Morse Code Translator in Python

INT 213 report

Submitted by

Sudip Kumar Mandal

12019167

&

Rohan Saraswat

12008941

B-Tech CSE (3rd semester)

School of Computer Science and Engineering

Submitted on: 29 November 2021

Abstract

Morse code is:

- an alphabet or code in which letters are represented by combinations of long and short light or sound signals.
- a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called *dots* and *dashes*, or *dits* and *dahs*
- named after **Samuel Morse**, one of the inventors of the telegraph

The *dit* duration is the basic unit of time measurement in Morse code transmission. The duration of a *dah* is three times the duration of a *dit*. Each *dit* or *dah* within an encoded character is followed by a period of signal absence, called a *space*, equal to the *dit* duration. The letters of a word are separated by a space of duration equal to three *dits*, and words are separated by a space equal to seven *dits*.

This code takes input from the user in the form of string. It may be English or Morse code. The user is required to set the following parameters according to his/her needs:

1. Select the mode
    a. English to Morse(default)
    b. Morse to English
2. Enter the symbols for
    a. short sign(default: '.')
    b. long sign(default: '-')
3. Choose display format
    a. International Morse Code format(default)
    b. Compact format
4. String type
    a. Enter string(default)
    b. Access the string from a file

After entering the string and choosing these options, the 'submit' button has to be pressed after which the translated code is displayed.

Creating database

Database containing alphabets, numbers and corresponding Morse codeis required. Dictionary is chosen for this purpose. The dictionary is a data structure consisting of key-value pairs. Two dictionaries are required for this purpose:

- Dictionary containing keys as alphabets and numbers and values as Morse code.
- Dictionary containing keys as Morse code and values as alphabets and numbers.

The symbols for the Morse code can vary according to the user, so the code was written as 0's for short sign and 1's for long sign, as it could be easily converted into other symbols.

It would be time consuming to enter the key-value pair one by one as:

```
d_eng_morse = {}

d_eng_morse['a'] = '01'

d_eng_morse['b'] = '1000'

d_eng_morse['c'] = '1010'
```

… and so on

So, instead the values were written in the easiest format in just a string as:

(string containing Morse code for alphabets)

```
s_alpha = 'a.01 b.1000 c.1010 d.100 e.0 f.0010 g.110 h.0000 i.00 j.0111 k.101
l.0100 m.11 n.10 o.111 p.0110 q.1101 r.010 s.000 t.1 u.001 v.0001 w.011 x.1001
y.1011 z.1100 '
```

(string containing Morse code for numbers)

```
s_num = '1.01111 2.00111 3.00011 4.00001 5.00000 6.10000 7.11000 8.11100 9.11110
0.11111 '
```

The two strings are then concatenated as:

s_alphanum = s_alpha + s_num

(string containing Morse code for both alphabets and numbers)

The string is then split into list containing the English-Morse pair as one attribute:

```
l_alphanum = s_alphanum.split()
```

The resulting list is:

```
['a.01', 'b.1000', 'c.1010', 'd.100', 'e.0', 'f.0010', 'g.110', 'h.0000', 'i.00',
'j.0111', 'k.101', 'l.0100', 'm.11', 'n.10', 'o.111', 'p.0110', 'q.1101',
'r.010', 's.000', 't.1', 'u.001', 'v.0001', 'w.011', 'x.1001', 'y.1011',
'z.1100', '1.01111', '2.00111', '3.00011', '4.00001', '5.00000', '6.10000',
'7.11000', '8.11100', '9.11110', '0.11111']
```

The list is then converted into a dictionary as:

```
d_eng_morse = {}

for i in range(len(l_alphanum)):

d_eng_morse[l_alphanum[i].split('.')[0]] = l_alphanum[i].split('.')[1]
```

Here:

1. `l_alphanum[i].split('.')` contains list split into English and Morse using the delimiter '.' as: ['a','01'] for i=0.
2. `l_alphanum[i].split('.')[0]` contains the first element of the list i.e., 'a', for i=0.
3. This element 'a' is stored in the key section of dictionary d_eng_morse as: `d_eng_morse[l_alphanum[i].split('.')[0]]`
4. `l_alphanum[i].split('.')[1]` returns the second element of the list `l_alphanum[i].split('.')` i.e., '01', for i=0 and is this is stored in the value section of the dictionary d_eng_morse
5. This process is repeated for ever element of the list: l_alphanum

For example, taking i=0, we have:

1. `l_alphanum[0]                             = 'a.01'`
2. `l_alphanum[0].split('.')                  = ['a', '01']`
3. `l_alphanum[0].split('.')[0]               = 'a'`
4. `l_alphanum[0].split('.')[1]               = '01'`
5. `d_eng_morse[l_alphanum[i].split('.')[0]]  = d_eng_morse['a']`
6. Finally, the statement `d_eng_morse[l_alphanum[i].split('.')[0]] = l_alphanum[i].split('.')[1]` is equivalent to: `d_eng_morse['a'] = '01'`

The resulting dictionary is:

```
{'a': '01', 'b': '1000', 'c': '1010', 'd': '100', 'e': '0', 'f': '0010', 'g':
'110', 'h': '0000', 'i': '00', 'j': '0111', 'k': '101', 'l': '0100', 'm': '11',
'n': '10', 'o': '111', 'p': '0110', 'q': '1101', 'r': '010', 's': '000', 't':
'1', 'u': '001', 'v': '0001', 'w': '011', 'x': '1001', 'y': '1011', 'z': '1100',
'1': '01111', '2': '00111', '3': '00011', '4': '00001', '5': '00000', '6':
'10000', '7': '11000', '8': '11100', '9': '11110', '0': '11111'}
```

The key and value position of the dictionary is then reversed to obtain a dictionary containing keys as Morse code and values as alphabets and numbers.

This was done as:

```
d_morse_eng = {}

for i in d_eng_morse:

d_morse_eng[d_eng_morse[i]] = i
```

Here,

- `d_eng_morse[i]]` returns the value of the dictionary for the key i
- `d_morse_eng[d_eng_morse[i]] = i` assigns the value i to the key of the dictionary d_morse_eng, i.e., d_eng_morse

`d_morse_eng[d_eng_morse[i]] = i`

(new dictionary) (old dictionary, returns Morse code) (English)

The resulting dictionary is:

```
{'01': 'a', '1000': 'b', '1010': 'c', '100': 'd', '0': 'e', '0010': 'f', '110':
'g', '0000': 'h', '00': 'i', '0111': 'j', '101': 'k', '0100': 'l', '11': 'm',
'10': 'n', '111': 'o', '0110': 'p', '1101': 'q', '010': 'r', '000': 's', '1':
't', '001': 'u', '0001': 'v', '011': 'w', '1001': 'x', '1011': 'y', '1100': 'z',
'01111': '1', '00111': '2', '00011': '3', '00001': '4', '00000': '5', '10000':
'6', '11000': '7', '11100': '8', '11110': '9', '11111': '0'}
```

Creating translator function

This function takes input in the form of string and some other parameters and returns translated string. It performs both the work:

- translate English to Morse
- translate Morse to English

Morse is a very versatile language in which, the short and long signal can be transmitted through various methods, like through sound pulses, light pulses or by representing them with symbols.

Here, the user has been given the opportunity to customize the short and long symbols.

According to the International Morse Code standards, the space between:

- parts of same letter: 1 unit
- letters: 3 units
- words: 7 units

This looks like a lot of spacing, so another scheme of spacing, the 'compact format' is provided according to which the space between:

- parts of same letter: 0 unit
- letters: 1 unit
- words: 3 units

The syntax for the function is:

```
translator(string, l, u, compact, em)
```

Where,

- string is the input text either in English or in Morse
  - input type: string
- l is the short sign
  - input type: character
- u is the long sign

    - input type: character
- compact is the spacing format
    - input type: boolean (0 or 1)
    - 0 -> compact spacing format
    - 1 -> International Morse Code spacing format
- em is the parameter which decides the mode of translation, English to Morse or vice-versa
    - input type: boolean (0 or 1)
    - 0 -> Morse to English
    - 1 -> English to Morse

English to Morse

1. Converting upper case letters(if any) to lower case: stringl = string.lower()
2. Reading English letters one by one
    a. If a space is encountered, a spacing equivalent to space between words is added
    b. If an alphabet or a number is encountered,
        i. it is converted to Morse code by accessing the dictionary d_eng_morse
        ii. the short and long sign is converted to user customized symbols
        iii. a space equivalent to the space between parts of same letter is added
    c. If anything else is encountered, the character ‘X’ is added to the output string
3. After converting each letter, a space equivalent to the space between letters is added

Since after every parts of same letter, letter and word, spaces are added, the spacing scheme had to be redefined as:

International Morse Code:

- parts of same letter: 1 unit
- letters: 3-1 = 2 units
- words: 7-3 = 4 units

Compact format:

- parts of same letter: 0 unit
- letters: 1-0 = 1 unit
- words: 3-1 = 2 units

For example:

Input:

string= ‘sample text’, l=’.’, u=‘-’, compact=0, em=1

Output:

```
. . .   . -   - -   . - - .   . - . .   .       -   .   - . . -   -
```

Morse to English

Input:

```
. . .   . -   - -   . - - .   . - . .   .       -   .   - . . -   -
```

Output:

```
sample text
```

Working:

- Converting the input string to list containing code for English words
  - `l_word = string.split(btw_word)`
  - List created for the sample input:
    `['. . .   . -   - -   . - - .   . - . .   .', '-   .   - . . -   -']`
- Traversing the list and creating another list containing code for English letters

  -
    ```
    for i in l_word:

            l_letter = i.split(btw_letter)
    ```
  - Lists created for the sample input:
    `['. . .', '. -', '- -', '. - - .', '. - . .', '.']`
    `['-', '.', '- . . -', '-']`
- Traversing each code for letter, converting their upper and lower sign to 1's and 0's
- Accessing the dictionary d_morse_eng to get the English equivalent for the Morse code and putting character 'X' for unrecognized symbols
- Adding a space after reading each word

Perks:

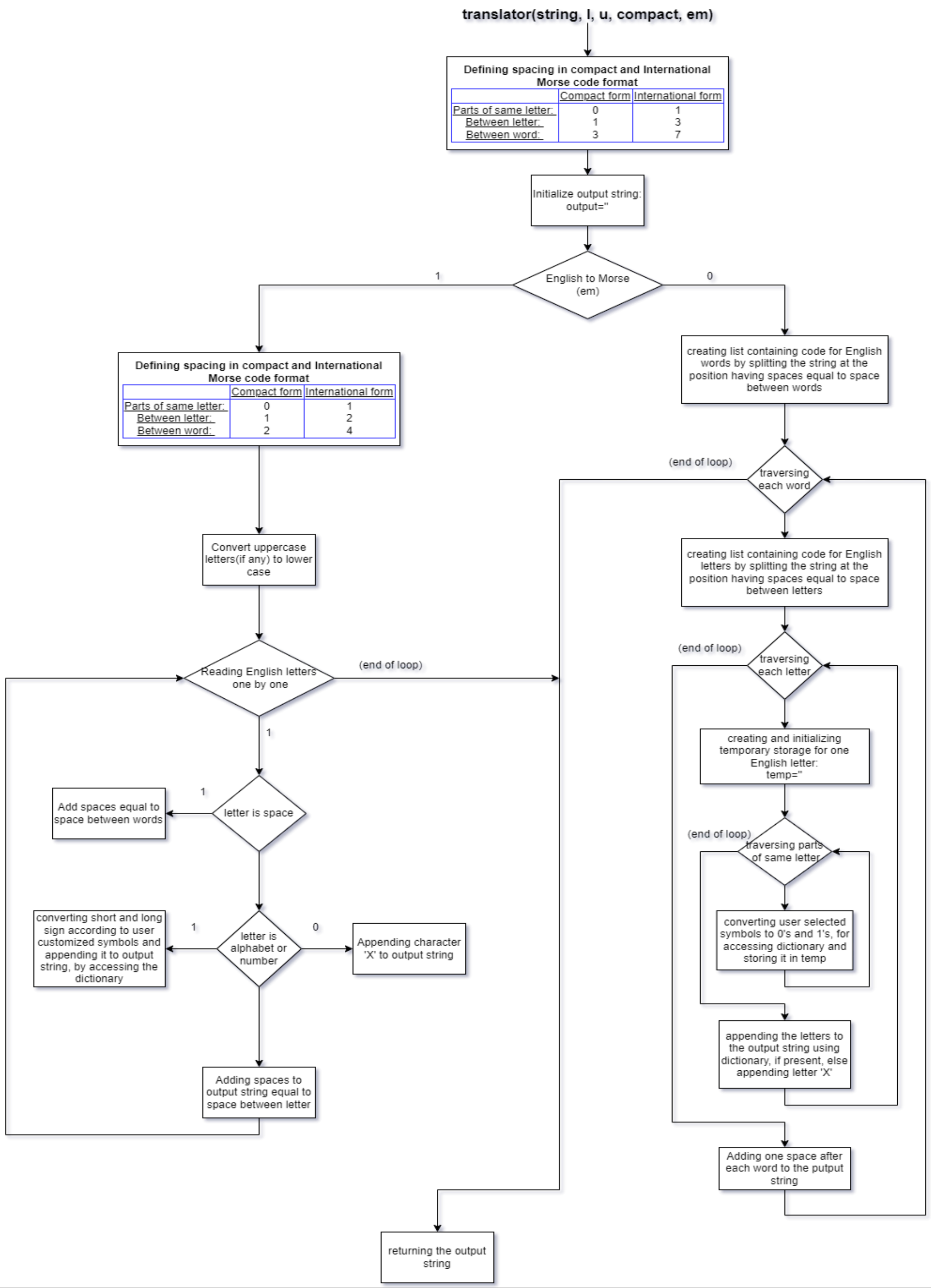
- Can do two-way conversion i.e., Morse to English and English to Morse
- Can give the output in compact format as well as in International Morse Code format
  - International:
    `. . .   . -   - -   . - - .   . - . .   .       -   .   - . . -   -`
  - Compact:
    `... .- -- .--. .-.. .   - . -..- -`
- Attaches the symbol 'X' for unrecognized characters
  - Does not stop the program, it continues to convert the remaining characters

  - For example:
  - Input: `ab $$ cd`
  - Output: `. -   - . . .       X   X       - . - .   - . .`
- It lets user decide the symbol for short and long sign
  - It is not limited to alphabets or numerals, but can be any ASCII character
    For example: l='☺', u='☻'
    Output: `☺ ☺ ☺   ☺ ☻   ☻ ☻   ☺ ☻ ☻ ☺   ☺ ☻ ☺ ☺   ☺       ☻   ☺   ☻ ☺ ☺ ☻ ☻`
  - In English to Morse mode, multiple characters can be given
    For example: `l='short', u='long'`
    Output: `short short short   short long   long long   short long long short   short long short short   short       long   short   long short short long   long`

Limitations:

- The multiple characters for short and long sign can only be given for English to Morse conversion; it does not work for Morse to English Mode.
- This function does not work in real time i.e., it does not display the output continuously while the input is given. The output is displayed only after the complete input is given.

The flow-chart for the translator function is given in the next page. It was created with the help of an online flow-chart creator tool drawio.



translator(string, l, u, compact, em)
Defining spacing in compact and International Morse code format
Compact form
International form
Parts of same letter: 0 1
Between letter: 1 3
Between word: 3 7
Initialize output string: output=''
English to Morse (em)
1
0
Defining spacing in compact and International Morse code format
Compact form
International form
Parts of same letter: 0 1
Between letter: 1 2
Between word: 2 4
Convert uppercase letters(if any) to lower case
Reading English letters one by one
(end of loop)
1
letter is space
1
Add spaces equal to space between words
letter is alphabet or number
1
0
converting short and long sign according to user customized symbols and appending it to output string, by accessing the dictionary
Appending character 'X' to output string
Adding spaces to output string equal to space between letter
creating list containing code for English words by splitting the string at the position having spaces equal to space between words
(end of loop)
traversing each word
creating list containing code for English letters by splitting the string at the position having spaces equal to space between letters
(end of loop)
traversing each letter
creating and initializing temporary storage for one English letter: temp=''
(end of loop)
traversing parts of same letter
converting user selected symbols to 0's and 1's, for accessing dictionary and storing it in temp
appending the letters to the output string using dictionary, if present, else appending letter 'X'
Adding one space after each word to the putput string
returning the output string

Files and exceptions handling

This module is created for accessing text from file and translating them.

Working:

1. The ‘file’ parameter is a Boolean value
    a. If file=1, read from file
    b. If file=0, translate the string as it is
2. During file handling,
    a. An user-defined exception is defined to check whether the file is empty
    b. A predefined exception IOError is used when no file is found
    c. If no exception is found, the file will be opened, read, and the text inside the file will be sent to translator function
3. If file=0, the string is sent to translator function as it is
4. The translator function returns converted string, which is then returned by this function

For example,

Consider two files:

- sample.txt (with contents: ‘aa bb .. 00’)
- sample2.txt (empty file)

Input 1:

- `string=’sample.txt’`
- `l=’.’`
- `u=’-‘`
- `compact=0`
- `em=1`
- `file=1`

Output:

```
. -   . -       - . . .   - . . .       X   X       - - - - -   - - - - -
```

Input 2:

- `string=’sample2.txt’`
- `l=’.’`
- `u=’-‘`
- `compact=0`
- `em=1`
- `file=1`

Output:

```
(empty file)
```

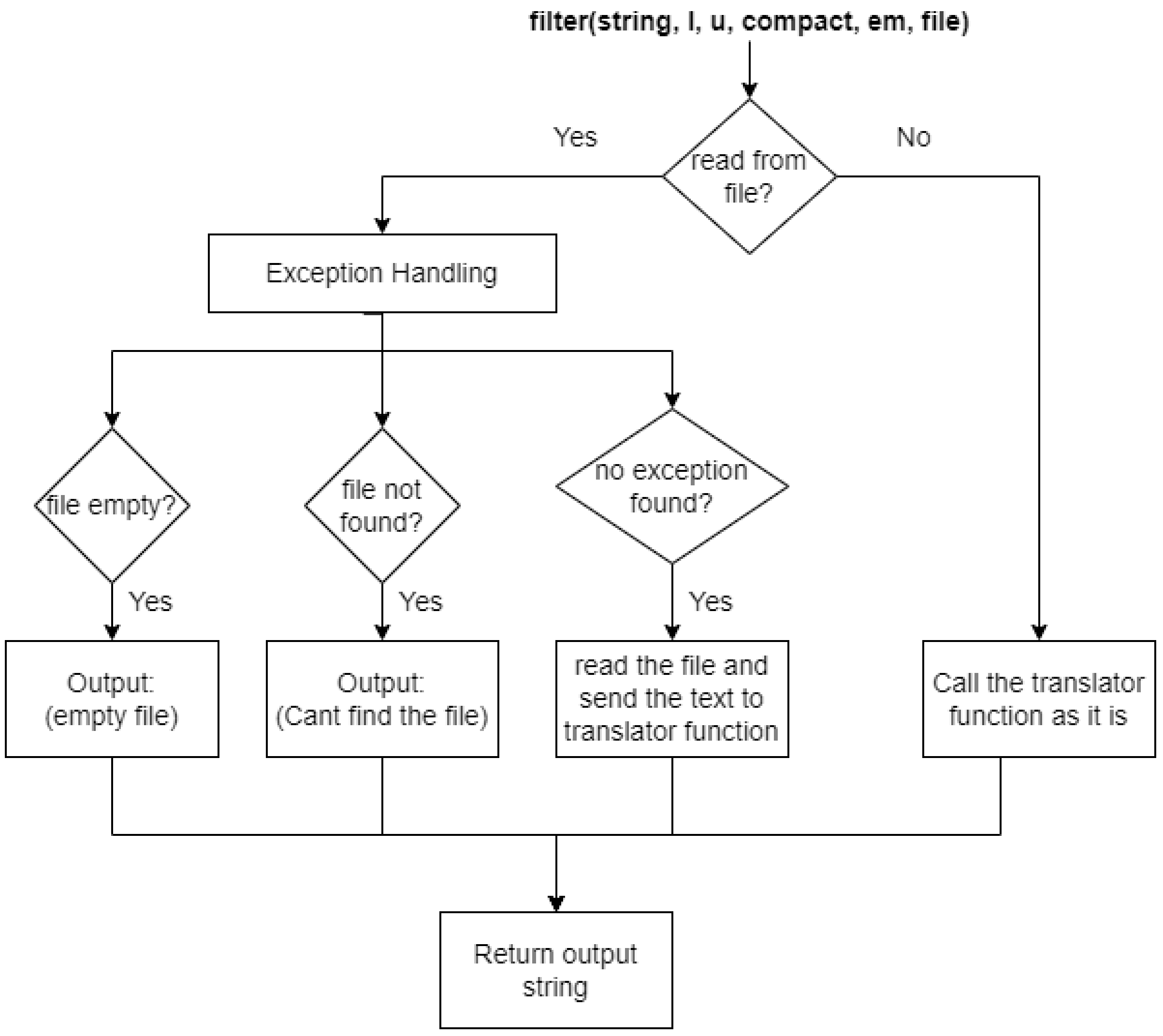
Input 3:

- `string=’sample3.txt’`
- `l=’.’`
- `u=’-‘`
- `compact=0`
- `em=1`
- `file=1`

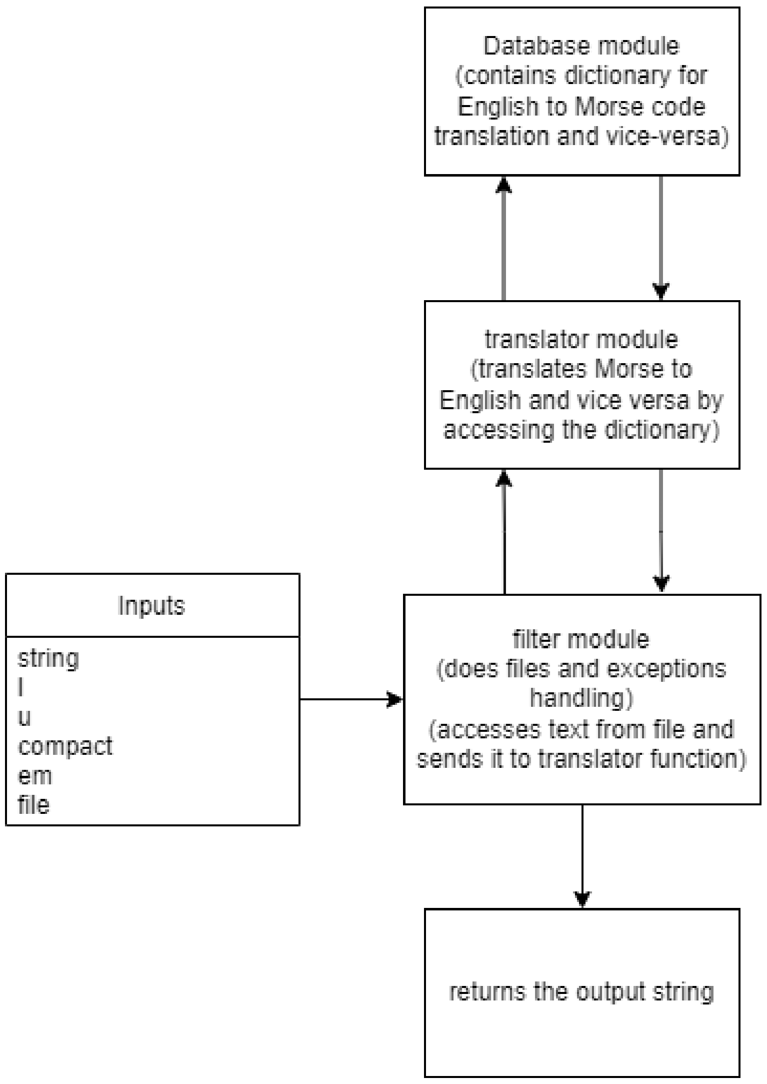
Output:

```
(Cant find the file)
```

Flow-chart for filter function


filter(string, l, u, compact, em, file)
read from file?
Yes
No
Exception Handling
file empty?
file not found?
no exception found?
Yes
Yes
Yes
Output: (empty file)
Output: (Cant find the file)
read the file and send the text to translator function
Call the translator function as it is
Return output string

Control flow from various functions and dictionaries


Database module (contains dictionary for English to Morse code translation and vice-versa)
translator module (translates Morse to English and vice versa by accessing the dictionary)
Inputs
string
l
u
compact
em
file
filter module (does files and exceptions handling) (accesses text from file and sends it to translator function)
returns the output string

Creating GUI

- **from tkinter import ***from tells the compiler to import something. Tkinter is a library in python and * means to import everything. The full line tells compiler to import everything from the library python.

- **root = Tk()**To initialize tkinter, we have to create a Tk root widget, which is a window with a title bar and other decoration provided by the window manager.

- **root. mainloop()**is simply a method in the main window that executes what we wish to execute in an application (lets Tkinter to start running the application). As the name implies it will loop forever until the user exits the window or waits for any events from the user.

- **root.geometry(“1000x650)** sets the default size of the GUI window to 1000 by 650.

- **root.minsize(900,600)** sets the minimimum size that can be given to the GUI window to 900 by 600.

- **root.wm_iconbitmap(“1.ico”)** used to change the default tkinter icon on the top left corner of the title bar to the icon given by the user.

- **root.title(“Morse code Translator”)** sets the title of the GUI window in the title bar to “Morse code Translator”.

- **var.set(1)**used to set the default value of a variable.

- **root.configure(bg="#FFBF86")** is used to access an object's attributes after its initialization.

Widgets Used

- **Frame** The Frame widget is used to process grouping and organizing other widgets. It works like a container, which is responsible for arranging the position of other widgets. It uses rectangular areas in the screen to organize the layout and to provide padding of these widgets. A frame can also be used as a foundation class to implement complex widgets.

- **Label** The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
- **Radiobutton**The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
- **Button**The Button widget is used to display buttons in your application.
- **Entry**The Entry widget is used to display a single-line text field for accepting values from a user.
- **Text**The Text widget is used to display text in multiple lines.
- **messagebox**This module is used to display message boxes in your applications.
- **Menu**The goal of this widget is to allow us to create all kinds of menus that can be used by our applications.

Geometry Management

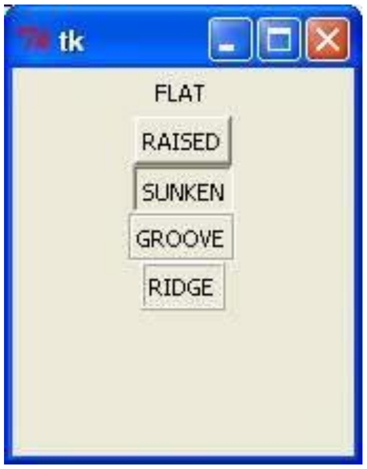
All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

- **The pack() Method**This geometry manager organizes widgets in blocks before placing them in the parent widget.
- **The grid() method**This geometry manager organizes widgets in a table-like structure in the parent widget.

Attributes used

- **bg**or **background** used to set the background color. It can hexadecimal value, octal value, rgb or some predefined color name.

- **relief**The relief style of a widget refers to certain simulated 3-D effects around the outside of the widget.The possible values which can be used for relief attribute FLAT, RAISED, SUNKEN, GROOVE, RIDGE.



- **Font** used to set the font size, style and formatting of the text. Ex:

  ("Helvetica", "16", “bold”, “underline”) for a 16-point Helvetica regular.

  (Times 24 bold italic) for a 24-point Times bold italic.

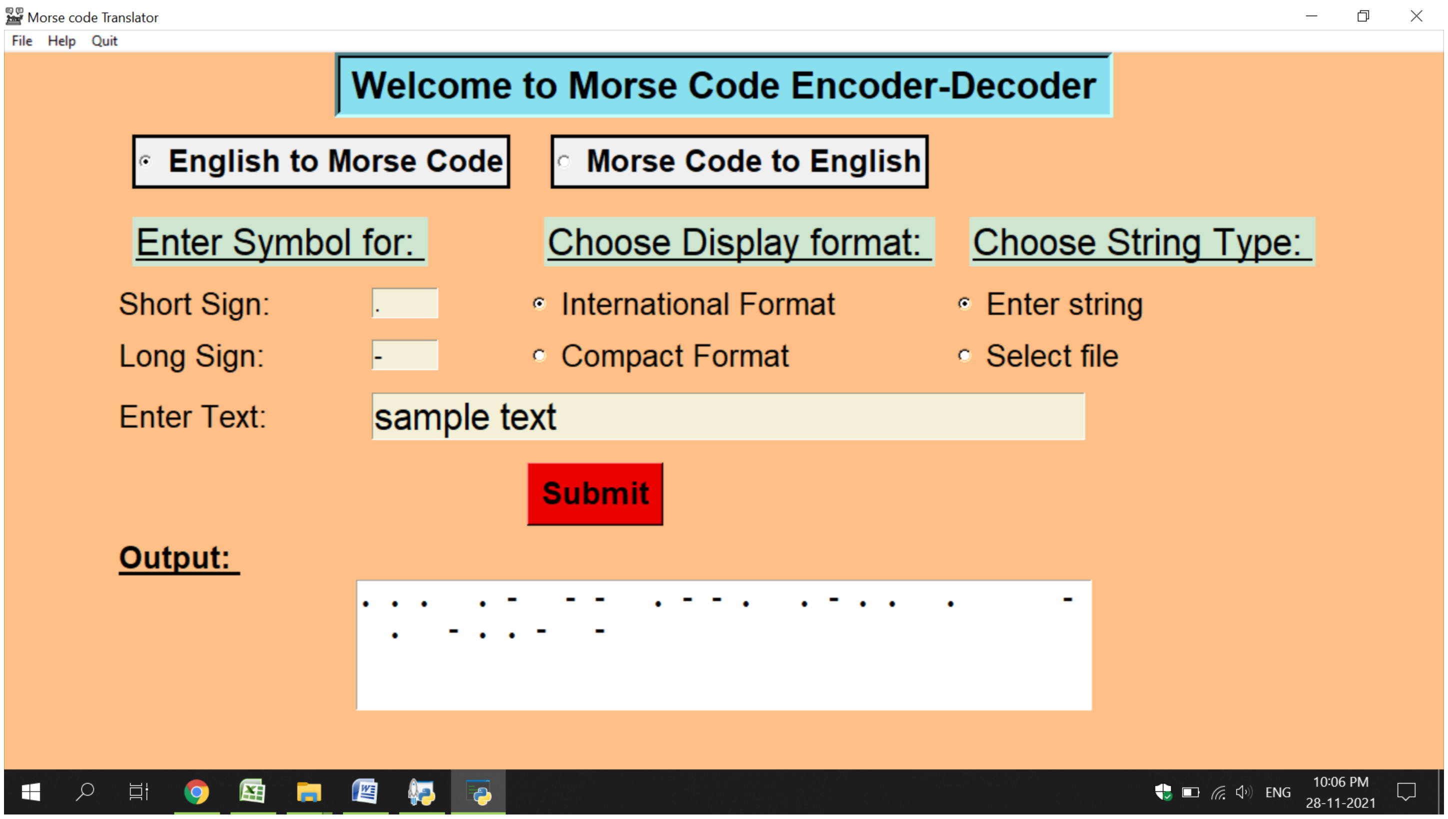
- **borderwidth**used to set the width of the border.
- **width**used to set the width of a widget in tkinter.
- **height**used to set the height of a widget in tkinter.
- **padx** used to set the padding on the x-axis.
- **pady** used to set the padding on the y-axis.
- **row** used to define the position/row of the widget. It is used in with the grid method.
- **column** used to define the position/column of the widget. It is used in with the grid method.
- **columnspan**defines how many columns the widget occupies. The default value is 1.
- **variable** used with the radiobutton, value in the variable is used to target the radiobutton. The value of the selected radiobutton gets stored in the variable

- **value** used to give value to a particular radio button. Each radio button is given a different value.
- **sticky** By default, with sticky='', widget is centered in its cell. sticky may be the string concatenation of zero or more of N, E, S, W, NE, NW, SE, and SW, compass directions indicating the sides and corners of the cell to which widget sticks.
- **textvariable** In order to be able to retrieve the current text from your entry widget, you must set this option to an instance of the StringVar class.
- **add_command**adds a menu item to the menu list.
- **add_separator()**adds a separator line to the menu.
- **add_cascade()**Created a new hierarchical menu by associating a given menu to a parent menu.
- **command**associates a function to a button/option/menu item.
- **tmsg.showinfo()**displays a message in a message box.

Variable Used

- **var1** used to store the conversion from english to morse code and vice-versa.
- **shortsign** used for the short sign in the morse code
- **longsign** used for the short sign in the morse code
- **var2** used to select the international or compact format of the morse code.
- **var3** used to enter the string mode or select file mode.
- **Input** used to store the sting variable entered into the enter text box.

Screenshots


Morse code Translator

File Help Quit

Welcome to Morse Code Encoder-Decoder

[✓ English to Morse Code] Morse Code to English

Enter Symbol for: Choose Display format: Choose String Type:

Short Sign: .

Long Sign: -

[✓ International Format] Compact Format

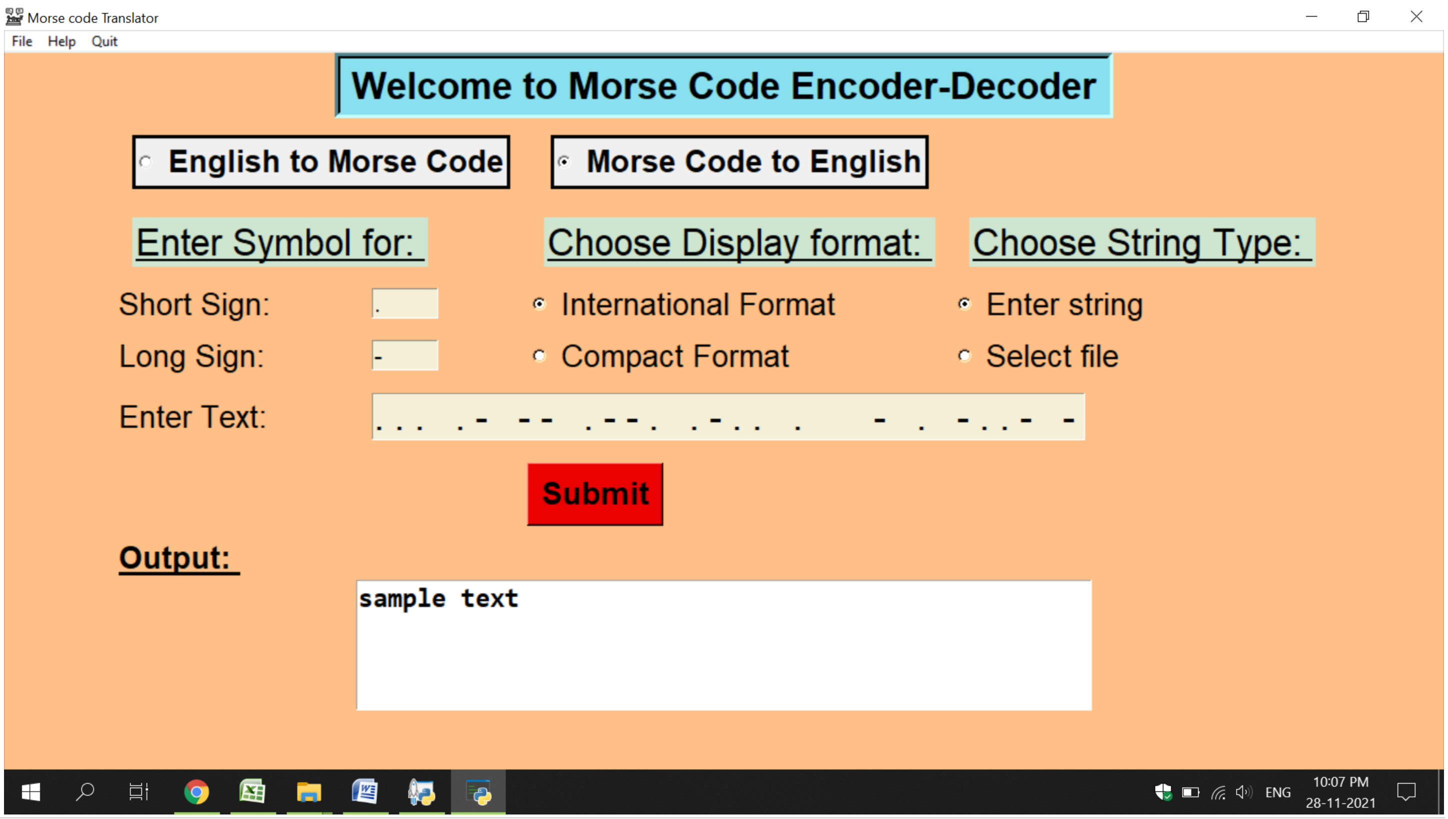
[✓ Enter string] Select file

Enter Text: sample text

Submit

Output:

```
. . .   . -   - -   . - - .   . - . .   .       -
  .     - . . -   -
```


Morse code Translator

File Help Quit

Welcome to Morse Code Encoder-Decoder

English to Morse Code [✓ Morse Code to English]

Enter Symbol for: Choose Display format: Choose String Type:

Short Sign: .

Long Sign: -

[✓ International Format] Compact Format

[✓ Enter string] Select file

Enter Text: `. . .  .-  --  .--.  .-..  .     -  .  -..-  -`

Submit

Output:

```
sample text
```

References

- https://app.diagrams.net/ (used for creating flow-charts)
- https://www.delftstack.com
- https://www.tutorialspoint.com/python/python_gui_programming.htm
- https://stackoverflow.com/
- https://www.geeksforgeeks.org/python-gui-tkinter/
- https://www.w3schools.in/python-tutorial/gui-programming/
- https://www.javatpoint.com/python-tkinter