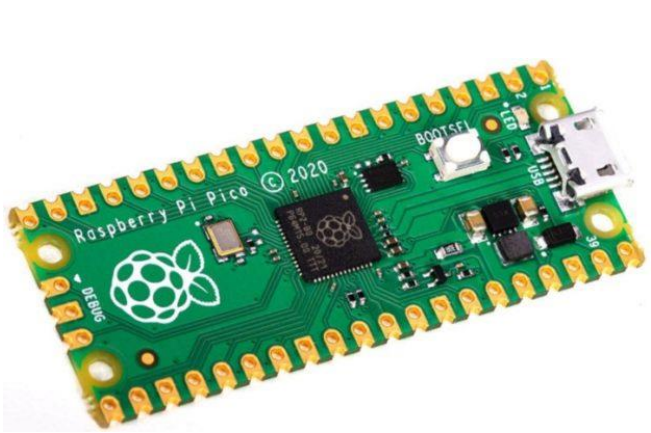
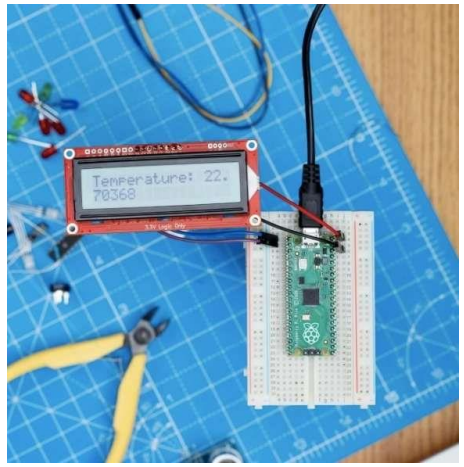
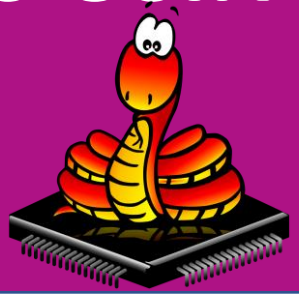


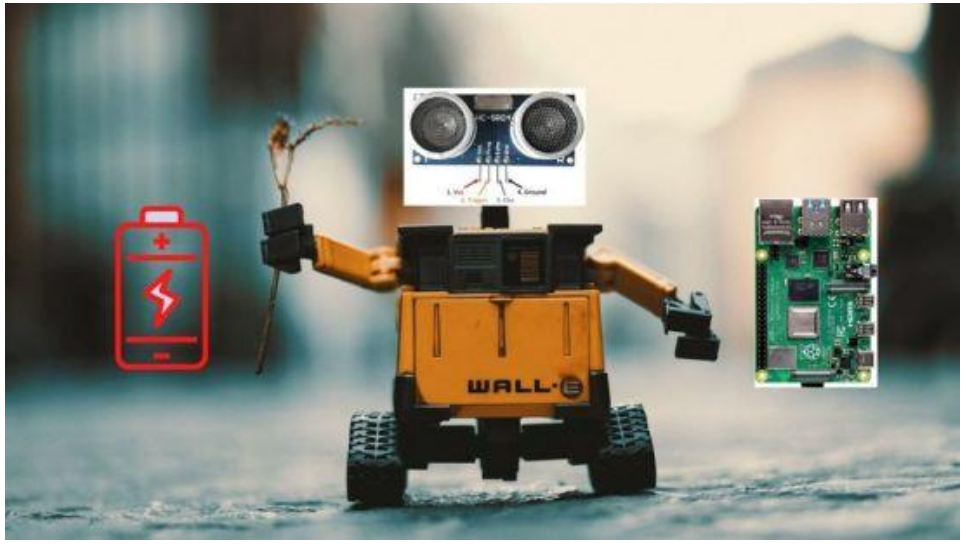


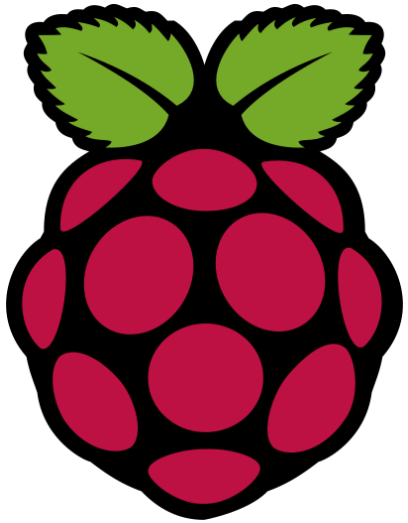
INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

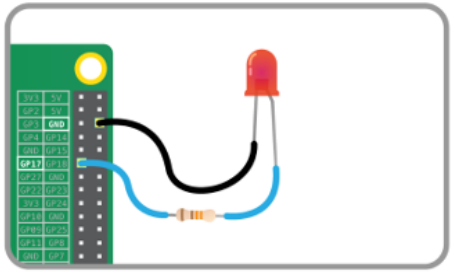
(CSE 4110)

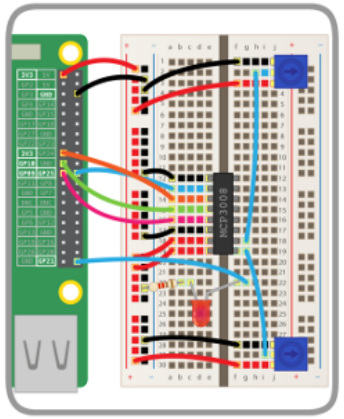
(LECTURE – 7)











Programming an OLED Screen on Raspberry Pi Pico

The SSD1306 OLED display is available in both I2C & SPI Modules. But we will use the 0.96″ I2C OLED Display as it requires only 2 wires for Interfacing. The Raspberry Pi Pico has two Pairs of I2C Pins. We can use any of the I2C Pins of Raspberry Pi Pico for Interfacing SSD1306 OLED Display.

The MicroPython IDE requires the SSD1306 Driver Code. After writing the driver code, we can write anything and display it on OLED Display. We will display the Analog value voltage from the Potentiometer on OLED Display.

SSD1306 OLED Display

- **0.96/1.3 inch blue OLED display module**
- **SPI/IIC protocols**
- **resolution of 128x64.**

Pin 1: GND

Pin 2: 3.3V to 5V

Pin 3: SCL - Serial Clock

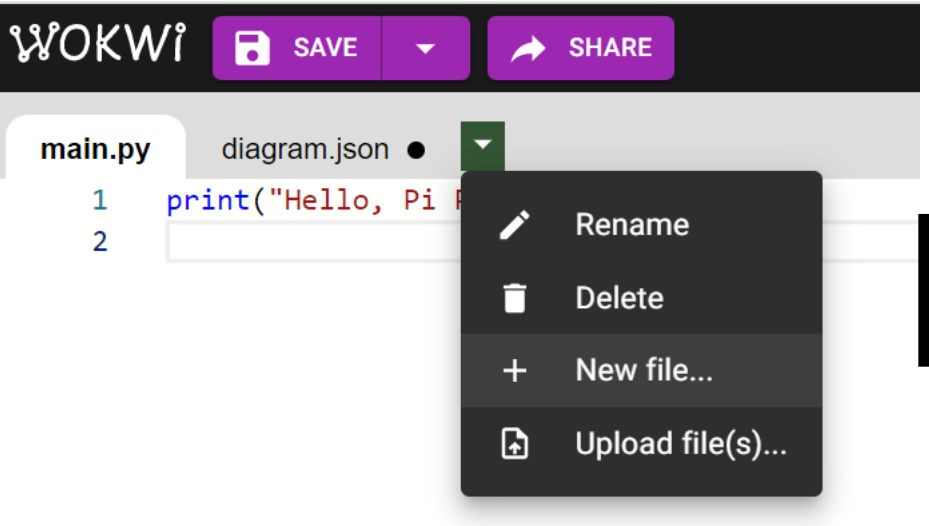
Pin 4: SDA - Serial Data

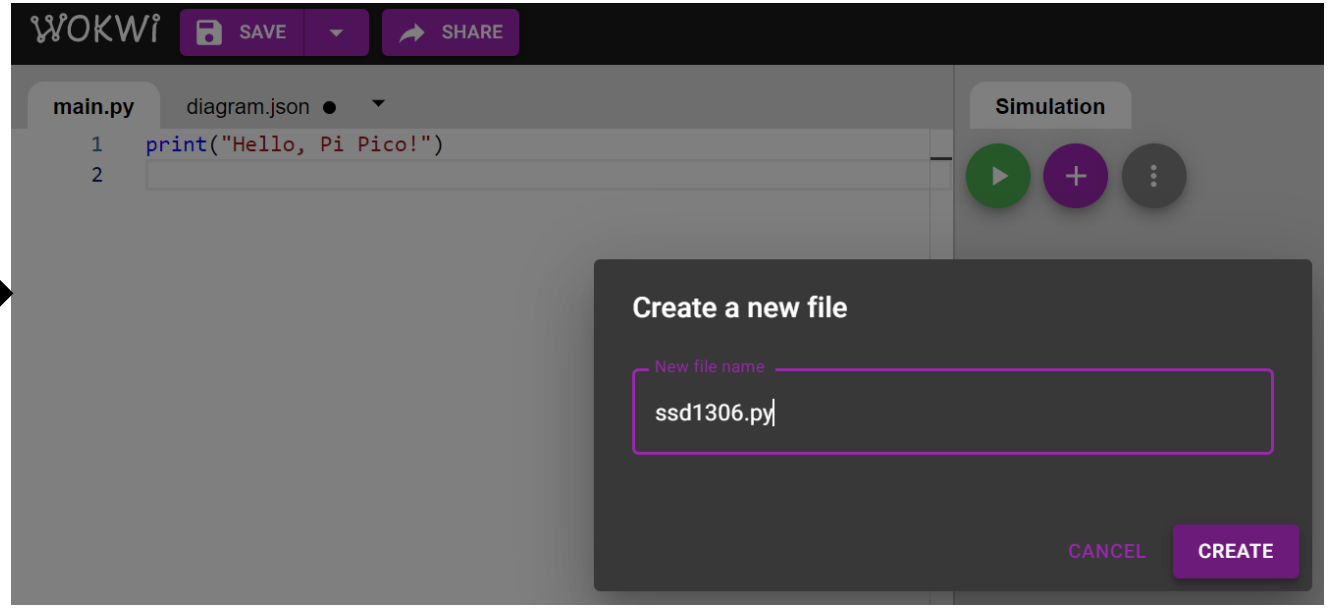
OLED vs LCD (IPS) – Which is better and Why?

Pros of IPS LCD	Cons of IPS LCD
Relatively cheap and easy to manufacture	Limited contrast
Good colour accuracy	Possible backlight 'leakage'
Doesn't suffer from image burn-in	
Pros of OLED	**Cons of OLED**
Thinner than IPS LCD	Possibility of image burn-in
Very power efficient	Expensive to manufacture
Excellent viewing angles	
Excellent black levels	
Excellent colour gamut	

Furthermore, over OLED displays, IPS LCD has advantages. The longer life and reliability of the devices equipped with that particular kind of LCD technology are major advantages because they translate into stability and stronger usability.

Programming an OLED Screen on Raspberry Pi Pico





Create a new file ssd1306.py

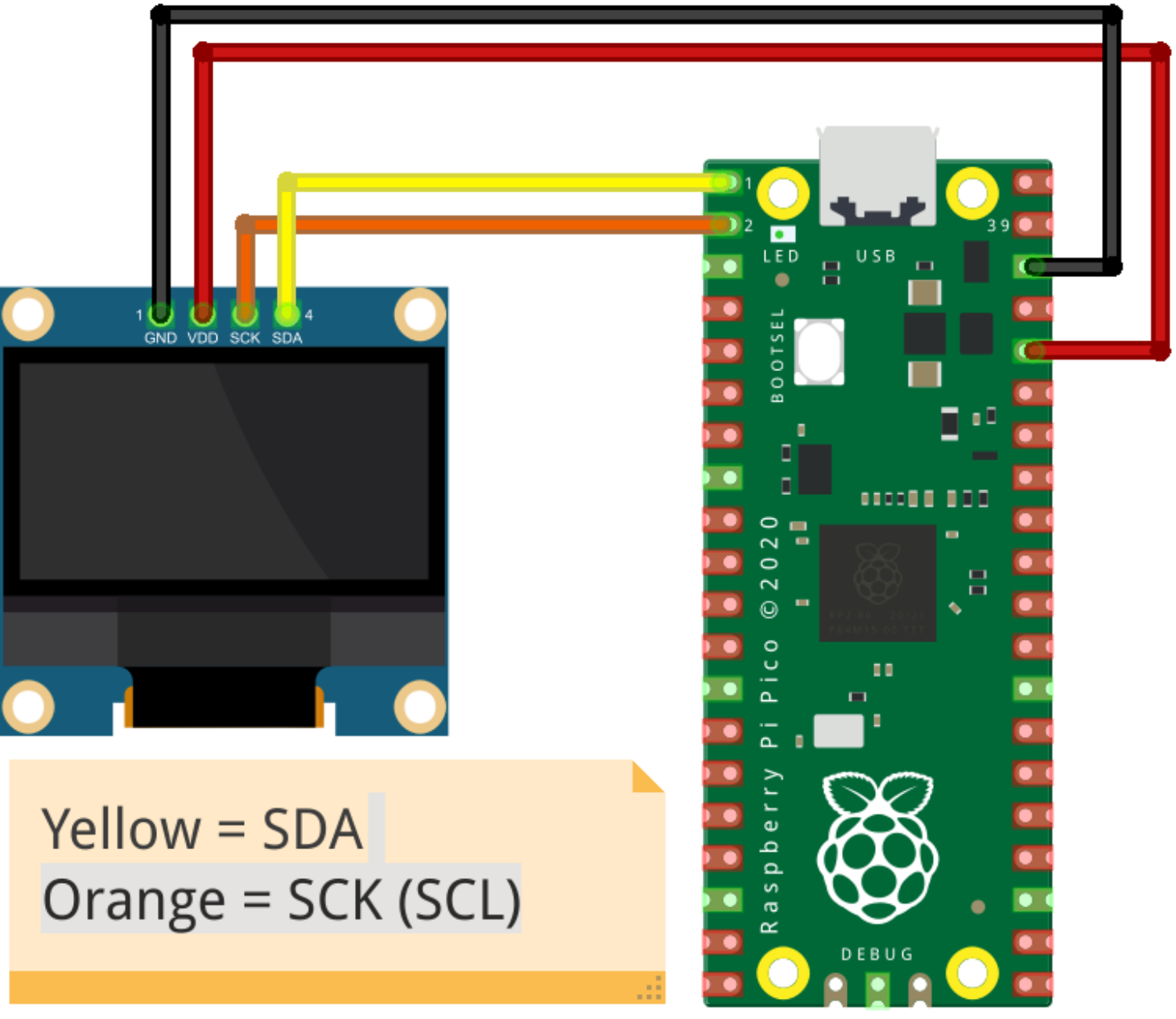
```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)

oled.text(" Welcome to ",8, 0)
oled.text("IOT Projects",8, 16)
oled.text("using PYTHON",8, 32)
oled.text("(CSE - 4110)",8, 48)
oled.show()
```

main.py

Create ssd1306 library :

https://wokwi.com/projects/350496872294515284

How to Connect an OLED screen to Raspberry Pi Pico



1. Connect the GND of the screen to any GND on the Pico (Black wire).

2. Connect VDD / VCC to 3V3 on the Pico (Red wire).

3. Connect SCK / SCL to I2C0 SCL (GP1, Physical pin 2, Orange wire).

4. Connect SDA to I2C0 SDA (GP0, Physical pin 1, Yellow wire).

5. Connect your Raspberry Pi Pico to your computer and open the Thonny application.

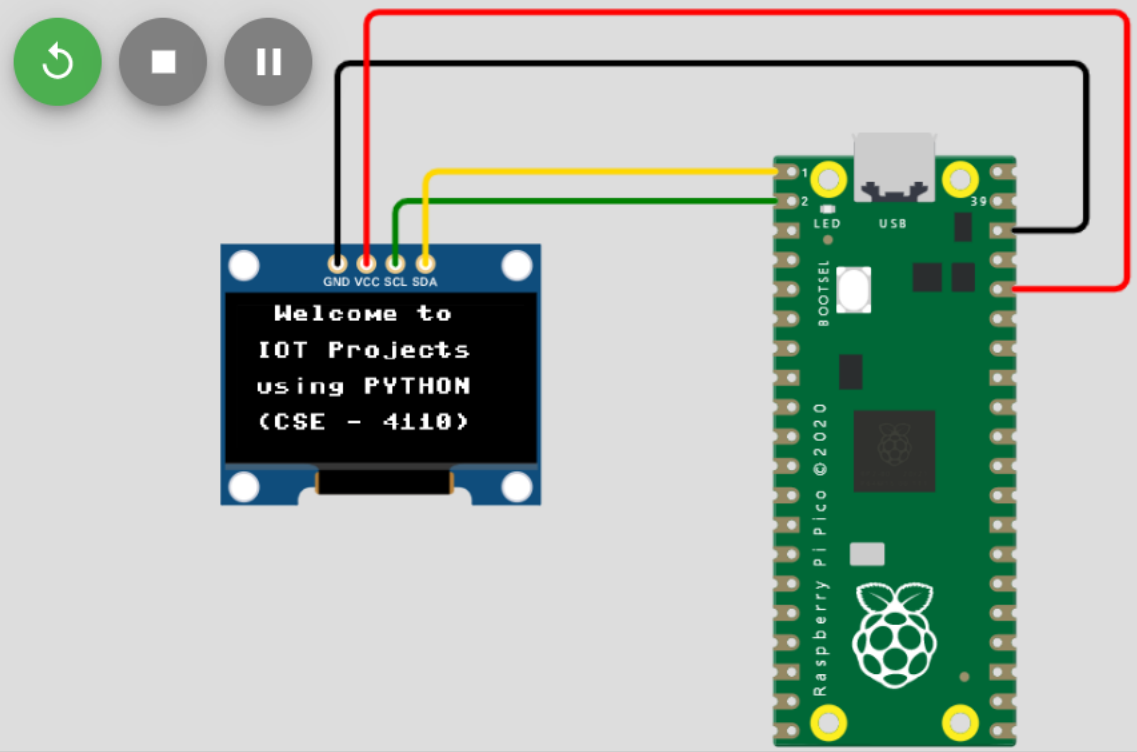
Programming an OLED Screen on Raspberry Pi Pico

WOKWI SAVE SHARE Docs B

main.py ● diagram.json ● ssd1306.py ● PIO

```
1  from machine import Pin, I2C
2  from ssd1306 import SSD1306_I2C
3
4  i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
5  oled = SSD1306_I2C(128, 64, i2c)
6
7  oled.text(" Welcome to ",8, 0)
8  oled.text("IOT Projects",8, 16)
9  oled.text("using PYTHON",8, 32)
10 oled.text("(CSE - 4110)",8, 48)
11 oled.show()
```

Simulation



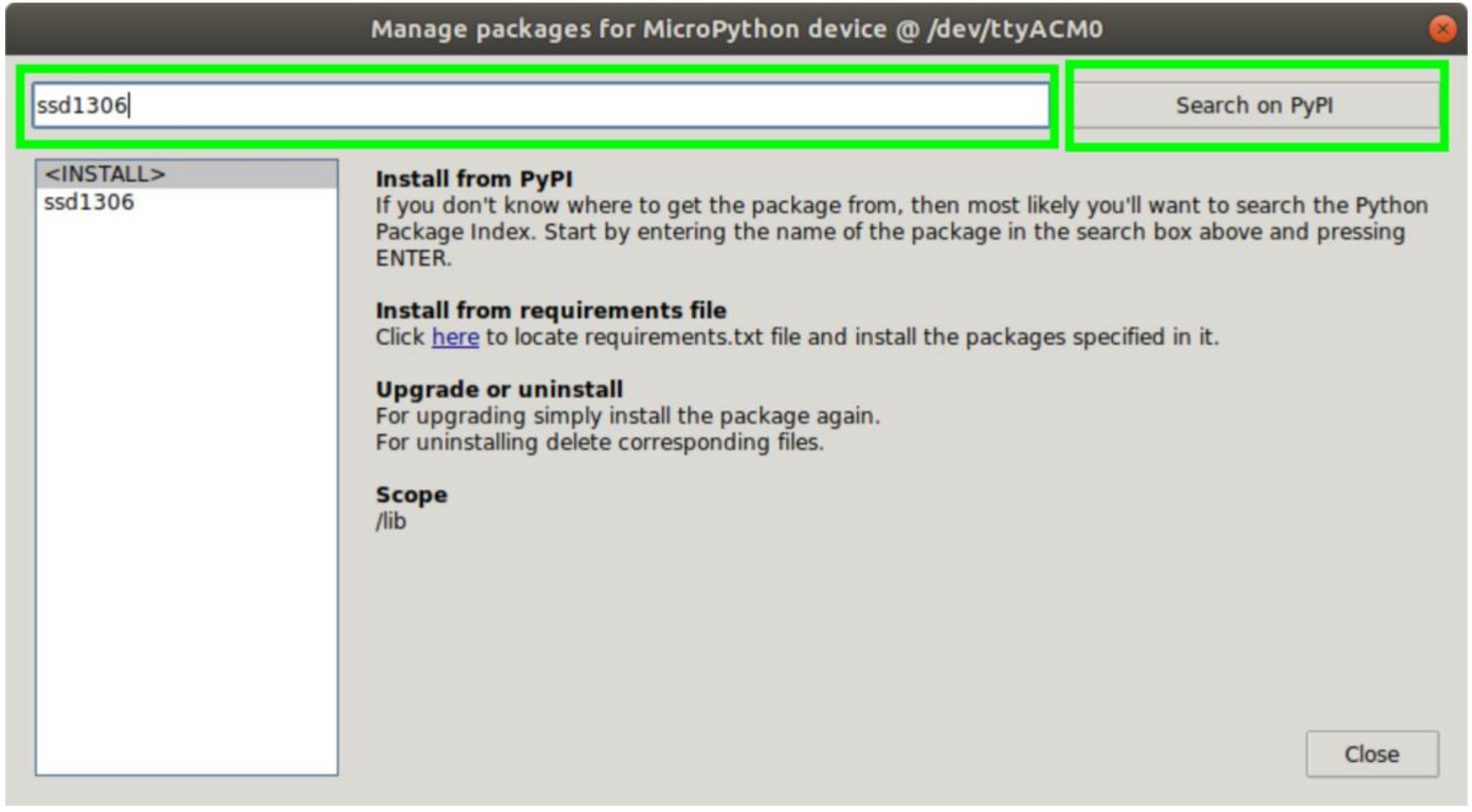

```
MPY: soft reboot
; Raspberry Pi Pico with RP2040
Type "help()" for more informatMicroPython v1.19.1 on 2022-06-18ion.
>>> ; Raspberry Pi Pico with RP2040
Type "help()" for more informat
```

How to Connect an OLED screen to THONNY

With the hardware connected and Thonny open, we now need to install a library in order for Python to communicate with the screen.

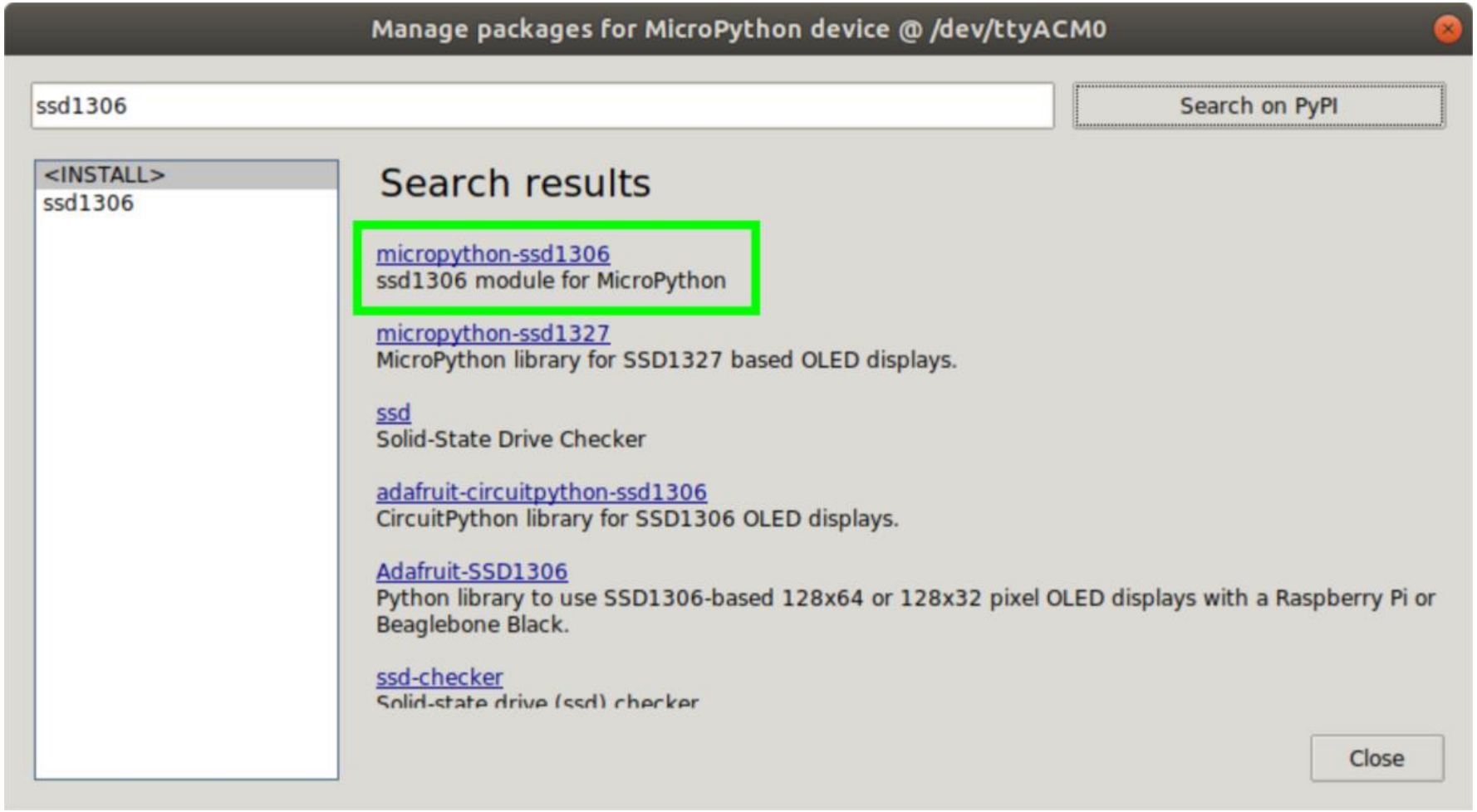
6. Click on Tools > Manage Packages to open Thonny's package manager for Python libraries.

7. Type "ssd1306" in the search bar and click "Search on PyPI".



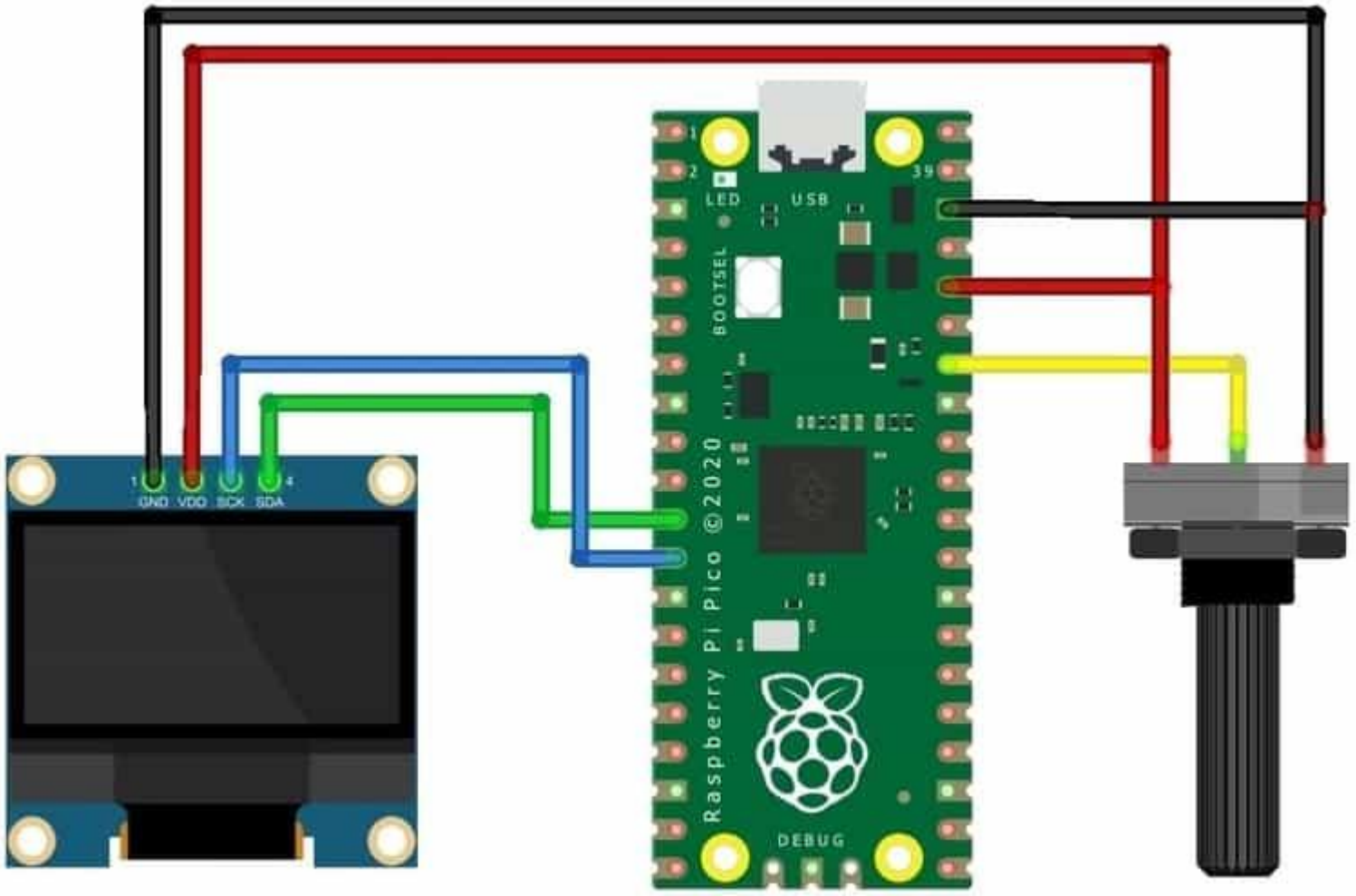
How to Connect an OLED screen to THONNY

8. Click on “micropython-ssd1306” in the returned results and then click on Install. This will copy the library to a folder, lib on the Pico.



9. Click Close to return to the main interface.

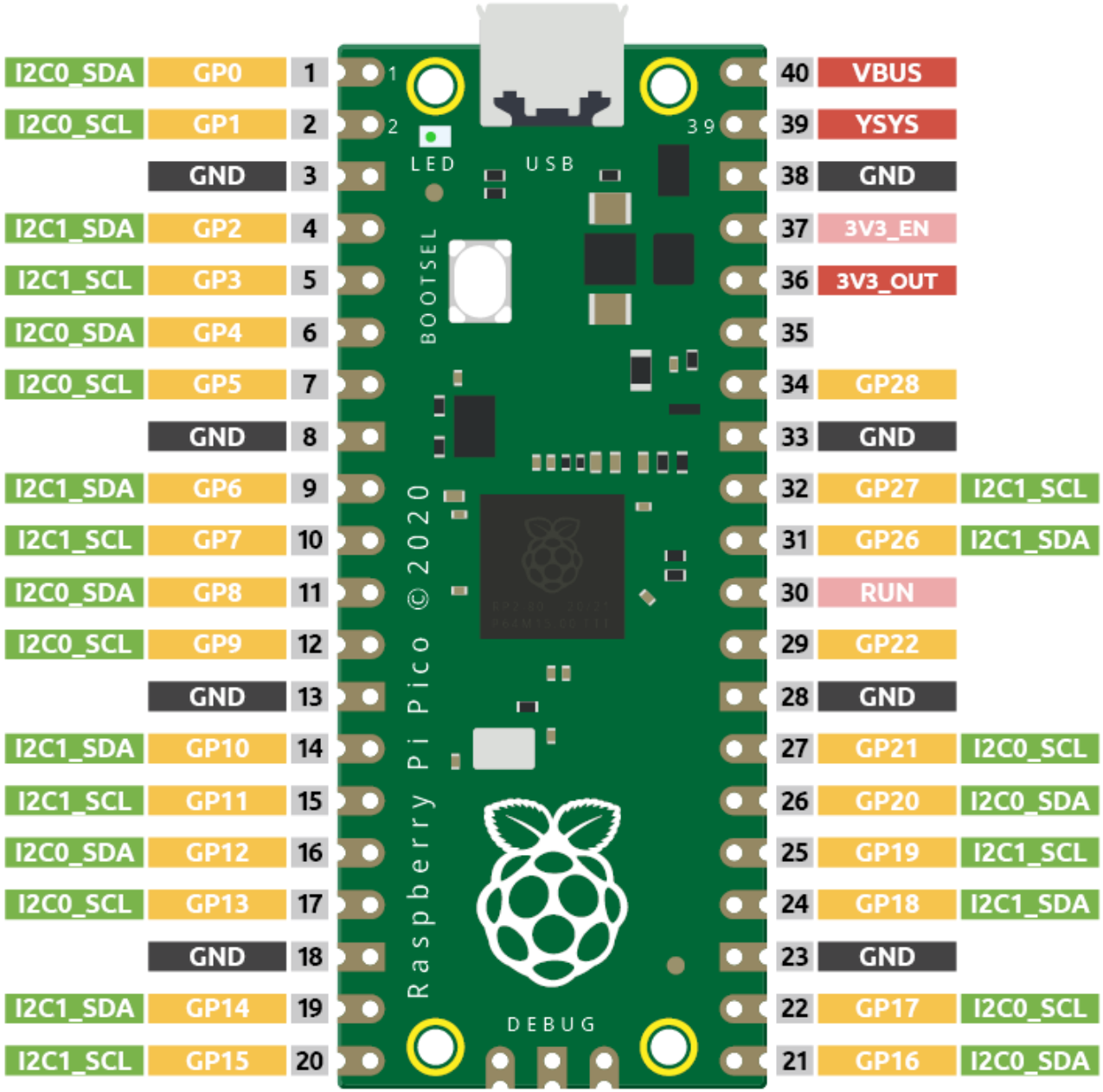
OLED screen interfacing with POT



Liquid Crystal Display

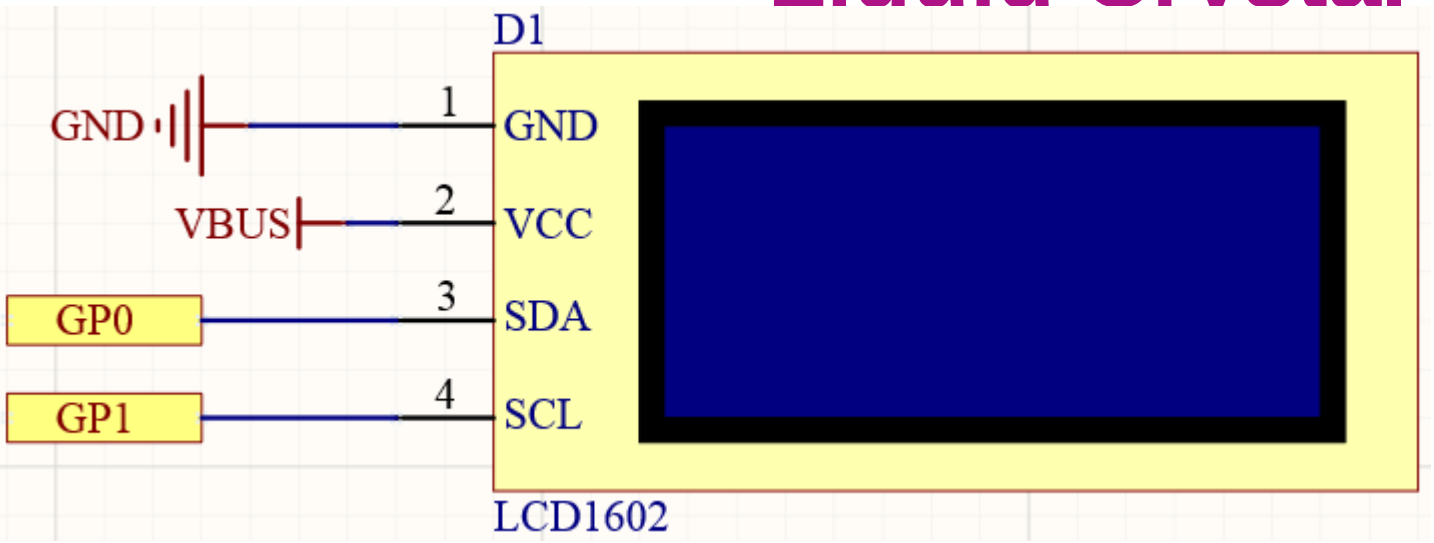
- **LCD1602 is a character type liquid crystal display, which can display 32 (16*2) characters at the same time.**
- **Though LCD and some other displays greatly enrich the man-machine interaction, they share a common weakness. When they are connected to a controller, multiple IOs will be occupied of the controller which has no so many outer ports. Also it restricts other functions of the controller. Therefore, LCD1602 with an I2C bus is developed to solve the problem.**
- **I2C(Inter-Integrated Circuit) bus is a very popular and powerful bus for communication between a master device (or master devices) and a single or multiple slave devices. I2C main controller can be used to control IO expander, various sensors, EEPROM, ADC/DAC and so on. All of these are controlled only by the two pins of host, the serial data (SDA) line and the serial clock line(SCL).**
- **These two pins must be connected to specific pins of the microcontroller. There are two pairs of I2C communication interfaces in Pico, which are marked as I2C0 and I2C1, as shown in the figure in the next slide.**

Liquid Crystal Display

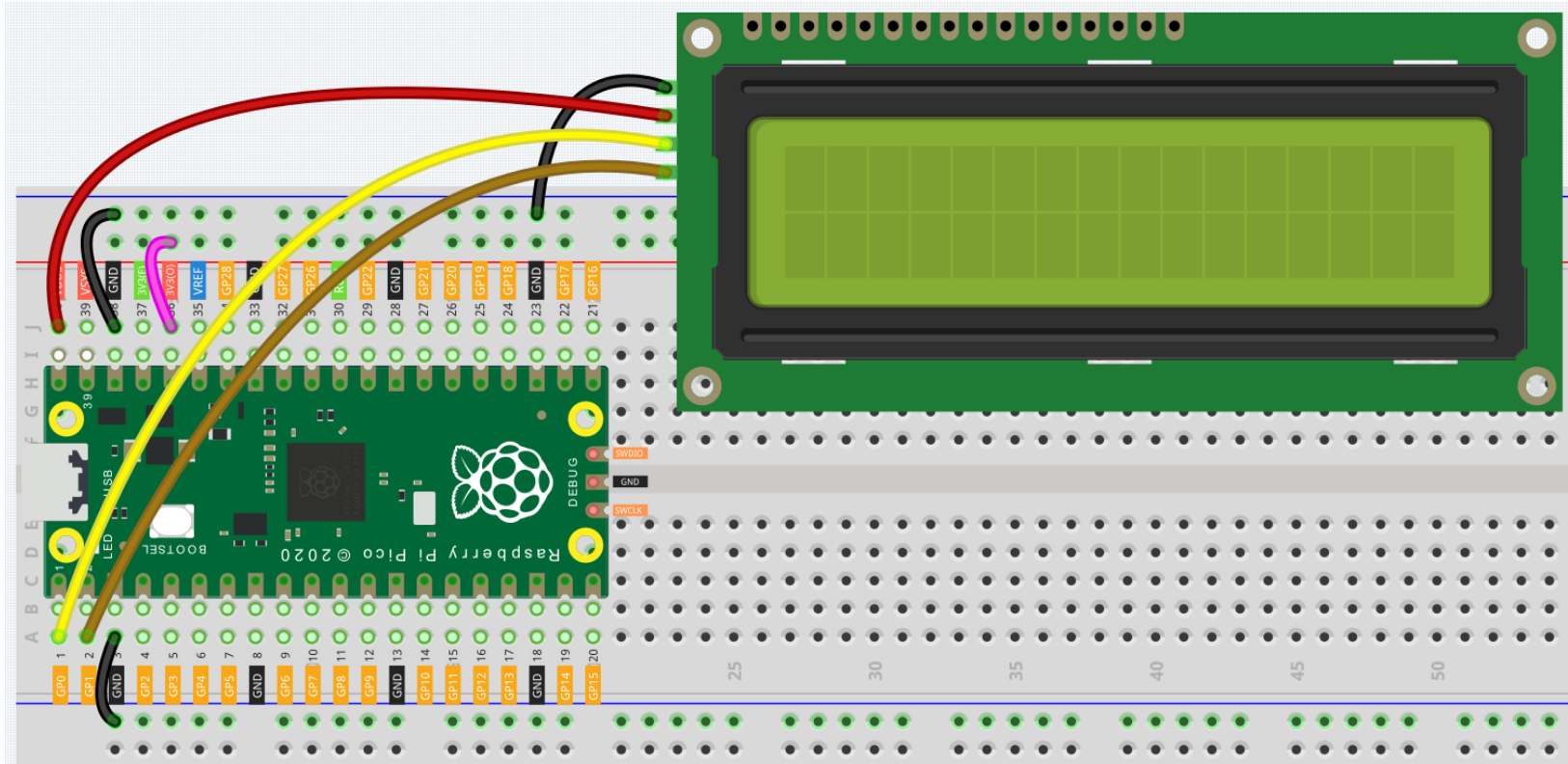


Here we will use the I2C0 interface to control the LCD1602 and display text.

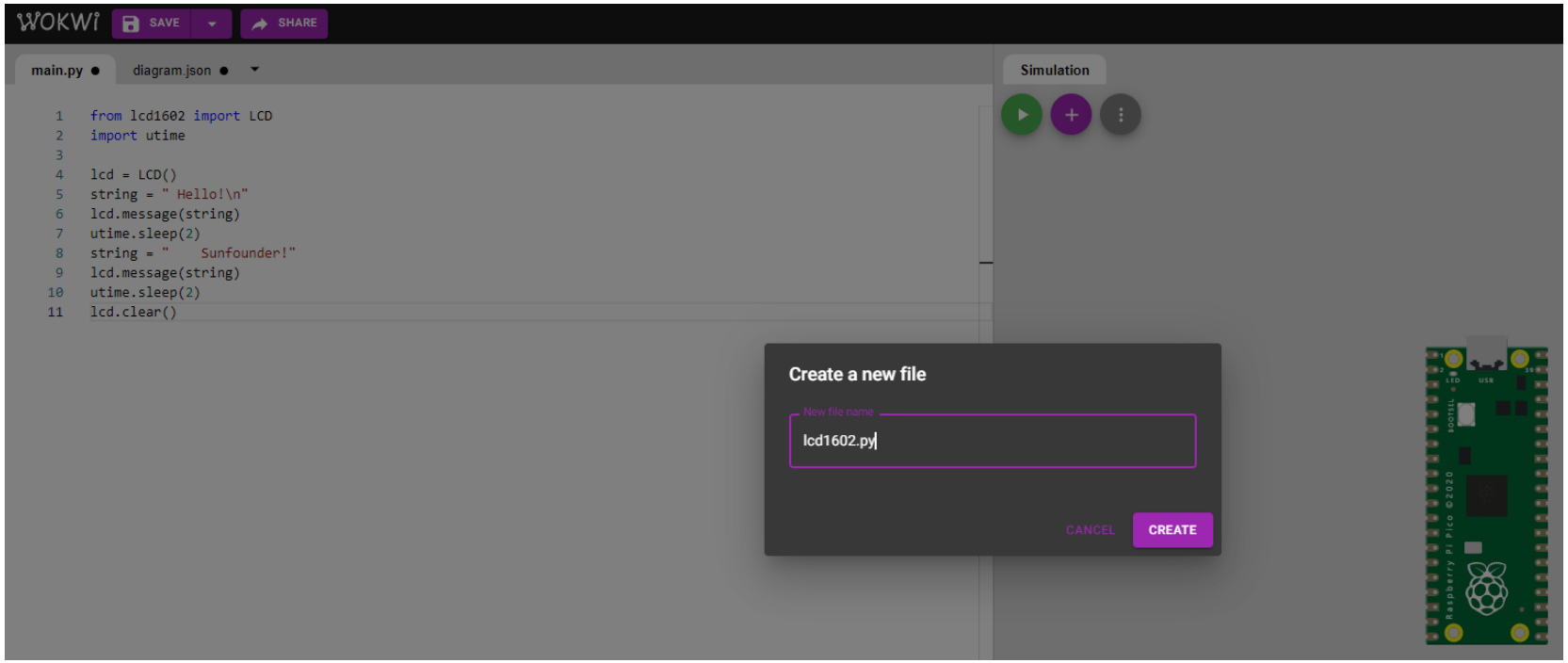
Liquid Crystal Display

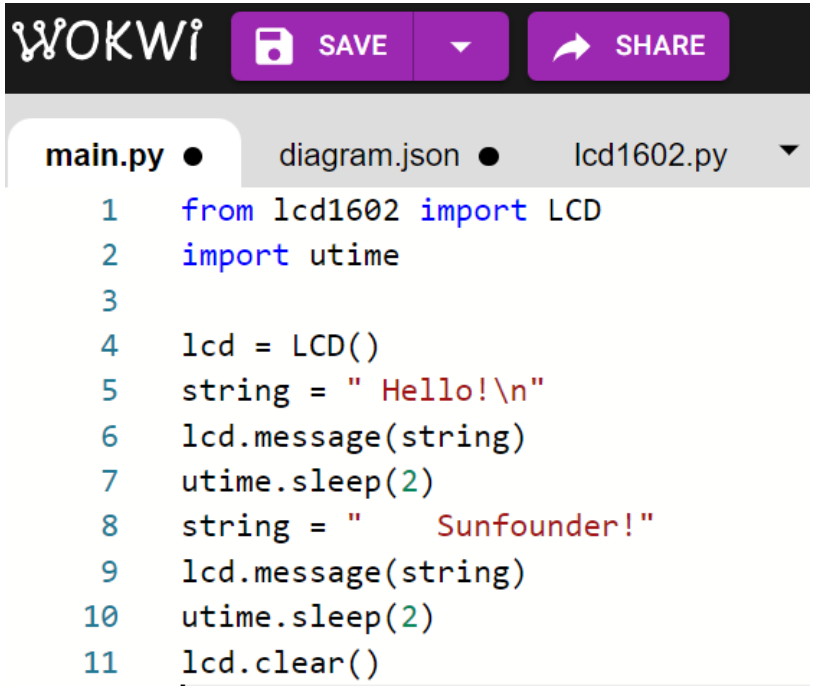


LCD 1602 with I2C



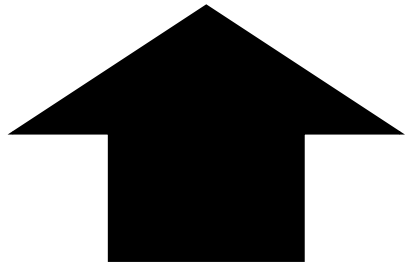
Liquid Crystal Display





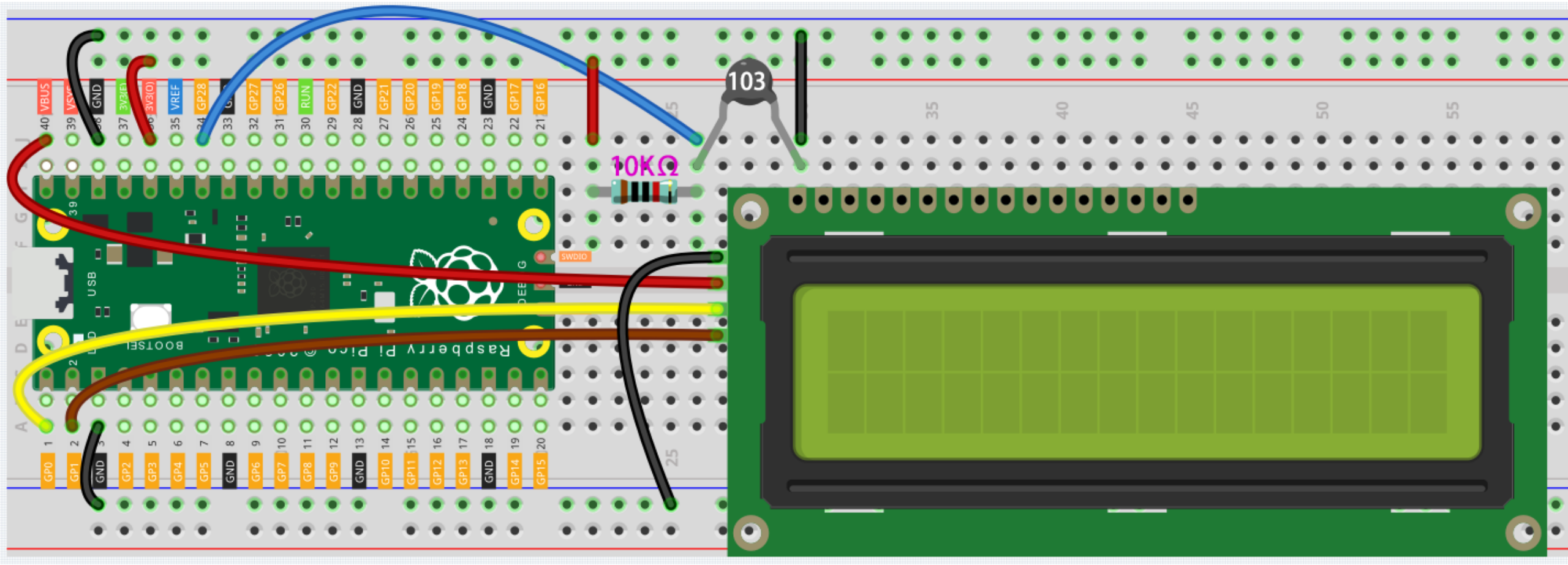
Lcd1602.py library

https://wokwi.com/projects/350811004307767891



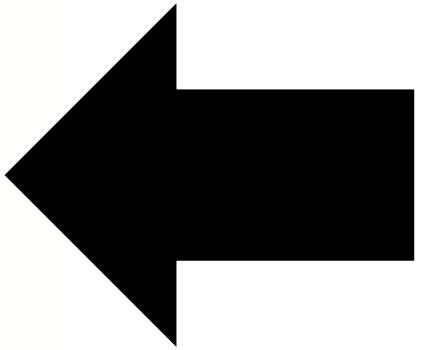
main.py

Room Température Meter



Room Température Meter

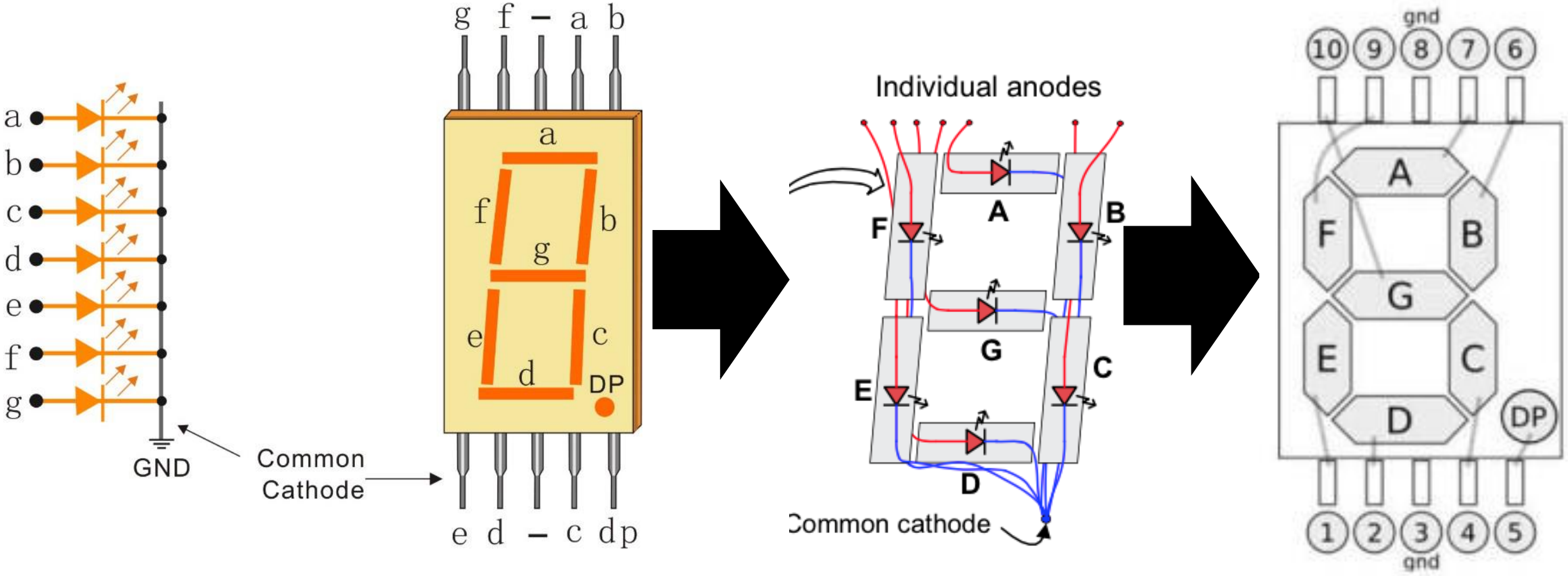
```
from lcd1602 import LCD
import machine
import utime
import math

thermistor = machine.ADC(28)
lcd = LCD()

while True:
    temperature_value = thermistor.read_u16()
    Vr = 3.3 * float(temperature_value) / 65535
    Rt = 10000 * Vr / (3.3 - Vr)
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    #Fah = Cel * 1.8 + 32
    #print ('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))
    #utime.sleep_ms(200)

    string = " Temperature is \n    " + str('{:.2f}'.format(Cel))+ " C"
    lcd.message(string)
    utime.sleep(1)
    lcd.clear()
```



main.py

LED Segment Display

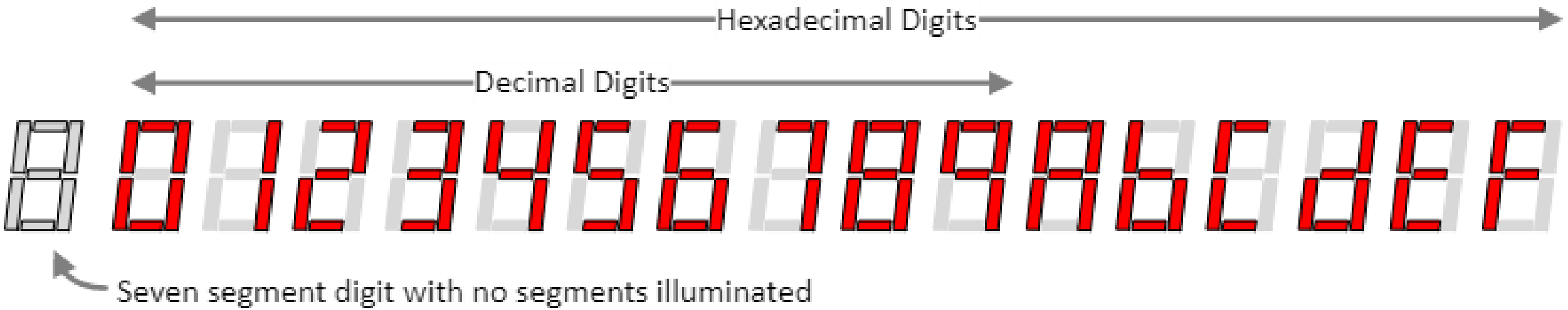
The LED Segment Display is essentially a device packaged by 8 LEDs, of which 7 strip-shaped LEDs form an “8” shape, and there is a slightly smaller dotted LED as a decimal point. These LEDs are marked as a, b, c, d, e, f, g, and dp. They have their own anode pins and share cathodes.



This means that it needs to be controlled by 8 digital signals at the same time to fully work

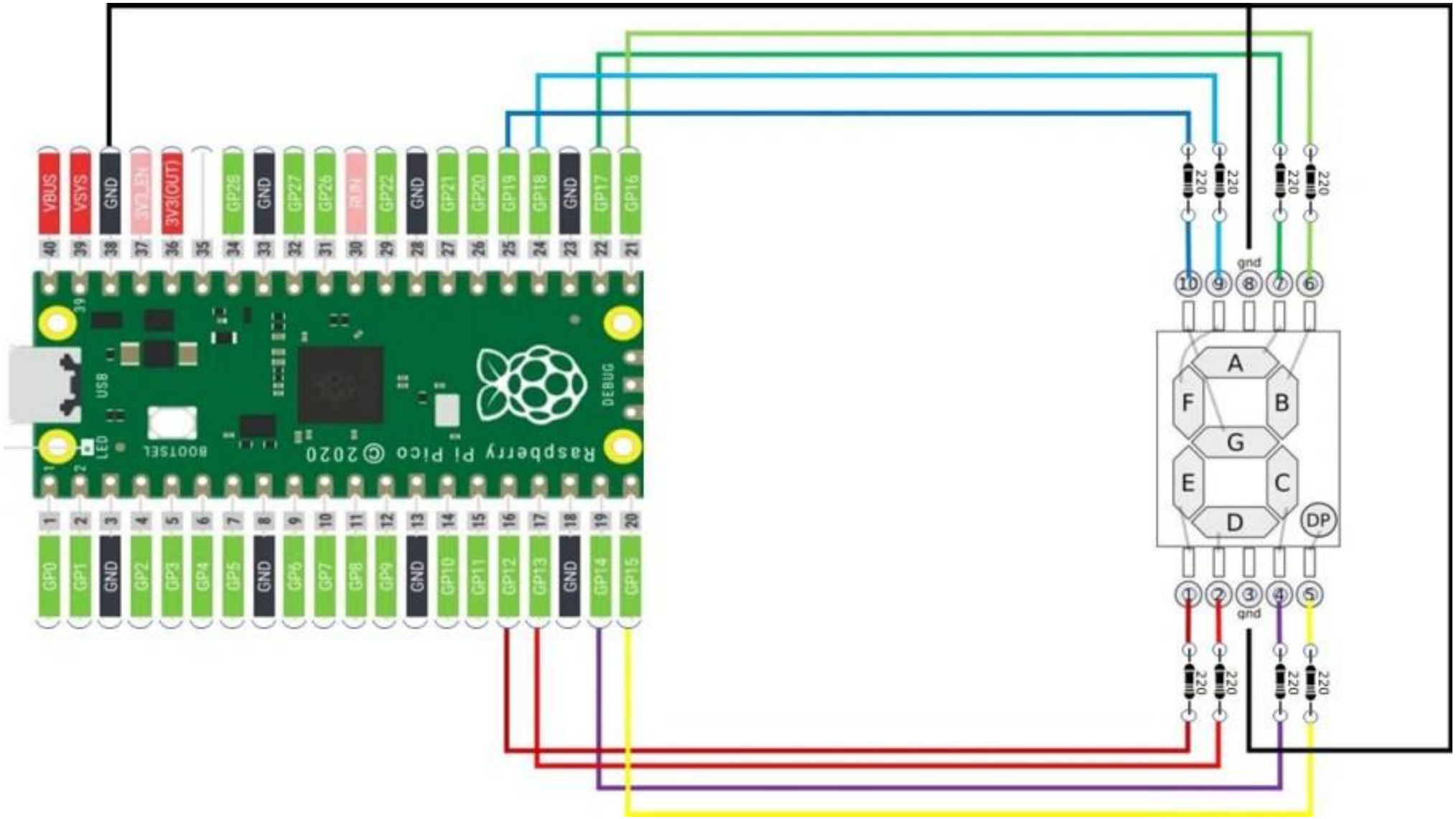
LED Segment Display

Any LED/segment can be individually illuminated, so any one of 128 different patterns



LED Segment Display

Any LED/segment can be individually illuminated, so any one of 128 different patterns



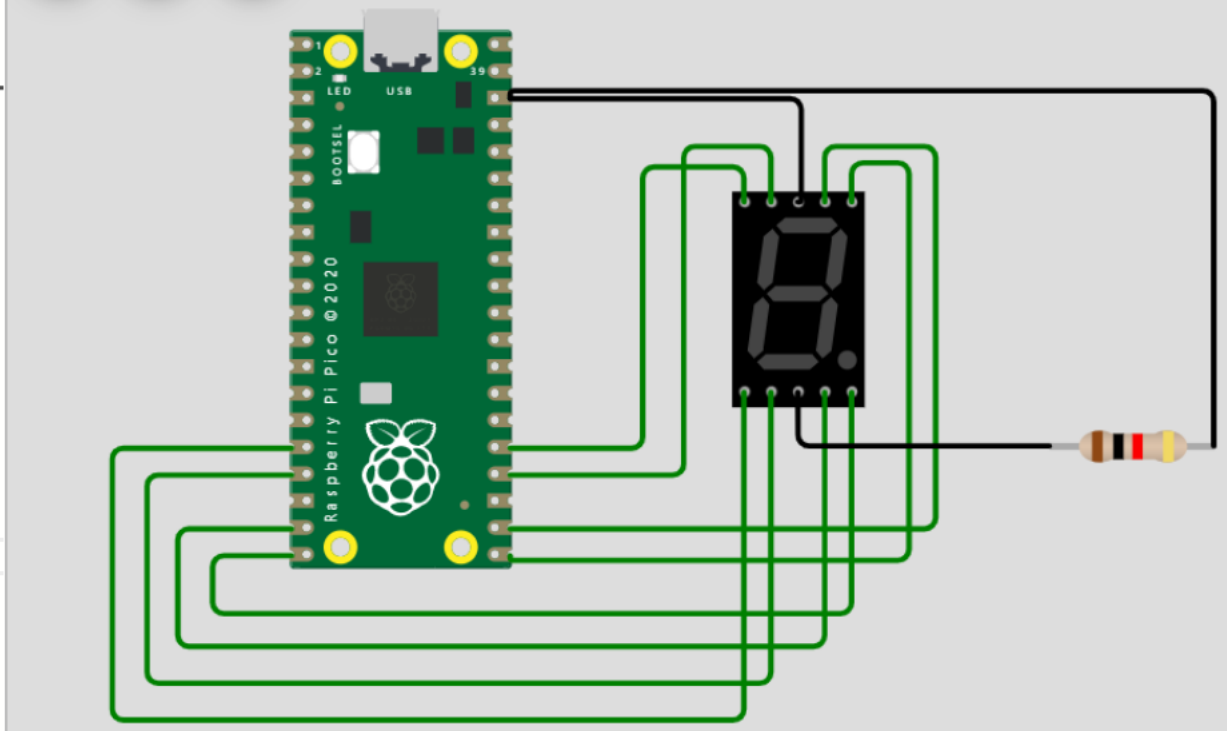
LED Segment Display

WOKWi SAVE SHARE Docs B

main.py ● diagram.json

```
1   from machine import Pin
2   from time import sleep
3
4   A_LED  = Pin(17,machine.Pin.OUT)  #Create an output pin for each segment
5   B_LED  = Pin(16,machine.Pin.OUT)
6   DP_LED = Pin(15,machine.Pin.OUT)
7   C_LED  = Pin(14,machine.Pin.OUT)
8
9   D_LED  = Pin(13,machine.Pin.OUT)
10  E_LED  = Pin(12,machine.Pin.OUT)
11  G_LED  = Pin(19,machine.Pin.OUT)
12  F_LED  = Pin(18,machine.Pin.OUT)
13
14  print("Ready, Set, GO!")
15  Seq_Del = .3
16  while True:
17      A_LED.value(1)        #Turn ON A LED segment
18      sleep(Seq_Del)        #Pause a bit
19      B_LED.value(1)
20      sleep(Seq_Del)
21      DP_LED.value(1)
22      sleep(Seq_Del)
23      C_LED.value(1)
```

Simulation



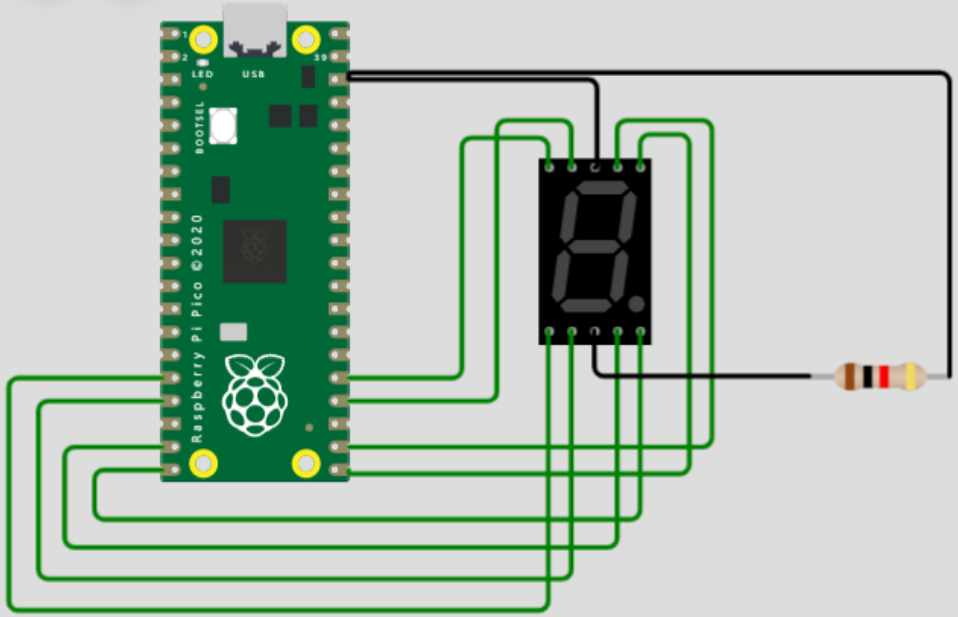
LED Segment Display

WOKWI SAVE SHARE

main.py ● diagram.json

```
24      sleep(Seq_Del)
25      D_LED.value(1)
26      sleep(Seq_Del)
27      E_LED.value(1)
28      sleep(Seq_Del)
29      G_LED.value(1)
30      sleep(Seq_Del)
31      F_LED.value(1)
32      sleep(Seq_Del)
33
34      sleep(1)
35
36      A_LED.value(0)        #Turn OFF A LED segment
37      sleep(Seq_Del)        #Pause a bit
38      B_LED.value(0)
39      sleep(Seq_Del)
40      DP_LED.value(0)
41      sleep(Seq_Del)
42      C_LED.value(0)
43      sleep(Seq_Del)
44      D_LED.value(0)
45      sleep(Seq_Del)
46      E_LED.value(0)
47      sleep(Seq_Del)
48      G_LED.value(0)
49      sleep(Seq_Del)
50      F_LED.value(0)
51      sleep(Seq_Del)
52
53      sleep(1)
```

Simulation



LED Segment Display

```
from machine import Pin

# define GP ports to use
# Pins Matching: A, B, C, D, E, F, G
display_list = [17,16,14,13,12,18,19]
dotPin=15
display_obj = []

# Set all pins as output
for seg in display_list:
    display_obj.append(Pin(seg, Pin.OUT))

dot_obj=Pin(dotPin, Pin.OUT)

# DIGIT map as array of array
arrSeg = [[1,1,1,1,1,1,0],\
          [0,1,1,0,0,0,0],\
          [1,1,0,1,1,0,1],\
          [1,1,1,1,0,0,1],\
          [0,1,1,0,0,1,1],\
          [1,0,1,1,0,1,1],\
          [1,0,1,1,1,1,1],\
          [1,1,1,0,0,0,0],\
          [1,1,1,1,1,1,1],\
          [1,1,1,1,0,1,1]]
```

```
def SegDisplay(toDisplay):
    numDisplay = int(toDisplay.replace(".", ""))
    for a in range(7):
        display_obj[a].value(arrSeg[numDisplay][a])
    # Manage DOT activation
    if toDisplay.count(".") == 1:
        dot_obj.value(1)
    else:
        dot_obj.value(0)

SegDisplay("5.")
```

If we don’t want the dot, simply call in this way:
SegDisplay("5")

The number can vary from 0 to 9 and the parameter needs to be a string. If you have integer numbers, you can use the str() function to convert them. An example:

SegDisplay(str(5))

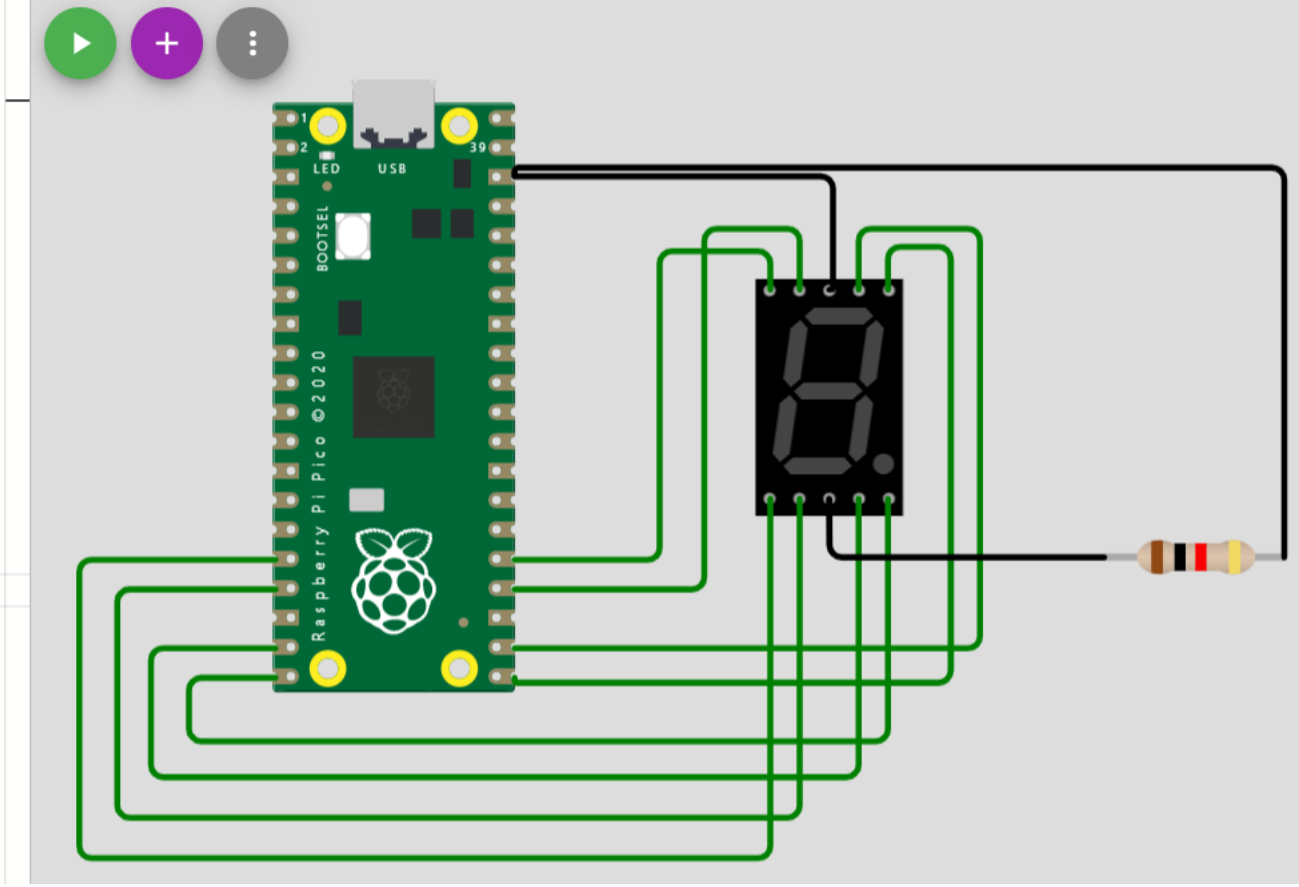
Displays number 0-9 with or without the decimal point

WOKWi SAVE SHARE Docs B

main.py ● diagram.json

```
1  import machine
2  import utime
3
4  A_LED  = machine.Pin(17,machine.Pin.OUT)  #Create an output pin for each segment
5  B_LED  = machine.Pin(16,machine.Pin.OUT)
6  DP_LED = machine.Pin(15,machine.Pin.OUT)
7  C_LED  = machine.Pin(14,machine.Pin.OUT)
8  D_LED  = machine.Pin(13,machine.Pin.OUT)
9  E_LED  = machine.Pin(12,machine.Pin.OUT)
10 G_LED  = machine.Pin(19,machine.Pin.OUT)
11 F_LED  = machine.Pin(18,machine.Pin.OUT)
12
13 def Clear_Display():
14     A_LED.value(0)       #Turn OFF all LED segments
15     B_LED.value(0)
16     DP_LED.value(0)
17     C_LED.value(0)
18     D_LED.value(0)
19     E_LED.value(0)
20     G_LED.value(0)
21     F_LED.value(0)
22
23 def Show_Number(Number, DecimalPoint):
24     Clear_Display()
25     if   Number == "0":
26         A_LED.value(1)       #Turn ON needed LED segments
```

Simulation



Displays number 0-9 with or without the decimal point

```
        B_LED.value(1)
        C_LED.value(1)
        D_LED.value(1)
        E_LED.value(1)
        F_LED.value(1)
    elif Number == "1":
        B_LED.value(1)
        C_LED.value(1)
    elif Number == "2":
        A_LED.value(1)
        B_LED.value(1)
        G_LED.value(1)
        E_LED.value(1)
        D_LED.value(1)
    elif Number == "3":
        A_LED.value(1)
        B_LED.value(1)
        G_LED.value(1)
        C_LED.value(1)
        D_LED.value(1)
    elif Number == "4":
        B_LED.value(1)
        G_LED.value(1)
        C_LED.value(1)
        F_LED.value(1)
    elif Number == "5":
        A_LED.value(1)
        F_LED.value(1)
        G_LED.value(1)
        C_LED.value(1)
        D_LED.value(1)
    elif Number == "6":
        A_LED.value(1)
        F_LED.value(1)
        E_LED.value(1)
        D_LED.value(1)
        C_LED.value(1)
        G_LED.value(1)
    elif Number == "7":
        A_LED.value(1)
        B_LED.value(1)
        C_LED.value(1)
    elif Number == "8":
        A_LED.value(1)
        B_LED.value(1)
        C_LED.value(1)
        D_LED.value(1)
        E_LED.value(1)
        G_LED.value(1)
        F_LED.value(1)
    elif Number == "9":
        A_LED.value(1)
        B_LED.value(1)
        C_LED.value(1)
        G_LED.value(1)
        F_LED.value(1)
    if DecimalPoint == ".":
        DP_LED.value(1)


print("Ready, Set, GO!")
Seq_Del = 1
while True:
    Show_Number("0","")
    utime.sleep(Seq_Del)
    Show_Number("1","")
    utime.sleep(Seq_Del)
    Show_Number("2","")
    utime.sleep(Seq_Del)
    Show_Number("3","")
    utime.sleep(Seq_Del)
    Show_Number("4","")
    utime.sleep(Seq_Del)
    Show_Number("5","")
    utime.sleep(Seq_Del)
    Show_Number("6","")
    utime.sleep(Seq_Del)
    Show_Number("7","")
    utime.sleep(Seq_Del)
    Show_Number("8","")
    utime.sleep(Seq_Del)
    Show_Number("9","")
    utime.sleep(Seq_Del)
    Show_Number("9",".")
    utime.sleep(Seq_Del)

    utime.sleep(1)
```