Project Objectives

The goal of this project was to write a program to reimplement the Morse Code MATLAB code, from *Experiments with MATLAB* (Moler, 2011) in Python. The success of the project was tested by the translation exercise (19.1) in the MATLAB pdf and additional encoding (Alphabetic input to Morse Code output) and decoding (Morse Code input to Alphabetic Output) experiments listed in the Results section of this report. This was an individual project and therefore, there are no team contributions to be listed.

The original MATLAB program worked by defining a tree data structure, implemented with cell arrays, and iterating through it to find the appropriate symbols corresponding to the input provided by the user. This project reimplemented the tree data structure by defining a Python class, "leaf," and using it to define each leaf in the tree and its place in the tree by its relationship (branches) to other leaves. The tree structure was then used successfully for encoding and decoding tasks.

Techniques and Tools

Implementing the Tree Data Structure

The primary technique used for this project was a Python class. A class, leaf, was defined allowing the objects it created to have a value, which was assigned as the empty string for the root leaf and as a letter for the subsequent leaves. It has allowed each leaf to, optionally, have attributes 'leftChild' and 'rightChild.' These attributes were used to define mother-duaghter relationships between leaves. By differentiating 'left' and 'right' this allowed the relationship to contain a distinction between two daughters such that a left daughter can be associated with a dot and a right daughter can be associated with a dash (see Fig. 1).

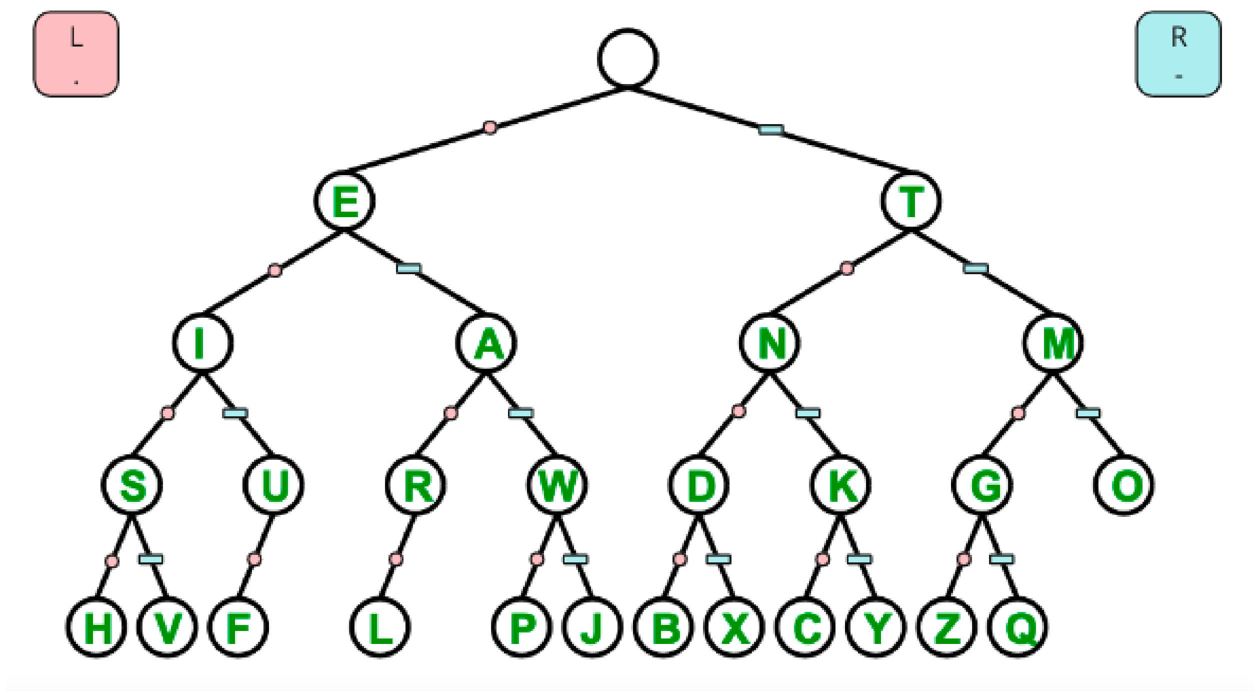


Figure 1. Image from Experiments with MATLAB (Moler, 2011) *with dots, dashes, and associated key added for clarity.*

For example: to make the leaf for “E,” the root (called mcTreeRoot) is given the attribute ‘leftChild’ which is assigned to a leaf given the value which is the string “E.”

```
mcTreeRoot.leftChild = leaf(“E”)
```

Encoding

Encoding takes Alphabetic input and outputs the same message in Morse Code. For each letter in the input, the leaf with the corresponding value would be identified and from there it was determined to be either a left or right child; if it was a left child a dot (“.”) would be collected and if it was a right child, then a dash (“-”). Then, the mother of that leaf (the leaf above it) would also be determined to be a left or right child and the appropriate dot or dash would be added to the front of the symbols being collected (since we are working from the leaves to the root, the Morse Code symbols are being collected backwards and must be added to the front). This continues until the root is reached.

Decoding

Decoding takes Morse Code input and outputs the same message in Alphabetic characters. Starting at the root leaf, for each dot or dash in the input, the program moves down to the leaf down the associated branch; if the symbol is a dot, it moves to the left child of the current leaf, and if it is a dash, it moves to the right child of the current leaf. If the character in the input is a space (“ “), the value of the current leaf is collected. This process is repeated for the entirety of the input string.

Input Handling and Error Prevention

To avoid errors from improper input, all input was converted to uppercase. Furthermore, regular expressions were used to determine if the input was appropriate for the current task. If encoding, the input was matched against a pattern for only alphabetic characters and spaces. If it does not match, the user receives a message indicating that they should use only those characters, and then they can try again. For decoding, this same technique was used, but the input is matched against a pattern for only dots (“.”), dashes (“-”), and spaces (“ “).

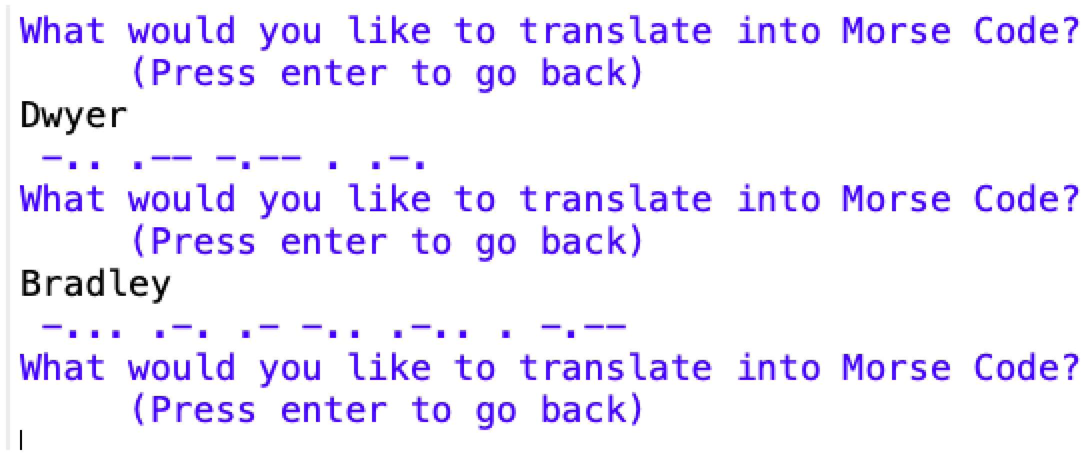
User Friendliness and Instructions

The program includes messages and instructions to accommodate multiple users, regardless of previous knowledge of the program. The main function allows the user to specify if they would like to decode or encode. Once they have picked, they can continue to enter input or the press "Enter" (input the empty string "") to go back and choose between decode or encode again, or to press "Enter" again and quit the program. If invalid input is entered in either the encode or decode function, the program will display a message explaining the issue and indicating the acceptable types of input.

Results

Encoding

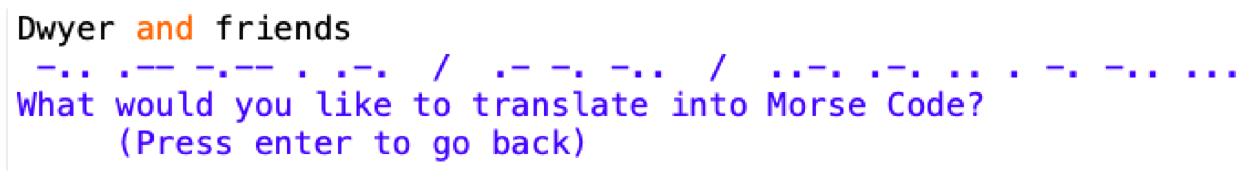
The program successfully encodes Alphabetic messages to Morse Code, as shown in Fig. 2 below.



```
What would you like to translate into Morse Code?
     (Press enter to go back)
Dwyer
 -.. .-- -.-- . .-.
What would you like to translate into Morse Code?
     (Press enter to go back)
Bradley
 -... .-. .- -.. .-.. . -.--
What would you like to translate into Morse Code?
     (Press enter to go back)
```

Figure 2. Screenshot of program running in IDLE Shell displaying encoding. Black text is user input. Blue text is program output.

The program also successfully encodes multi-word Alphabetic messages to Morse Code, as shown in Fig. 3 below. The program includes slash between words to facilitate easier reading.

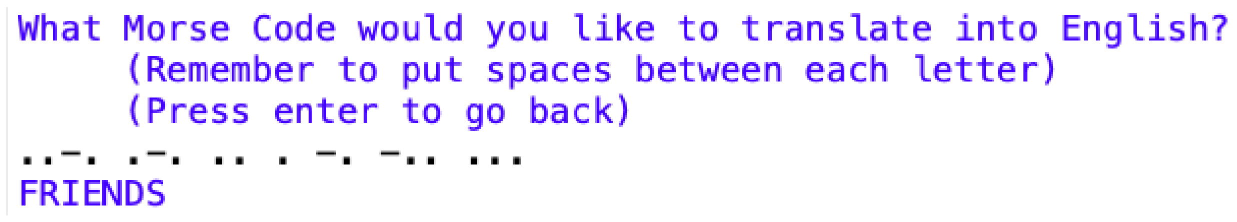


```
Dwyer and friends
 -.. .-- -.-- . .-.  /  .- -. -..  /  ..-. .-. .. . -. -.. ...
What would you like to translate into Morse Code?
     (Press enter to go back)
```

Figure 3. Screenshot of program running in IDLE Shell displaying encoding of multi-word input. Black and orange text is user input. Blue text is program output.

Decoding

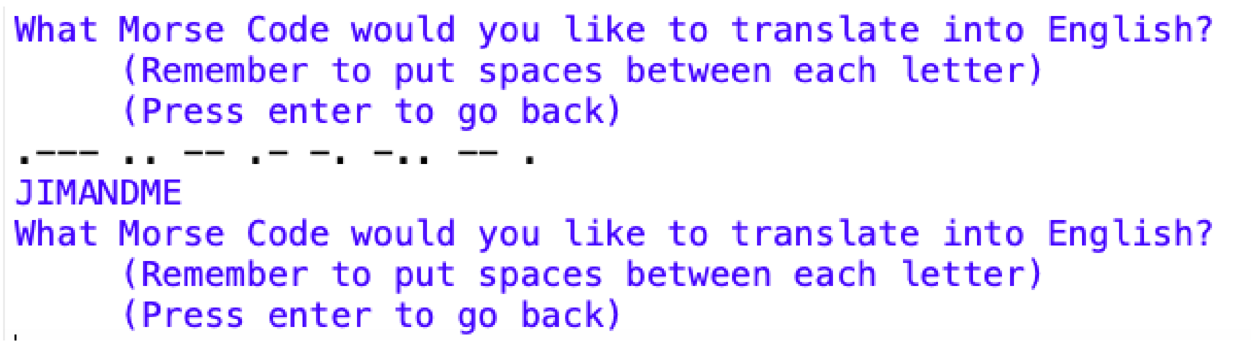
The program successfully decodes Morse Code messages to Alphabetic characters, as shown in Fig. 4 below. However, the program has a limitation that the user must include spaces between each letter.



```
What Morse Code would you like to translate into English?
     (Remember to put spaces between each letter)
     (Press enter to go back)
..-. .-. .. . -. -.. ...
FRIENDS
```

Figure 4. Screenshot of program running in IDLE Shell displaying decoding. Black text is user input. Blue text is program output.

A further limitation is found in decoding multi-word messages. The program will succeed in outputting the correct letters; however it will do so without spaces between words (see Fig. 5). This is an area for improvement and will likely involve allowing the user to input slashes where they intend spaces to be.

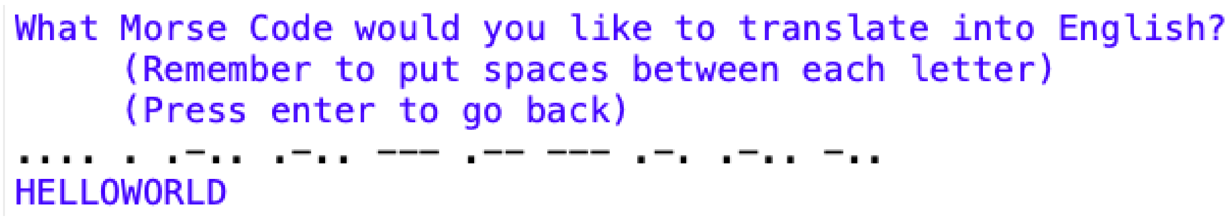


```
What Morse Code would you like to translate into English?
     (Remember to put spaces between each letter)
     (Press enter to go back)
.--- .. -- .- -. -.. -- .
JIMANDME
What Morse Code would you like to translate into English?
     (Remember to put spaces between each letter)
     (Press enter to go back)
```

Figure 5. Screenshot of program running in IDLE Shell displaying decoding of multi-word input. Black text is user input. Blue text is program output. Output does not contain spaces between words.

Testing with Experiments with MATLAB (Moler, 2011)

To test that this program can correctly handle the same type of input as the MATLAB implementation it is based on, it was tested with exercise 19.1 from Moler 2011. It successfully decoded the message as, "HELLOWORLD." See Fig. 6 below.



```
What Morse Code would you like to translate into English?
     (Remember to put spaces between each letter)
     (Press enter to go back)
.... . .-.. .-.. --- .-- --- .-. .-.. -..
HELLOWORLD
```

Figure 6. Screenshot of program running in IDLE Shell displaying decoding of exercise 19.1 from Experiments with MATLAB (Moler, 2011). *Black text is user input. Blue text is program output.*

Conclusion

Despite some shortcomings regarding output format for multi-word input as well as strictness in the spacing required from the user for Morse Code input, the program successfully outputs the appropriate Alphabetic characters in decoding and the appropriate Morse Code symbols in encoding. Furthermore, the tree data structure was implemented successfully and can be used to perform the required tasks: encoding and decoding. Additionally, the class, “leaf,” can be used in future programs to define tree data structures as needed. This provides efficient means to represent syntactic hierarchies for theoretical and computational linguistic research.

References

Moler, C. B. (2011). *Experiments with MATLAB*. Society for Industrial and Applied Mathematics.