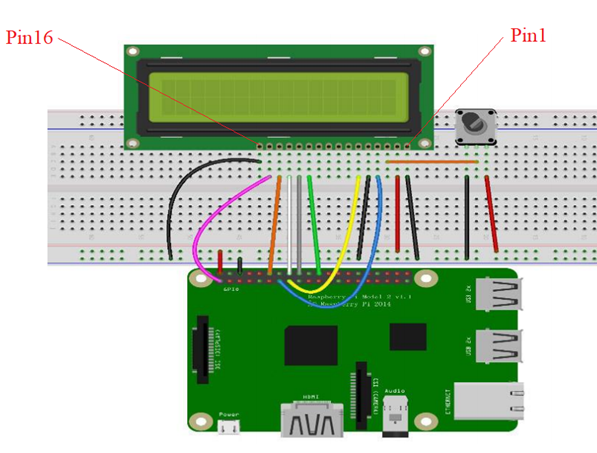
**Fall LCD Display**

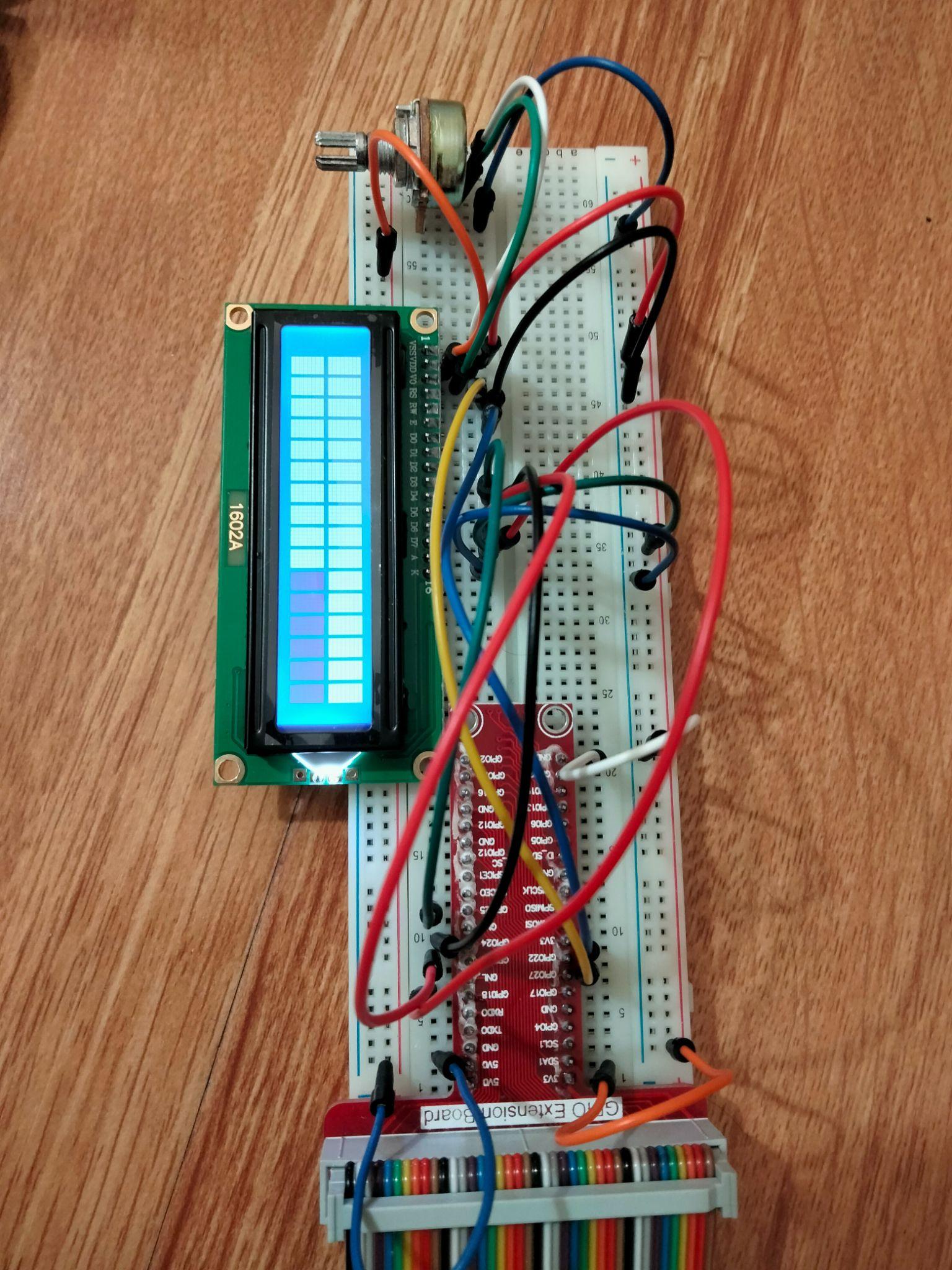
**Output:** <https://photos.app.goo.gl/cCf5DaJx2svQ7ebT6>

**Components:**

* Raspberry Pi
* Breadboard
* LCD 1602
* Potentiometer
* Jumper wires
* 40-Pin GPIO Cable
* T-Extension Board

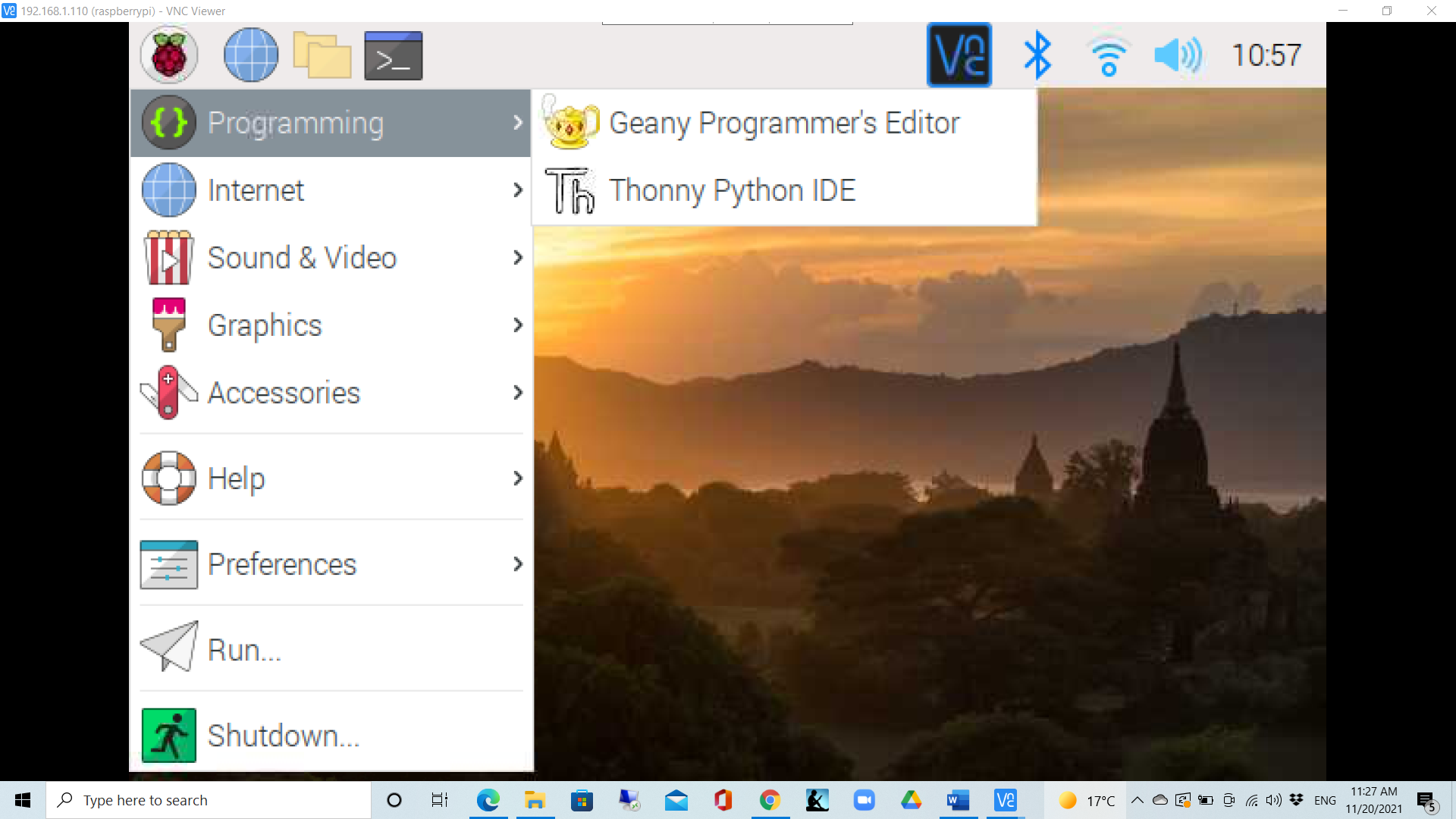
**Connections**:





**Working:**

1. Once the wireless connection is established then open thonny Python IDE and open a new file.



1. Type in the program and save as LCD.py

**# Python Program**

from time import sleep

class LCD:

LCD\_CLEARDISPLAY = 0x01 # 0000\_0001

LCD\_RETURNHOME = 0x02

LCD\_ENTRYMODESET = 0x04

LCD\_DISPLAYCONTROL = 0x08

LCD\_CURSORSHIFT = 0x10

LCD\_FUNCTIONSET = 0x20

LCD\_SETCGRAMADDR = 0x40

LCD\_SETDDRAMADDR = 0x80

# flags for display entry mode

LCD\_ENTRYRIGHT = 0x00

LCD\_ENTRYLEFT = 0x02

LCD\_ENTRYSHIFTINCREMENT = 0x01

LCD\_ENTRYSHIFTDECREMENT = 0x00

# flags for display on/off control

LCD\_DISPLAYON = 0x04

LCD\_DISPLAYOFF = 0x00

LCD\_CURSORON = 0x02

LCD\_CURSOROFF = 0x00

LCD\_BLINKON = 0x01

LCD\_BLINKOFF = 0x00

# flags for display/cursor shift

LCD\_DISPLAYMOVE = 0x08

LCD\_CURSORMOVE = 0x00

# flags for display/cursor shift

LCD\_DISPLAYMOVE = 0x08

LCD\_CURSORMOVE = 0x00

LCD\_MOVERIGHT = 0x04

LCD\_MOVELEFT = 0x00

# flags for function set

LCD\_8BITMODE = 0x10

LCD\_4BITMODE = 0x00

LCD\_2LINE = 0x08

LCD\_1LINE = 0x00

LCD\_5x10DOTS = 0x04

LCD\_5x8DOTS = 0x00

def \_\_init\_\_(self, pin\_rs=27, pin\_e=22, pins\_db=[25, 24, 23, 18], GPIO = None):

# Emulate the old behavior of using RPi.GPIO if we haven't been given

# an explicit GPIO interface to use if not GPIO:

import RPi.GPIO as GPIO

self.GPIO = GPIO

self.pin\_rs = pin\_rs

self.pin\_e = pin\_e

self.pins\_db = pins\_db

self.used\_gpio = self.pins\_db[:]

self.used\_gpio.append(pin\_e)

self.used\_gpio.append(pin\_rs)

self.GPIO.setwarnings(False)

self.GPIO.setmode(GPIO.BCM)

self.GPIO.setup(self.pin\_e, GPIO.OUT)

self.GPIO.setup(self.pin\_rs, GPIO.OUT)

for pin in self.pins\_db:

self.GPIO.setup(pin, GPIO.OUT)

self.write4bits(0x33) # initialization

self.write4bits(0x32) # initialization

self.write4bits(0x28) # 2 line 5x7 matrix

self.write4bits(0x0C) # turn cursor off 0x0E to enable cursor

self.write4bits(0x06) # shift cursor right

self.displaycontrol = self.LCD\_DISPLAYON | self.LCD\_CURSOROFF | self.LCD\_BLINKOFF

self.displayfunction = self.LCD\_4BITMODE | self.LCD\_1LINE | self.LCD\_5x8DOTS

self.displayfunction |= self.LCD\_2LINE

""" Initialize to default text direction (for romance languages) """

self.displaymode = self.LCD\_ENTRYLEFT | self.LCD\_ENTRYSHIFTDECREMENT

self.write4bits(self.LCD\_ENTRYMODESET | self.displaymode) # set the entry mode

self.clear()

def begin(self, cols, lines):

if (lines > 1):

self.numlines = lines

self.displayfunction |= self.LCD\_2LINE

self.currline = 0

def home(self):

self.write4bits(self.LCD\_RETURNHOME) # set cursor position to zero

self.delayMicroseconds(3000) # this command takes a long time!

def clear(self):

self.write4bits(self.LCD\_CLEARDISPLAY) # command to clear display

self.delayMicroseconds(3000) # 3000 microsecond sleep, clearing the display takes a long time

def setCursor(self, col, row):

self.row\_offsets = [ 0x00, 0x40, 0x14, 0x54 ]

if ( row > self.numlines ):

row = self.numlines - 1 # we count rows starting w/0

self.write4bits(self.LCD\_SETDDRAMADDR | (col + self.row\_offsets[row]))

def noDisplay(self):

# Turn the display off (quickly)

self.displaycontrol &= ~self.LCD\_DISPLAYON

self.write4bits(self.LCD\_DISPLAYCONTROL | self.displaycontrol)

def display(self):

# Turn the display on (quickly)

self.displaycontrol |= self.LCD\_DISPLAYON

self.write4bits(self.LCD\_DISPLAYCONTROL | self.displaycontrol)

def noCursor(self):

# Turns the underline cursor on/off

self.displaycontrol &= ~self.LCD\_CURSORON

self.write4bits(self.LCD\_DISPLAYCONTROL | self.displaycontrol)

def cursor(self):

# Cursor On

self.displaycontrol |= self.LCD\_CURSORON

self.write4bits(self.LCD\_DISPLAYCONTROL | self.displaycontrol)

def noBlink(self):

# Turn on and off the blinking cursor

self.displaycontrol &= ~self.LCD\_BLINKON

self.write4bits(self.LCD\_DISPLAYCONTROL | self.displaycontrol)

def noBlink(self):

# Turn on and off the blinking cursor

self.displaycontrol &= ~self.LCD\_BLINKON

self.write4bits(self.LCD\_DISPLAYCONTROL | self.displaycontrol)

def DisplayLeft(self):

# These commands scroll the display without changing the RAM

self.write4bits(self.LCD\_CURSORSHIFT | self.LCD\_DISPLAYMOVE | self.LCD\_MOVELEFT)

def scrollDisplayRight(self):

# These commands scroll the display without changing the RAM

self.write4bits(self.LCD\_CURSORSHIFT | self.LCD\_DISPLAYMOVE | self.LCD\_MOVERIGHT);

def leftToRight(self):

# This is for text that flows Left to Right

self.displaymode |= self.LCD\_ENTRYLEFT

self.write4bits(self.LCD\_ENTRYMODESET | self.displaymode);

def rightToLeft(self):

# This is for text that flows Right to Left

self.displaymode &= ~self.LCD\_ENTRYLEFT

self.write4bits(self.LCD\_ENTRYMODESET | self.displaymode)

def autoscroll(self):

# This will 'right justify' text from the cursor

self.displaymode |= self.LCD\_ENTRYSHIFTINCREMENT

self.write4bits(self.LCD\_ENTRYMODESET | self.displaymode)

def noAutoscroll(self):

# This will 'left justify' text from the cursor

self.displaymode &= ~self.LCD\_ENTRYSHIFTINCREMENT

self.write4bits(self.LCD\_ENTRYMODESET | self.displaymode)

def write4bits(self, bits, char\_mode=False): # one of functions in class LCD

# Send command to LCD

self.delayMicroseconds(1000) # 1000 microsecond sleep

bits=bin(bits)[2:].zfill(8)

self.GPIO.output(self.pin\_rs, char\_mode)

for pin in self.pins\_db:

self.GPIO.output(pin, False)

for i in range(4):

if bits[i] == "1":

self.GPIO.output(self.pins\_db[::-1][i], True)

self.pulseEnable()

for pin in self.pins\_db:

self.GPIO.output(pin, False)

for i in range(4,8):

if bits[i] == "1":

self.GPIO.output(self.pins\_db[::-1][i-4], True)

self.pulseEnable()

def delayMicroseconds(self, microseconds):

seconds = microseconds / float(1000000) # divide microseconds by 1 million for seconds

sleep(seconds)

def pulseEnable(self):

self.GPIO.output(self.pin\_e, False)

self.delayMicroseconds(1) # 1 microsecond pause - enable pulse must be > 450ns

self.GPIO.output(self.pin\_e, True)

self.delayMicroseconds(1) # 1 microsecond pause - enable pulse must be > 450ns

self.GPIO.output(self.pin\_e, False)

self.delayMicroseconds(1) # commands need > 37us to settle

def message(self, text):

# Send string to LCD. Newline wraps to second line

print ("message: %s",text)

for char in text:

if char == '\n':

self.write4bits(0xC0) # next line

else:

self.write4bits(ord(char),True)

def destroy(self):

print ("clean up used\_gpio")

self.GPIO.cleanup(self.used\_gpio)

def print\_msg():

print ("========================================")

print ("| LCD1602 |")

print ("| ------------------------------ |")

print ("| D4 connect to BCM25 |")

print ("| D5 connect to BCM24 |")

print ("| D6 connect to BCM23 |")

print ("| D7 connect to BCM18 |")

print ("| RS connect to BCM27 |")

print ("| CE connect to bcm22 |")

print ("| RW connect to GND |")

print ("| |")

print ("| Control LCD1602 |")

print ("| |")

print ("| SunFounder|")

print ("========================================\n")

print ('Program is running...')

print ('Please press Ctrl+C to end the program...')

raw\_input ("Press Enter to begin\n")

def main():

global lcd

print\_msg()

lcd = LCD() # obj: lcd

line0 = " sunfounder.com"

line1 = "---SUNFOUNDER---"

lcd.clear()

lcd.message("Welcome to --->\n sunfounder.com")

sleep(3)

msg = "%s\n%s" % (line0, line1)

while True:

lcd.begin(0, 2)

lcd.clear()

for i in range(0, len(line0)):

lcd.setCursor(i, 0)

lcd.message(line0[i])

sleep(0.1)

for i in range(0, len(line1)):

lcd.setCursor(i, 1)

lcd.message(line1[i])

sleep(0.1)

sleep(1)

if \_\_name\_\_ == '\_\_main\_\_':

try:

main()

except KeyboardInterrupt:

lcd.clear()

lcd.destroy()

1. Run the program in command prompt as:

sudo python LCD.py

1. Now you can observe the characters displayed on the LCD.