Morse code :: Worksheet(2 part 1)

> This worksheet is split into 2 parts. Part one is worth 50%.
>
> All work is handed in via git/GitLab and will require the skills demonstrated in this worksheet.

If you have not already done so, complete the setup process for you choice of OS, along with worksheet 1. Instructions for this are on Blackboard under Learning Materials.

For this worksheet you should create a new Git repo and all of your work should be pushed to there. Once you have completed the worksheet and pushed your work to Git you should submit the URL to BB under assignments, worksheet 2.

You repo should contain the following:

- Complete the 4 tasks specified in this worksheet.
- All the Python source code for the worksheet.
- A README.md describing what you have completed, how to run the code, and some examples of each part working.
  - This should be written in Markdown (hence the .md)
- Unit tests that test the functionality of your code.

Marking Scheme

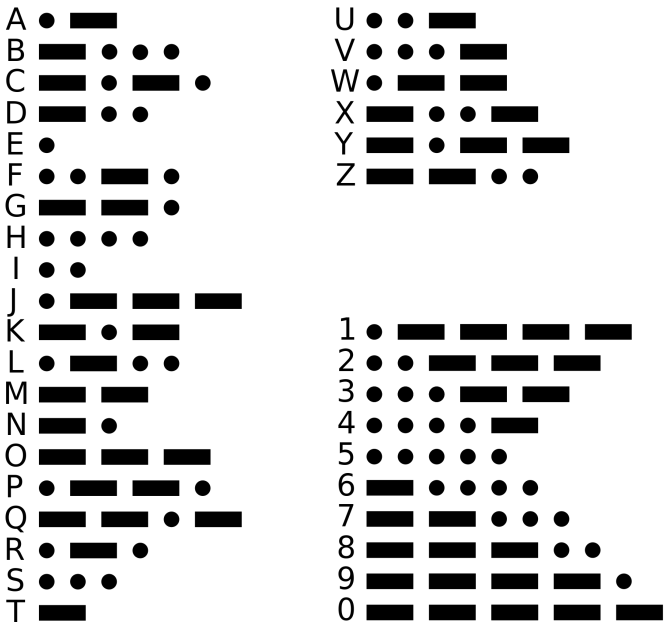
- Each task is worth 20% and will include the following metrics:
  - Correctness of solutions.
  - Comments and overall quality of code.
- Finally, the remaining 20% will include the quality of the READMME.md, the use of Git, and other unspecified additions.

Morse Code

Morse code is an approach to transmitting text information as a series of on-off-tones across a telegraph wire. It is an example of an encoding and a protocol, albeit a simple one. The encoding is shown on the below.

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.



Some example Morse messages are given below.

Original text:

A

Morse code:

. –

Notice that is we were to put lower case ‘a’, then it would also be encoded as:

. –

Original text:

```
Hello there
```

Morse code:

```
.... . .-.. .-.. --- / - .... . .-. .
```

> Note, it is strange to see a '/' symbol in Morse encodings, as there should only be a '.' and '-', this is there is represent the space between words. Space can't be used as this is already taken to seperate letters.

As noted on Wikipedia: "Morse code is usually transmitted by on-off keying of an information-carrying medium such as electric current, radio waves, visible light, or sound waves. The current or wave is present during the time period of the dot or dash and absent during the time between dots and dashes.

For this worksheet we will not focus on timing, however, we will return to in a later worksheet and so it is worth keeping in the back of your mind as you work through this one.

More information about Morse code can be found at https://en.wikipedia.org/wiki/Morse_code.

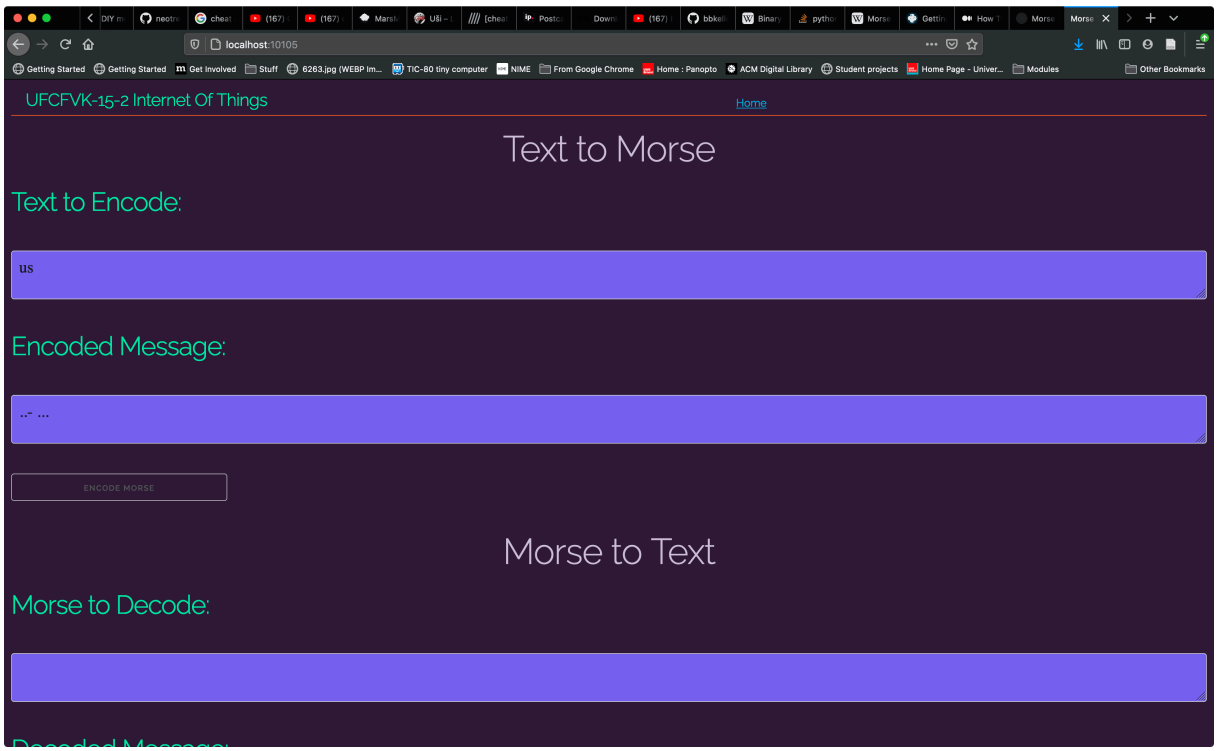
Tasks

Create a new Git repo for this worksheet.

Task 1

This first task is to simply get to know Morse code a little better. At least some of you are likely to have heard of Morse code before, maybe in a movie it might have sounded something: "beep, beep, beeeeep, beep". Or Hopper's secret message, so that Eleven knows it's him, in Stranger Things, which spells "us" in Morse:

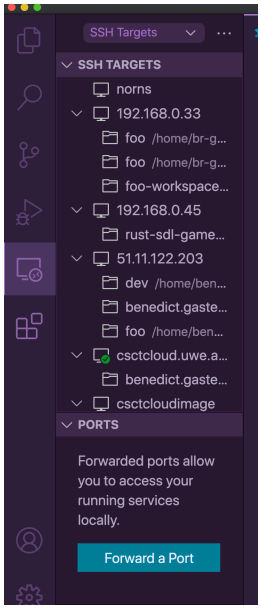
On the remote server there is web-server, on port 10105, that returns the following webpage that provides functionality to convert to and from Morse code:



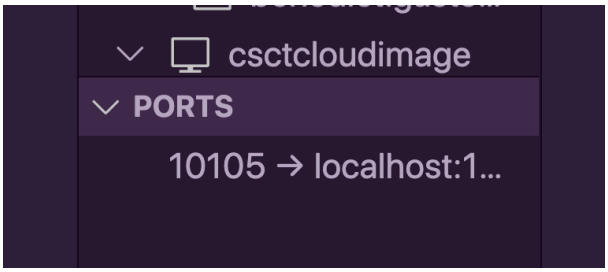
To connect to the server in a browser running on your local machine you need to forward the port (10105), using SSH, to your local machine. SSH port forwarding is a mechanism in SSH for tunneling application ports from the server machine to the client machine, or vice versa. This can be done direclty using the `ssh` command, from the command line, but easier still it can be acheived directly from VS code.

Before this process can be completed you need to connect to the remote server with VS code. Once connected click the “Remote Explorer” icon on the left:

which will open the a window that looks similar to the following:



This shows you remote connections and in the lower part a button to forward ports. Click this button and enter the port you want to forward, in this case **10105**, and press enter. It will set the port forwarding to your local machine (i.e. **localhost**):



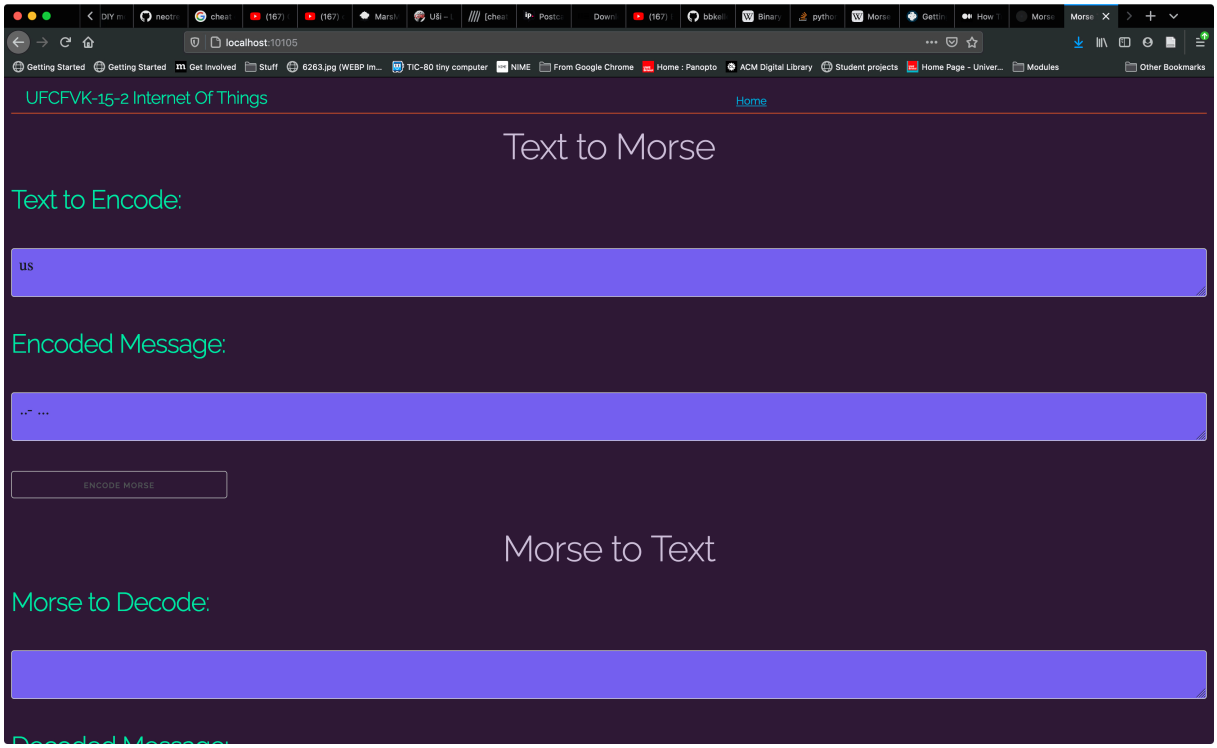
Now the cool thing is that you will be able to access the webserver running the on remote machine locally, even though the remote machine is not accesable to publically.

> Note, this works when you are creating your own server applications on the remote server, enabling you to test them locally, while benifiting from the compute power of a remote server and so on.

Open up a new tab in a browser and enter the url:

```
http://localhost:10105
```

You should now see the page:



Try entering Hopper's secret message, i.e. "us", what is the Morse encoding?

Try entering some of your own messages and decoding them too.

Task 2

The goal of this task is to implement two Python functions:

```
encode(msg: str) -> str
decode(msg: str) -> str
```

The first function takes a string of text and encodes it into a string of '.' and '-', as per the Morse code encoding given above. For example,

```
encode('us')
```

should return the string

```
'..- ..'
```

> Note, it maybe hard to see, but there is a space in between the `..-` and the `..`. This is important because without it, then it would be impossible to decode the `..-` as a 'U' and `..` as 'S'. Why is this the case?

while the function `decode` does the inverse and returns a string of text given a Morse message, e.g.:

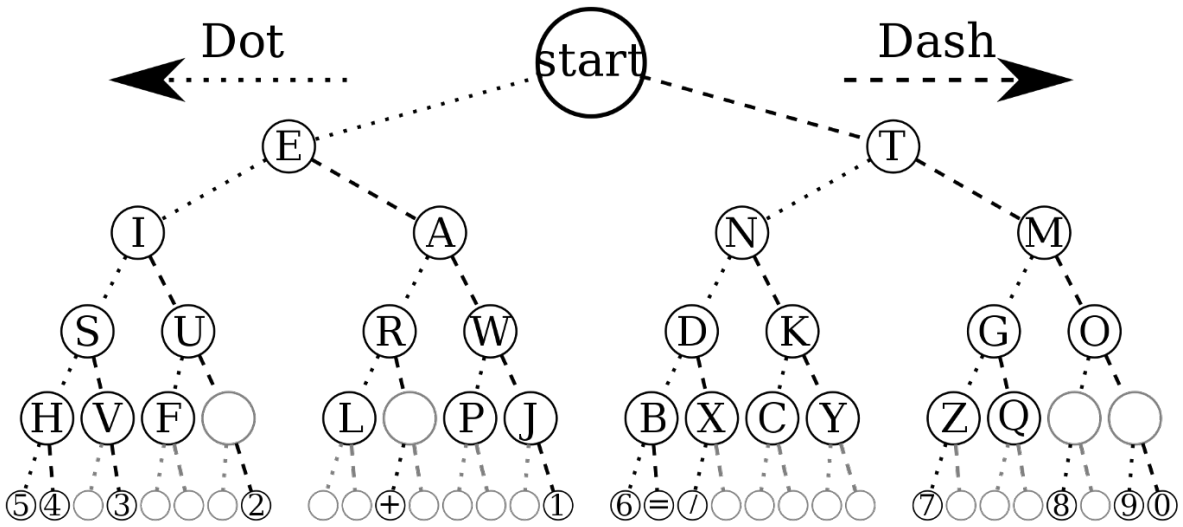
```
decode('..- ..')
```

should return the string

```
'us'
```

These functions should be implemented in a file called `morse.py`.

There are many ways to implement these functions, including a naive search, however, a key observation with Morse code is that it is a binary encoding, i.e. there are two possible values that can appear in a string that is the output of `encode` or the input to `decode`, '.' and '-'. If we think of the dots and dashes making up a path of the encoded message, with '.' meaning left and '-' meaning right, then a Morse encoding for a specfic letter can be described as a "walk" down a binary tree. This can be seen in the following:



The empty circles are called blanks, they will be important in the next worksheet, but for now are not important.

How do we use the tree with repsect to the decoding process. Let's consider the first letter in Hopper's message to Eleven, i.e. 'U', which is encoded as `..-`.

> Remember that in Morse there is no difference between upper and lower case.

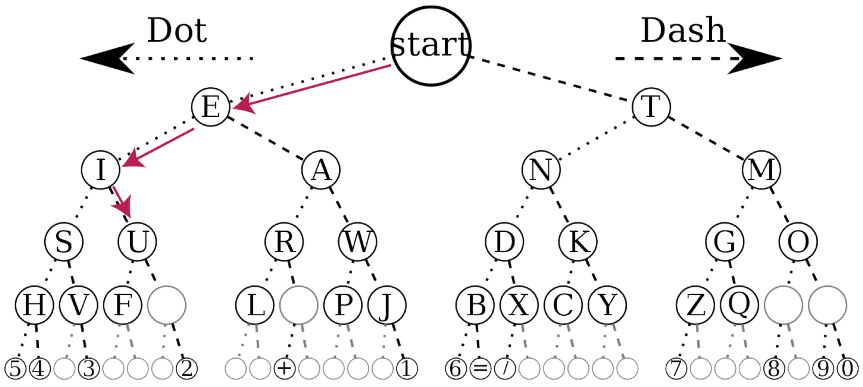
As noted above if we see a '.', then we go down the left branch of the tree, and so starting at the start:

- we first encounter a '.', so go left to 'E'
- we then encounter a '.', so go left again to 'I'
- we then encounter a '-', so go right this time to 'U'

and this point the algorithm does not no to stop and select 'U'. There are two possible states that would cause 'U' to be returned:

1. a character seperator is encountered, i.e. ' ' (space)
2. the end of input has been reached

In the case when the input message is just `..-`, then we will have rearched the end of the message once the two '.'s and the single'-' have been consumed. Diagramatically this looks like:



To implement **decode** you need to implement a Binary tree representation of the Morse encoding. From Wikipedia: ”In computer science, a binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

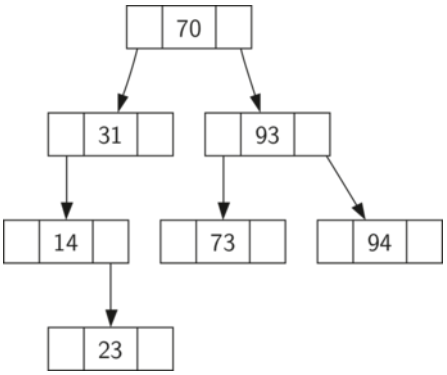
> Note, that while there are other approaches to implementing a Morse encoder and decoder to complete this task you are required to implement them using the Binary tree approach.

There are many excellent tutorials on the web for how to implement a binary tree in Python, in many ways easier than C as you don’t have to worry about allocation and deallocation of memory. A great example of such a tutorial and a good place to start can be found here.

Find yourself a good place to start and spend some time understanding what is going on and implement a basic Binary tree. It is important that you spend time understanding how it works, as you will need to extend it for working with Morse code.

In the linked tutorial it is important to note that the approach to inserting a value is designed for searching for a value, using lexicographical order, which works great for finding strings or numbers with low search complexity, however, it is different for the Morse encoding which uses ‘.’ and ‘-’ to move around the tree. To this end you will need to explictly build the Morse tree such that when going left (‘.’) and right (‘-’) from a particiular node it makes sense with respect to the encoding.

Write a function that prints a tree instance to the console. For example, if you were to take the tree from the top of the linked tutorial:



Your print function should produce something like:

```
r 70
  l 31
    l 14
      l -
      r - 23
    r -
  r - 93
    l - 73
    r - 94
```

Where **r** is the root or start node and **l** and **r** are the left and right children, respectively.

Once you have a working version you need to populate it to create the Morse tree, and then implement functions to perform **encode** and **decode** to and from Morse encodings.

> Hint, you will likely want to walk the tree one character of input at a time, which means for decode you will need to keep track of the current position in the tree.

The next task looks in more detail at testing your Morse encoder and decoder for now you should use the following `main`, which should be defined in **main.py**, to confirm that Eleven knows it's Hopper.

```
import morse

if __name__ == "__main__":
```

```
    e = morse.encode('us')
    print('%s' % e)
    d = morse.decode(e)

    assert morse.encode('us') == '..- ...', "Should be ..-"
    assert morse.decode('..- ...') == 'us', "Should be ..-"
```

Task 3

There are many ways to test your code, in this task you are going to explore Unit testing.

What is a unit test? A unit test is a small test, one that checks that a single bit of functionality operates in the right way. A unit test helps you to isolate what is broken in your code and fix it.

To write a unit test in Python you would check the output of `encode('us')` against a known output, which in this case is `..-`.

There are multiple places in the work you did in the previous task where unit tests would be useful, e.g.:

- Testing parts of the binary tree implemention
- Testing the functions **encode** and **decode**

In fact in its most basic form you already used a unit test in the last part of the previous task. There the function `assert` was used to check the result of calling `encode` and `decode` matched what was expected, given the respective input.

Of course, these were just called in main and not very extensible or easy to call from other places. Instead it makes more sense to lift them out to there own module, let's call it **assert_tests.py** and define the following:

```
import morse

def test_encode_us():
   assert morse.encode('us') == '..- ...', "Should be ..- ..."
```

You can also add a `main`, so if run directly it can run the test:

```
if __name__ == "__main__":
    assert_tests.test_encode_us(s)
    print('Everything passed')
```

run it and you should see the message:

```
Everything passed
```

> Note, if you see an error message, then something is likely wrong with your original code.

Try changing the line to:

```
assert morse.encode('us') == '..- ...', "Should be ..- ..."
```

and see what happens when you run it.

Change it back to the expect the correct answer and then add the function:

```
test_encode_us()
```

call it from **main**.

Writing tests like this is a quick and easy way to test your code as you develop it. Additionally, if someone else reports a bug in your code, they may (hopefully) provide a small test case, which can form the basis for a new unit test. This test will cause the bug to be exposed when the test is run, assuming it has not been fixed, and will pass once it has. This form of testing is often called regression testing, where tests are added to check that a fixed bug is not broken in a later release.

This form of testing is very popular and as you might expect there are many frameworks for helping with the development of unit testing. This is what test runners are for. A test runner is a special application designed for running tests, checking the output, and giving you tools for debugging and diagnosing tests and applications.

There are many test runners for Python. The one that will are going to use is built into the Python standard library and is called **unittest**.

To convert above example to a **unittest** test case, you have to:

- Import unittest from the standard library
- Create a class called TestSum that inherits from the TestCase class
- Convert the test functions into methods by adding self as the first argument
- Change the assertions to use the self.assertEqual() method on the TestCase class
- Change the command-line entry point to call unittest.main()

Create a file **morseunit.py** with the following:

```
import unittest
import morse

class TestMorse(unittest.TestCase):
    def test_encode_us():
        self.assertEqual( morse.encode('us'), '..- ...')

if __name__ == '__main__':
    unittest.main()
```

runing this should give the following:

```
python3 morseunit.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Now add a test for decode and check it works. Add a test that causes the particular test to fail.

To complete this task:

- add at least 5 tests for **encode** and five for **decode**, some of which are expect to fail.
- add tests for your binary tree implementation, which test at least:
  - that a tree is correctly empty
  - that a tree is not empty
  - the insert function
  - the delete function
  - find function

Task 4

So far your Morse alphabet supports letters from the English alphabet, which is Latin based alphabet. However, it does not provide any punctuation, e.g. an apostrophe('), which restricts the kind of messages that can be sent.

Below is a set of Morse encodings for puntuation and other useful for symbols.

Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code
.	.-.-.-	,	--..--	?	..--.	’	.----.	!	-.-.--
(	-.--.	)	-.--.-	&	.-...	:	---...	;	-.-.-.
+	.-.-.	-	-....-	_	..--.-	”	.-..-.	$	...-..-
¿	..-.-	¡	--...-						

Extend your implementation to support these additional symbols.

You should test your implementation, adding some new units tests to validate that they work correctly, as per thge previous task.