\n', '.', '-']

    #When class is initiated, save message for encoding and a message for decoding

    def \_\_init\_\_(self,  stop\_words\_path=None, letter\_message=None, morse\_message = None, write\_file = None):

        #Message to encode

        self.letter\_message = letter\_message

        #File path to write to

        if write\_file != None:

            if  write\_file[-4:] != '.txt':

                raise ValueError('File is invalid or not set. Please set a proper .txt file using set\_write\_file()')

            else:

                self.\_\_write\_file = write\_file

        else:

            self.\_\_write\_file = None

        #Contains a message to be decoded

        if morse\_message != None:

            for letter in morse\_message:

                #Check if txt file only contains allowed data '.'  ','  '-'  '\n'  and  'space'

                if letter not in self.\_\_valid\_morse:

                    raise ValueError("Invalid Values. Please have only '.' ',' '\\n' and ' ' in your message ")

        self.\_\_morse\_message = morse\_message

        #Stop words file path or list

        #List for custom quick stop words

        #Ensure all lowercase

        if stop\_words\_path == None:

            self.\_\_stop\_words = None

        elif isinstance(stop\_words\_path, list):

            self.\_\_stop\_words = list(map(lambda x: x.lower(), stop\_words\_path))

        #Accepts a file of stop-words

        elif not os.path.isfile(stop\_words\_path):

            raise FileNotFoundError("Please enter a valid path or input a custom python list")

        #Stop words from a file

        else:

            with open(current\_dir+'\\'+ stop\_words\_path, 'r') as f:

                lines = f.read()

            self.\_\_stop\_words = list(map(lambda x: x.lower(), lines.split('\n')))

    #Encoder for a full message

    def encodeMorse(self):

        encoded = ""

        if self.letter\_message == None:

            raise  ValueError('Object letter message to be encoded not found use set\_letter\_message method')

        #For each word in the sentence

        for i, item in enumerate(self.letter\_message.split(' ')):

            #For each letter in the word

            for letter in item.upper():

                if letter.isalnum() and letter in self.\_\_codes.keys():

                    encoded += self.\_\_codes[letter] + ","

                else:

                    encoded += letter

            if i != len(self.letter\_message.split(' '))-1:

                encoded += ' '

        #Return encoded message

        return encoded.rstrip(',').lstrip(',')

    #Morse code decoder

    def decodeMorse(self):

        decoded = ""

        #If morse message does not exist

        if self.\_\_morse\_message == None:

            raise ValueError('Object morse\_message to be decoded not found use set\_morse\_message() method')

        message = self.\_\_morse\_message.split('\n')

        for line in range(len(message)):

            #Split into word

            for word in message[line].split(' '):

                #Remove trailing comma

                for letter in list(filter(None,word.split(','))):

                    for key, value in Morse.\_\_codes.items():

                        if letter == value:

                            decoded += key

                decoded += " "

            #Check if the line is less than message length to add '\n'

            if line < len(message)-1:

                decoded += '\n'

        return decoded

    #Message analyzer method

    def analyze(self):

        final\_message = ""

        print('\n>>> Analysis and sorting has started'+'\n')

        #Decode of morse code

        final\_message +='\*\*\* Decoded Morse Text\n'

        decoded\_message = self.decodeMorse()

        final\_message += decoded\_message +'\n\n'

        #Returns unique set of words

        unique = Morse.get\_unique(decoded\_message)

        #Create a word sorter list and locate each words coordinates and insert to WordSorter() as Word() object

        sort\_list = WordSorter()

        for item in unique:

            coordinates = Morse.get\_coordinates(item, decoded\_message)

            word = Word(item, coordinates)

            sort\_list.insert(word)

        #Convert sort\_list to an actual list() of Word() objects with their frequencies in sublists

        sort\_list = sort\_list.node\_to\_list()

        #In each frequency group

        for frequency\_grp in sort\_list:

            final\_message += f'\*\*\* Morse Words with frequency=> {frequency\_grp[0].get\_frequency()}\n'

            #For each word in each frequency group

            for word in frequency\_grp:

                #Focus on the word and encode that word

                self.letter\_message= word.get\_word()

                final\_message += self.encodeMorse().rstrip(',').lstrip(',')+ "\n"

                final\_message += str(word) + '\n'

            final\_message += '\n'

        #Another word sorter list, but this is for Coord\_Word() subclass

        essential = WordSorter()

        filtered = self.\_\_filter\_stop\_words(list([item for sub\_list in sort\_list for item in sub\_list]))

        for i in filtered:

            essential.insert(i.word\_to\_coordword())

        essential\_message = ""

        essential = [item for sub\_list in essential.node\_to\_list() for item in sub\_list]

        #get essential message

        for i in essential:

            essential\_message += i.get\_word()

            essential\_message += " "

        final\_message += "\*\*\* Essential Message\n" + essential\_message

        print(final\_message)

        return final\_message

    #Filter out the recognized stop words from mesage

    def \_\_filter\_stop\_words(self, decoded\_message):

        important = []

        if self.\_\_stop\_words == None:

            raise ValueError('Object stop word reference list not found use set\_stop\_words() method')

        #For each word in the decoded message, check if exist in stop word list

        for word in decoded\_message:

            if word.get\_word().lower() not in self.\_\_stop\_words:

                important.append(word)

        #Return list of important words only

        return important

    #Write a text file

    def write\_file(self, message):

        #Ensure that the text file is valid

        if self.\_\_write\_file == None or self.\_\_write\_file[-4:] != '.txt':

            raise ValueError('File is invalid or not set. Please set a proper .txt file using set\_write\_file()')

        if os.path.exists(os.path.split(self.\_\_write\_file)[0]):

            with open(self.\_\_write\_file, 'w') as f:

                f.write(message)

        else:

            os.makedirs(os.path.split(self.\_\_write\_file)[0])

            with open(self.\_\_write\_file, 'w') as f:

                f.write(message)

    #Request user morse message input from a file

    def request\_morse\_from\_file(self):

        #Checks the read path

        readpath = ''

        while not os.path.isfile(readpath):

            readpath = input('\nPlease enter input file: ')

            #Try again if file path is non-existant

            if not os.path.isfile(readpath):

                print("Path does not exist")

        with open(current\_dir + '\\' + readpath, 'r') as f:

            lines = f.read()

            self.set\_morse\_message(lines)

    #Request where to save content to file

    def request\_write\_file(self):

        savepath = ''

        while savepath[-4:] != '.txt':

            savepath = input('Please enter output file: ')

            #If path does not end with a valid .txt extention

            try:

                self.set\_write\_file(savepath)

            except ValueError:

                print("Please enter a valid file extension .txt")

    #Default Morse does not have printing

    def \_\_str\_\_(self):

        return "Base Morse class has no printing, use MorseHorizontal() or MorseVertical()"

    #Set methods

    #Set message to be decoded

    def set\_morse\_message(self, morse\_message):

        for letter in morse\_message:

            #Check if txt file only contains allowed data '.'  ','  '-'  '\n'  and  'space'

            if letter not in self.\_\_valid\_morse:

                raise ValueError("Invalid Values in file. Please have a Text file with only '.' ',' '\\n' and ' ' ")

        self.\_\_morse\_message = morse\_message

    #Edit stop words (custom list or using a file)

    def set\_stop\_words(self, stop\_words\_path):

        #Check if list or file path before setting stop words

        if isinstance(stop\_words\_path, list):

            self.\_\_stop\_words = list(map(lambda x: x.lower(), stop\_words\_path))

        elif not os.path.isfile(stop\_words\_path):

            raise FileNotFoundError("Please enter a valid path or input a custom python list")

        else:

            with open(current\_dir+'\\'+ stop\_words\_path, 'r') as f:

                lines = f.read()

            self.\_\_stop\_words = lines.split('\n')

    #Set a write file

    def set\_write\_file(self, file\_path):

        #If path does not end with a valid .txt extention

        if file\_path[-4:] != '.txt':

            raise ValueError("Please enter a valid file extension .txt")

        else:

            self.\_\_write\_file = file\_path

    #Get methods for attribute retrieval

    def get\_morse\_message(self):

        return self.\_\_morse\_message

    def get\_stop\_words(self):

        return self.\_\_stop\_words

    def get\_write\_file(self):

        return self.\_\_write\_file

    def get\_valid\_morse(self):

        return self.\_\_valid\_morse

    #See translation sheet of Morse object

    def get\_codes(self):

        return self.\_\_codes

    #Change global morse values that are allowed (list of values)

    @classmethod

    def set\_valid\_morse(cls, morse\_list):

        if isinstance(morse\_list, list):

            cls.\_\_valid\_morse = morse\_list

        else:

            raise TypeError("Please enter a list")

    #Static methods

    #Meant for 2D list coordinates retrieval for each word

    @staticmethod

    def get\_coordinates(item, decoded\_message):

        source\_list = [list(filter(None, i.split(' '))) for i in decoded\_message.upper().split('\n')]

        coordinates = []

        #Retrieve row and column indexes of word

        for row, sub\_list in enumerate(source\_list):

            for column in range(len(sub\_list)):

                if sub\_list[column] == item.upper():

                    #Append coordinates

                    coordinates.append((row, column))

        #Returns list of coordinates of word in original message

        return coordinates

    #Get all unique values in decoded mesage

    @staticmethod

    def get\_unique(decoded\_message):

        decoded\_array = [list(filter(None, i.split(' '))) for i in decoded\_message.split('\n')]

        unique = set([item.upper() for sub\_list in decoded\_array for item in sub\_list])

        return unique

#Morse class that supports horizontal printing

class MorseHorizontal(Morse):

    #Convert Horizontal Morse to Vertical Morse class

    def hori\_to\_verti(self):

        return MorseVertical(self.get\_stop\_words(), self.letter\_message, self.get\_morse\_message(), self.get\_write\_file())

    def \_\_str\_\_(self) :

        return self.encodeMorse()

#Morse class that supports vertical printing

class MorseVertical(Morse):

    def \_\_str\_\_(self):

        maxlen = 0

        letters = []

        #Split word from sentence

        for word in self.encodeMorse().split():

            #Split alphabets in each word

            for letter in word.split(','):

                #Get max length/ longest alphabet

                if len(letter) > maxlen:

                    maxlen = len(letter)

                letters.append(letter)

            for i in range(len(letters)):

                for n in range(maxlen - len(letters[i])):

                    letters[i] = " " + letters[i]

        final\_line = ""

        for ix in range(maxlen): #How many words are there

            for w in letters:

                final\_line += w[ix]

            final\_line += "\n"

        return final\_line

    #Convert Vertical Morse to Horizontal Morse

    def verti\_to\_hori(self):

        return MorseHorizontal(self.get\_stop\_words() ,self.letter\_message, self.get\_morse\_message(), self.get\_write\_file())

WordSorter.py (Linked list and sorting objects for analyzer tool)

####################################################################

# ST1507 DSAA: MorseCode Message Analyzer (Morse Class)            #

#------------------------------------------------------------------#

#                                                                  #

# - Name: Aw Shao Yang                                             #

# - Class: DAAA/FT/2B/03                                           #

# - Admission Number: p2012126                                     #

####################################################################

#Node for sorted linked list Classes

class Node:

    def \_\_init\_\_(self):

        self.\_\_nextNode = None

    def get\_nextNode(self):

        return self.\_\_nextNode

    def set\_nextNode(self, next\_node):

        self.\_\_nextNode = next\_node

#Coordinate Class Meant for comparing letter placements

class Coordinate:

    #Ony private attributes users can only set the values when initializing object

    def \_\_init\_\_(self, row, column):

        self.\_\_row = row          #Row of item

        self.\_\_column = column    #Column of item

    #Less than comparison

    def \_\_lt\_\_(self, other\_coord):

        #If the row is lesser

        if self.\_\_row < other\_coord.\_\_row:

            return True

        #If the row is the same but column is lesser

        if self.\_\_column < other\_coord.\_\_column and self.\_\_row == other\_coord.\_\_row:

            return True

        else:

            return False

    #Greater than comparison

    def \_\_gt\_\_(self, other\_coord):

        #If the row is lesser

        if self.\_\_row > other\_coord.\_\_row:

            return True

        #If the row is the same but column is lesser

        if self.\_\_column > other\_coord.\_\_column and self.\_\_row == other\_coord.\_\_row:

            return True

        else:

            return False

    def \_\_str\_\_(self):

        return f'Coordinate({self.\_\_row},{self.\_\_column})'

#Word object meant for easy word sorting

class Word(Node):

    #Attributes are all privatized so that people cannot set them by themselves,

    #Values are set only when inserted

    def \_\_init\_\_(self, word=None, placement=[]):

        self.\_\_word = word                    #Associated word

        self.\_\_frequency = len(placement)     #Word frequency is deduced from placement list

        self.\_\_length = len(word)             #Word length is length of word

        self.\_\_placement = placement          #Word coordinates

        super().\_\_init\_\_()

    #Greater than comparison for word

    def \_\_gt\_\_(self, otherword):

        #If word is more frequent, if equal then if word is shorter, if both equal then alphabets are earlier

        if self.\_\_frequency > otherword.\_\_frequency:

            return True

        elif self.\_\_length < otherword.\_\_length and self.\_\_frequency == otherword.\_\_frequency:

            return True

        elif self.\_\_word < otherword.\_\_word and self.\_\_frequency == otherword.\_\_frequency and self.\_\_length == otherword.\_\_length:

            return True

        else:

            return False

    #Lesser than comparison for word

    def \_\_lt\_\_(self, otherword):

        #If word is less frequent, if equal then if word is longer, if both equal then alphabets are later

        if self.\_\_frequency < otherword.\_\_frequency:

            return True

        elif self.\_\_length > otherword.\_\_length and self.\_\_frequency == otherword.\_\_frequency:

            return True

        elif self.\_\_word > otherword.\_\_word and self.\_\_frequency == otherword.\_\_frequency and self.\_\_length == otherword.\_\_length:

            return True

        else:

            return False

    #Print word e.g [HELLO] (6) [(1,2), (2,3), (4,4)]

    def \_\_str\_\_(self):

        return f"[{self.\_\_word}] ({self.\_\_frequency}) {self.\_\_placement}"

    def \_\_repr\_\_(self):

        return repr((self.\_\_word, self.\_\_frequency, self.\_\_length))

    #Convertion of word to Coordinate word

    def word\_to\_coordword(self):

        return Coordinate\_Word(self.\_\_word, self.\_\_placement)

    #getters for subclasses and MorseTool

    def get\_frequency(self):

        return self.\_\_frequency

    def get\_placement(self):

        return self.\_\_placement

    def get\_word(self):

        return self.\_\_word

#Coordinate Word is meant for easy word sorting in WordSorter

#Sort based on frequency and 1st coordinate in placement [] list

class Coordinate\_Word(Word):

    def \_\_gt\_\_(self, otherword):

        #If frequency is higher and if equal, word appears earlier

        if self.get\_frequency() > otherword.get\_frequency():

            return True

        if Coordinate(self.get\_placement()[0][0], self.get\_placement()[0][1]) < Coordinate(otherword.get\_placement()[0][0], otherword.get\_placement()[0][1]) and self.get\_frequency() == otherword.get\_frequency():

            return True

        else:

            return False

    def \_\_lt\_\_(self, otherword):

        #If frequency is lower and if equal, word appears later

        if self.get\_frequency < otherword.get\_frequency():

            return True

        if Coordinate(self.get\_placement()[0][0], self.get\_placement()[0][1]) > Coordinate(otherword.get\_placement()[0][0], otherword.get\_\_placement()[0][1]) and self.get\_frequency() == otherword.get\_frequency():

            return True

        else:

            return False

#Linked List meant for sorting words upon insert

class WordSorter:

    #Values are privatized as they should'nt be set from outside and sorted only upon insert

    def \_\_init\_\_(self):

        self.\_\_headWord = None

        self.\_\_length = 0

    #Add as the head of the list

    def \_\_appendToHead(self, newWord):

        oldHeadWord = self.\_\_headWord

        self.\_\_headWord = newWord

        self.\_\_headWord.set\_nextNode(oldHeadWord)

        self.\_\_length += 1

    #Insertion of new items will be sorted as follows

    def insert(self, newWord):

        self.\_\_length += 1

        # If list is currently empty the first node is the new node

        if self.\_\_headWord == None:

            self.\_\_headWord = newWord

            return

        #If the new node is higher than head, assign it

        if newWord > self.\_\_headWord:

            self.\_\_appendToHead(newWord)

            return

        # Check it is going to be inserted between any pair of Nodes (left,right)

        leftWord = self.\_\_headWord

        rightWord = self.\_\_headWord.get\_nextNode()

        #While the list has not reached the end

        while rightWord != None:

            #If the new node is less than the right node do reassignment

            if newWord > rightWord:

                leftWord.set\_nextNode(newWord)

                newWord.set\_nextNode(rightWord)

                return

            #If it is still more than the right node move down the list

            leftWord = rightWord

            rightWord = rightWord.get\_nextNode()

        #If last, then just add to tail

        leftWord.set\_nextNode(newWord)

    def \_\_str\_\_(self):

    # We start at the head

        output =""

        node= self.\_\_headWord

        firstWord = True

        while node != None:

            if firstWord:

                output = "<" + node.\_\_str\_\_()

                firstWord = False

            else:

                output += (',' + node.\_\_str\_\_())

            node= node.get\_nextNode()

        return output + ">"

    #Meant for conversion of sorted linked list to normal py list, with sublists based on frequency

    def node\_to\_list(self):

        word\_list = []

        node = self.\_\_headWord

        while node != None:

            word\_list.append(node)

            node = node.get\_nextNode()

        #Keeps only unique frequency values

        data = list(set([data.get\_frequency() for data in word\_list]))

        #Sort the frequencies

        getLength = sorted(data, reverse=True)

        result = []

        #For each frequency

        for length in getLength:

            #Create a list of elements with the same frequency

            frequency\_groups = [data for data in word\_list if length == data.get\_frequency()]

            result.append(frequency\_groups)

        return result