# Morse code

## Introduction

A program that converts English text to Morse code and vice versa. For example, English text “SOS” should be converted to “•••−−−•••”. Converter is case insensitive, thus “sos” should produce same output “•••−−−•••” as “SOS” does. When converted to English, everything should be uppercased, thus “sos” converted to Morse and back to English should output “SOS”.
For this task, text (Either English or Morse) should be read from a file and output should be written to another. Simple command line interface for usage is enough which asks for input and output files and offers a way to do conversion from either English or Morse.

As requested, the simple code of command line version should get files from the user to convert it to morse code as encoding and get the English translation of it as decoding. The user will need to interact with the graphical user interface in the future, however, I wanted to make sure that the user can get best of interaction with the system, so I implemented a graphical user interface based on the command line version, but the text preprocessing was done in each dependently. All implementation is done in Python and testing as well.
To cover most of the requirements, I started in the API approach as well, defining the main form of the design and the skeleton files that needs time to be fully implemented and functional REST API that can be used on localhost or online if hosted.

In this documentation, I will explain the main architecture of the system, the command line version flow and the GUI version as well. I will go through the testing done, limitations and challenges. Also, the API design approach.

# Main architecture

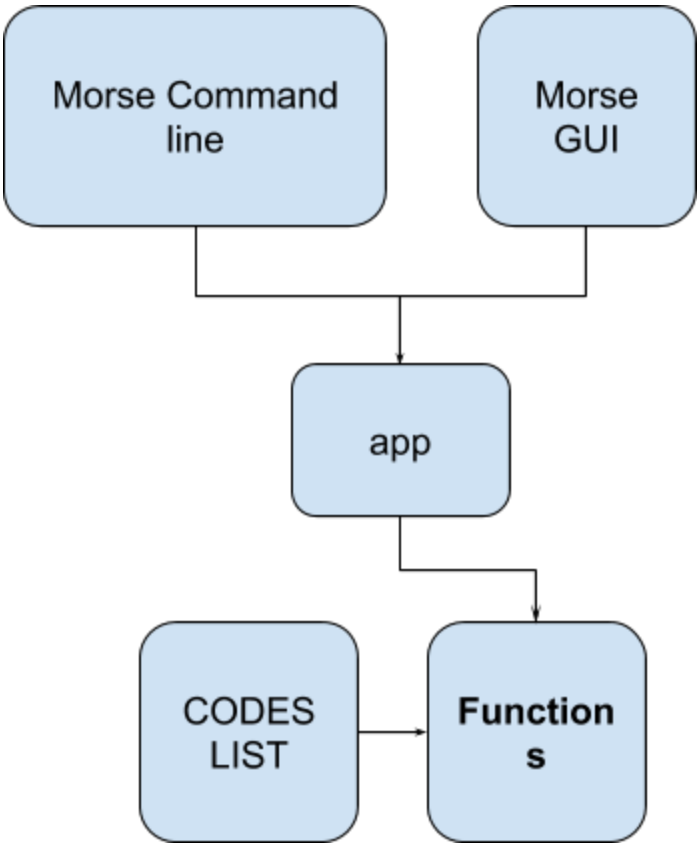


*Figure 1: Simple illustration on the main architecture of the core (simple) software of the system*

# Command line version

The command line version gives the user, the option to create their own file and enter the string they want to decode or encode, the user should enter the file name correctly with the extension, so it helps later in processing and reading the file. After creating a file or set of files, the user can then encode and decode by choosing the correct option among the options that are visible to him/her as shown:

```
Enter 1: encode to Morse, 2: decode to English, 3: create a file, 4: quit:
```

Then to implement fault tolerance, the user should enter a value in the specified options or else an error will show up asking the user to re-enter the correct value.

```
Enter 1: encode to Morse, 2: decode to English, 3: create a file, 4: quit:
5
Error: please enter a valid number
```

```
Enter 1: encode to Morse, 2: decode to English, 3: create a file, 4: quit:
```

And the same if the input is not a number or another value.

If the user starts in beginning they will choose 3 and create a new file, asking for the file name with the extension and the text inside that file where the user should enter.

```
Enter file name with extension: input.txt
Enter string to write to file: hello world
Output successfully written to files
```

Then the user can perform encoding or decoding where the system will pass through two layers of checking, first that the file name entered actually exists and it can be loaded, while the second layer is that the text inside the file can be processed and found in morse coding or decoding list. If the file name is wrong, an error message will be displayed asking for a new input, while if the processing in the text is wrong the system will display an error message asking the user to check the file and edit it.

For “hello world” as sample text:

```
Enter 1: encode to Morse, 2: decode to English, 3: create a file, 4: quit:
1
Enter the file name you want to decode to Morse: input.txt
.... . .-.. .-.. --- .-- --- .-. .-.. -..
Output successfully written to files
```

# Graphical user interface version

The graphical user interface follows the same approach as the command line in the flow, follow some screenshots of the graphical user interface while testing the system with the GUI together. I used Tkinter library for implementing the Graphical user interface, supporting fault tolerance as in the command line version.


*Figure 2: List of sample figures of GUI output*

# Functions and their testing

```
% ./run_tests.sh
+ python -m tests.morse_tests
Start running tests

('(test_get_english)', '\n        test decoding an english text to morse
code\n        ')
.--.....................-.-.------.-.-.-.--------..
.('(test_get_english)', '\n        test decoding an english text to morse
code\n        ')
.....---.---.---.---
.('(test_get_english)', '\n        test decoding an english text to morse
code\n        ')
-....-.....-..-.-.....-.....-.-------.--.--.---.----
.('(test_get_english_bad_request)', '\n        test when passing a bad
request to morse code decoder\n        ')
alkila
.('(test_get_english_wrong_format)', '\n        test when passing a wrong
format to the decoder\n        ')
.('(test_get_morse)', '\n        test sample input to make sure the
conversion to morse code (encoding) works properly\n        ')
What
is
even
your
name
?
.('(test_get_morse_bad_request)', '\n        test when passing a bad
request to morse code encoder\n        ')
.('(test_get_morse_with_numbers)', '\n        test sample input to make
sure the conversion to morse code (encoding) works properly\n        ')
12
continents
.('(test_get_morse_with_symbols)', '\n        test sample input to make
sure the conversion to morse code (encoding) works properly\n        ')
?
/
,
@
```

```
.('(test_get_morse_wrong_format)', '\n        test when passing a wrong
format to the encoder\n        ')
.('(test_write_file_empty_string)', '\n         test when passing an empty
string to create a file method\n        ')
.
----------------------------------------------------------------------
Ran 11 tests in 0.001s

OK
```

Please note that I had to hard code the tests to run and did not implement most of them due to shortage of time, however given some time the tests can be implemented through testing scripts. Also note that these exact test cases and more were tested manually and results were exact and provide robustness of the code.

### Suggested design for the API

# Restful API description

## General description

A simple program that converts English text to Morse code and vice versa is added as REST API. For example, English text “SOS” should be converted to “•••−−−•••”. Converter is case insensitive, thus “sos” should produce same output “•••−−−•••” as “SOS” does. When converted to English, everything should be uppercased, thus “sos” converted to Morse and back to English should output “SOS”. Text (Either English or Morse) should be read from a file and output should be written to another.

## Main concepts and relations

API permits the user to upload files in morse code or in English and gives him/her the option to create one and to encode or decode the text inside.

## Related work

There are many similar programs and APIs online, as an example:
https://morsecode.scphillips.com/translator.html

# RESTful API design

## Resources:

/morse/api/convert
/morse/api/convertMorse
morse/api/convertEnglish
/mores/api/upload

# RESTful client design

## RESTful client application description

The client can be an online platform (website) and a mobile application as well (Android)
The application will permit GET, PUT, POST and DELETE methods on the specified methods.