Python Radio 3: Text to Morse Code

Simon Quellen Field

Simon Quellen Field

Follow

5 min read

·

Aug 20, 2024

Listen

Share

More

Send text over the air with our little radio transmitter

Press enter or click to view image in full size

MidJourney

In our first project:

Python Radio: Simple Beginnings

The simplest digital radio mode

Medium.com

We had fun sending Morse code with the key by hand, but since we have a computer, why not let it translate the keys on the keyboard into Morse code?

We don’t need to change the hardware, except to get rid of the key.

We will first need a data structure to hold the Morse code table. A Python dictionary is just the thing:

Code = {

‘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’: ‘--..’,

‘0’: ‘-----’,

‘1’: ‘.----’,

‘2’: ‘..---‘,

‘3’: ‘...—‘,

‘4’: ‘....-‘,

‘5’: ‘.....’,

‘6’: ‘-....’,

‘7’: ‘--...’,

‘8’: ‘---..’,

‘9’: ‘----.’,

‘.’: ‘.-.-.-‘,

‘,’: ‘--..—‘,

‘?’: ‘..--..’,

‘\’’: ‘·−−−−·’,

‘!’: ‘−·−·−−‘,

‘/’: ‘−··−·’,

‘(‘: ‘−·−−·’,

‘)’: ‘−·−−·−‘,

‘&’: ‘·−···’,

‘:’: ‘−−−···’,

‘;’: ‘−·−·−·’,

‘=’: ‘−···−‘,

‘+’: ‘·−·−·’,

‘-‘: ‘−····−‘,

‘\_’: ‘··−−·−‘,

‘”’: ‘·−··−·’,

‘$’: ‘···−··−‘,

‘@’: ‘·−−·−·’,

}

We will put that into a file called the\_code.py.

Now we create a class to hold all the data and methods needed to make our Morse code transmitter function:

From machine import Pin, PWM

Class Morse:

Character\_speed = 18

Def \_\_init\_\_(self, pin):

Self.key = PWM(Pin(pin, Pin.OUT))

Self.key.freq(300)

Def speed(self, overall\_speed):

If overall\_speed >= 18:

Self.character\_speed = overall\_speed

Units\_per\_minute = int(self.character\_speed \* 50) # The word PARIS is 50 units of time

OVERHEAD = 2

Self.DOT = int(60000 / units\_per\_minute) – OVERHEAD

Self.DASH = 3 \* self.DOT

Self.CYPHER\_SPACE = self.DOT

If overall\_speed >= 18:

Self.LETTER\_SPACE = int(3 \* self.DOT) – self.CYPHER\_SPACE

Self.WORD\_SPACE = int(7 \* self.DOT) – self.CYPHER\_SPACE

Else:

# Farnsworth timing from <https://www.arrl.org/files/file/Technology/x9004008.pdf>

Farnsworth\_spacing = (60000 \* self.character\_speed – 37200 \* overall\_speed) / (overall\_speed \* self.character\_speed)

Farnsworth\_spacing \*= 60000/68500 # A fudge factor to get the ESP8266 timing closer to correct

Self.LETTER\_SPACE = int((3 \* farnsworth\_spacing) / 19) – self.CYPHER\_SPACE

Self.WORD\_SPACE = int((7 \* farnsworth\_spacing) / 19) – self.CYPHER\_SPACE

Def send(self, str):

From the\_code import code

From time import sleep\_ms

For c in str:

If c == ‘ ‘:

Self.key.duty(0)

Sleep\_ms(self.WORD\_SPACE)

Else:

Cyphers = code[c.upper()]

For x in cyphers:

If x == ‘.’:

Self.key.duty(512)

Sleep\_ms(self.DOT)

Else:

Self.key.duty(512)

Sleep\_ms(self.DASH)

Self.key.duty(0)

Sleep\_ms(self.CYPHER\_SPACE)

Self.key.duty(0)

Sleep\_ms(self.LETTER\_SPACE)

Let’s look at the send() method first. It accepts the string of characters we wish to transmit. For each character in the string, we look it up in the code table and get the string of dots and dashes for that letter.

Then we walk through the string of dots and dashes, turning on the PWM pin for the length of a dot or a dash.

After each dot or dash, we turn off the PWM pin and wait for a time equal to a dot.

When we are done with the string of dots and dashes, we wait for the proper time to indicate that we are done with the letter.

Half of the code in the class worries about setting the proper delays for the right speed. This is because humans find it easiest to hear Morse characters by the rhythm of the dots and dashes, and that is actually easier when the dots and dashes are happening quickly. For those who have not learned Morse at higher speeds, we add a delay between the characters to give the person time to recall what letter that sound was.

This is called the Farnsworth method for learning Morse code. Letters are always sent at least 18 words per minute, even if the overall speed is only something like 5 words per minute or less.

We put the Morse class into a file called morse.py.

The main routine is fairly simple:

From morse import Morse

Def main():

PIN\_D4 = 2

Morse = Morse(PIN\_D4)

Print(”Morse code transmitter”)

While(True):

Wpm = input(”How many words per minute? “)

If int(wpm) > 0 and int(wpm) < 50:

Morse.speed(int(wpm))

Str = input(”Enter the message to send: “)

Morse.send(str)

Else:

Print(”Try a more reasonable speed.”)

Main()

We put it into main.py

When all the files are loaded onto the ESP8266 (using the send\_files.cmd script), we are ready to run it.

Since it asks for input, we will need to access the Python REPL (Read, Evaluate, Print Loop) using a terminal emulator. I use the Putty program on Windows, but Linux and Macs have their own terminal emulators.

The terminal emulator connects to the USB port that powers the ESP8266:

Putty -load com3

At first, the putty screen is blank. If you hit the Enter key, you get a prompt from the REPL of three greater-than signs. To do a soft reboot (which then runs the program) type a control-D.

Here is what the screen looks like after entering a speed and a message:

>>>

MPY: soft reboot

WebREPL server started on <http://192.168.4.1:8266/>

Started webrepl in normal mode

Morse code transmitter

How many words per minute? 20

Enter the message to send: Hello

How many words per minute?

Note that it says it has started the WebREPL. This is because of the web\_repl.py file we loaded onto the ESP8266. Because of this, you can connect to our little computer over Wi-Fi. Look for a Wi-Fi SSID that looks like MicroPython-xxxxxx where the x characters are hexadecimal characters. Connect to that Wi-Fi SSID, and you will be asked for a password. The web\_repl.py file has the password in it. Enter the password, and now your web browser can control the ESP8266 in the same way the terminal emulator does. So, you can talk to it even if it is battery-powered up on a pole or on the roof.

As before, you can hear the Morse code coming out of the speaker on the receiver we built in the previous Morse code project. This is one way to learn Morse code. You can send whole files and the little computer will send the Morse to anyone listening on their receivers. A whole class or a whole neighborhood can listen in on their receivers, which they can build for about a dollar.

The files for this project can be found here: <https://scitoys.com/pythonradio/morse_keyboard.zip>.

Radio Transmitter

Python Programming

Morse Code