Freenove i2c 1602 LCD Display.

The FreeNove LCD 1602 LCD display is a low-cost LCD display screen. These can be purchased in different screen colours and are great for outputting program data e.g. temperature.

Contents

Introduction to the FreeNove i2c 1602 LCD display.

These output devices can be purchased from a wide range of vendors such as Amazon or Ali Express.

https://www.amazon.co.uk/FREENOVE-Display-Compatible-Arduino-Raspberry/dp/B0B76Z83Y4?th=1

https://www.aliexpress.com/item/1005006478068684.html

Be aware that the labelling and order of the 4 pins can vary! Connect the jumper wires to these 4 pins and then see the wiring diagram below to see how to connect to the Pico.

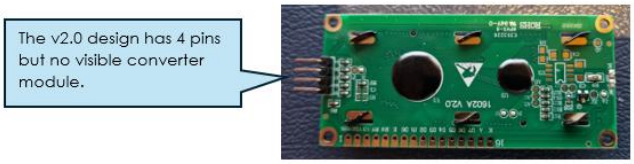
If your LCD Display has an **SCL** pin you probably need to wire it as follows. See wiring diagram to see where these coloured wires connect to the breadboard. The Waveshare LCD screen has this layout.

This FreeNove LCD screen v2.0 (from Amazon) has slightly different labels for its pins e.g.

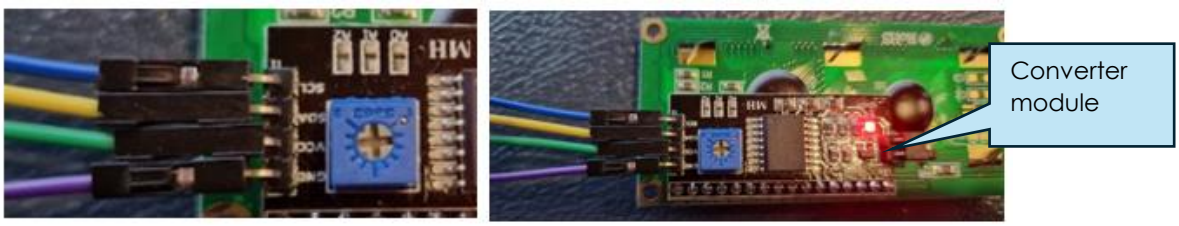
- **VSS** instead of GND (this is the negative power wire)
- **VDD** instead of VCC (this is the positive power wire)
- **SDA** is the same on both (this is the signal wire)
- **SCK** instead of SCL (this is the serial clock wire)

You do not need to know exactly what these pins do at this stage. Wire up the pins according to the colours in the diagram. Then view the wiring diagram to see where they fit into the breadboard.

The back of this FreeNove LCD screen v2.0 does **not** have the separate converter. It still has 4 pins but this new design has the converter built into the main board.



The older version of the FreeNove LCD screen (see below) has a back with a black circuit board on it. This is the 'converter module'. It means we only need 4 pins to control it.

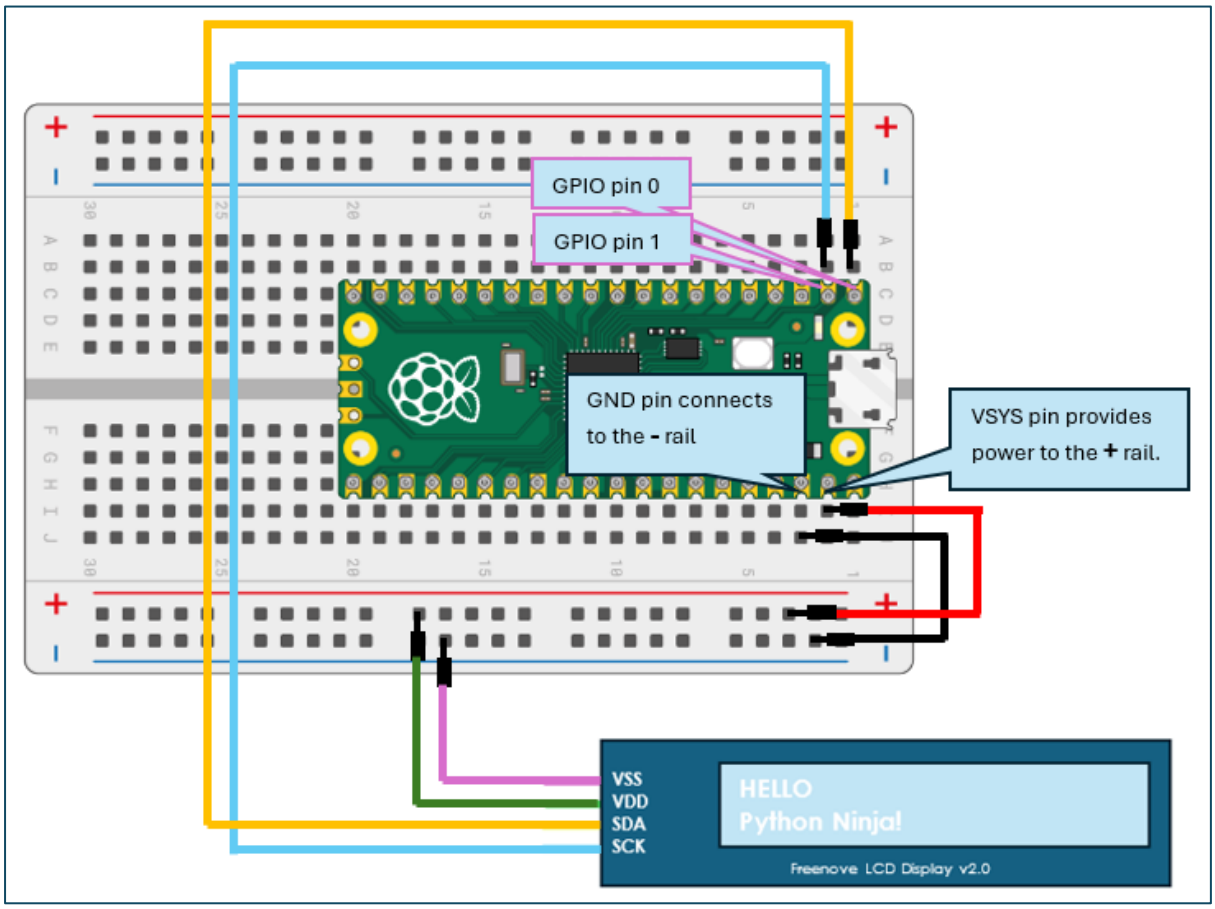


Wiring Diagram

When wiring up your FreeNove i2c 1602 LCD screen, pay close attention to labels on the 4 pins.

The **red** and **black** jumper wires connect the + and – power pins on the Pico, to the + and – breadboard rail.

The breadboard allows us to easily connect Pico Pins to other hardware devices. If we peeled the back off the breadboard (please don't) we can see how the holes in the breadboard are connected up.



The **purple** and **green** wires provide power to the LCD display.

The **yellow** and **blue** wires send control signals to the LCD Display.

The SCK pin on the Freenove LCD display refer to the 'Serial Clock'. The Waveshare version labels this as SCL. We treat them as the same thing.

Now you have the LCD screen wired up, we can install the libraries to the Pico.

Installing the Libraries

A library is provided code we can refer to in our programming. It means the coding we have to do is a lot simpler.

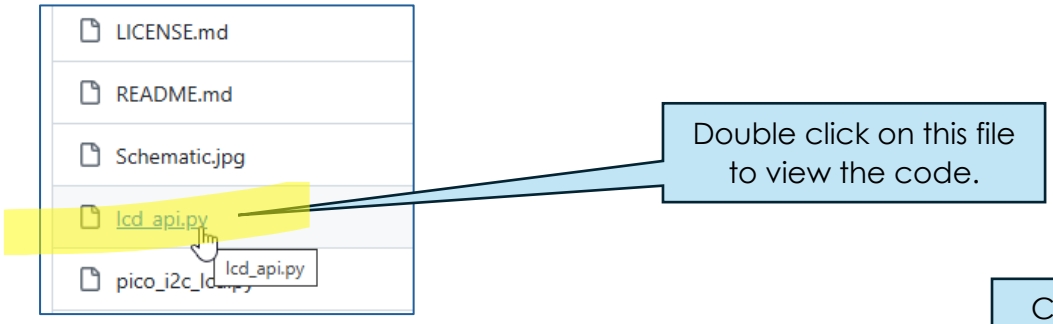
We need to import two libraries into the Pico.

The lcd_api.py library

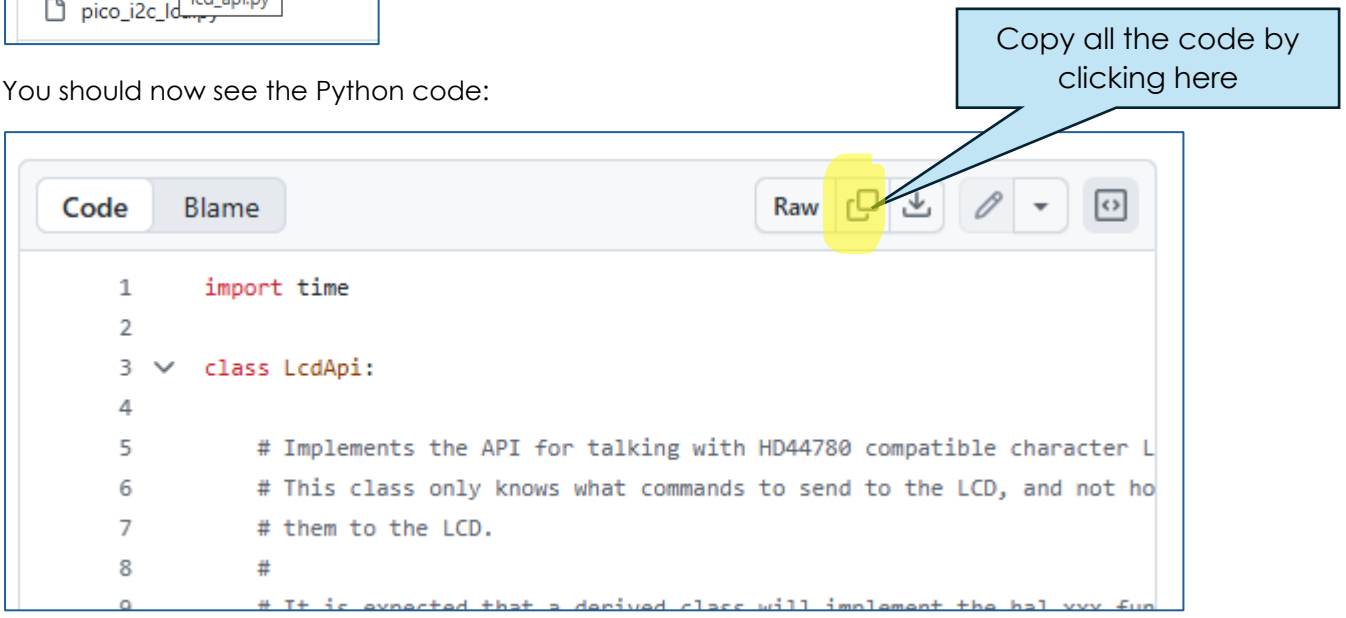
The first file to save is the **lcd_api.py** library.

Open this GitHub page: https://github.com/T-622/RPI-PICO-I2C-LCD

You will see several files. Select this one.

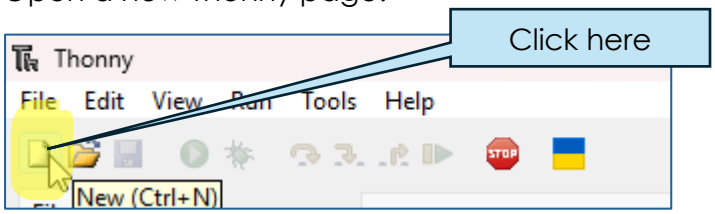


You should now see the Python code:



Now go back to the Thonny app.

Open a new Thonny page:



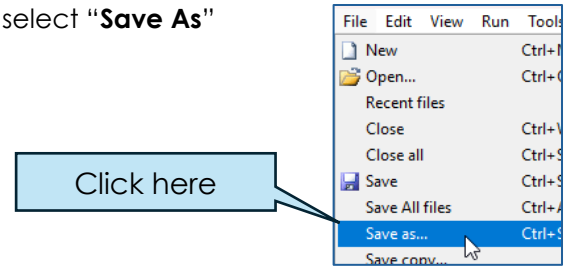
Paste the code you copied into this new Thonny page (CTRL V).

Your Thonny page should look like this.

```
<untitled> *
156         location &= 0x7
157         self.hal_write_command(self.LCD_CGRAM | (location << 3))
158         self.hal_sleep_us(40)
159         for i in range(8):
160             self.hal_write_data(charmap[i])
161             self.hal_sleep_us(40)
162         self.move_to(self.cursor_x, self.cursor_y)
163
164     def hal_backlight_on(self):
165         # Allows the hal layer to turn the backlight on.
166         # If desired, a derived HAL class will implement this fu
167         pass
```

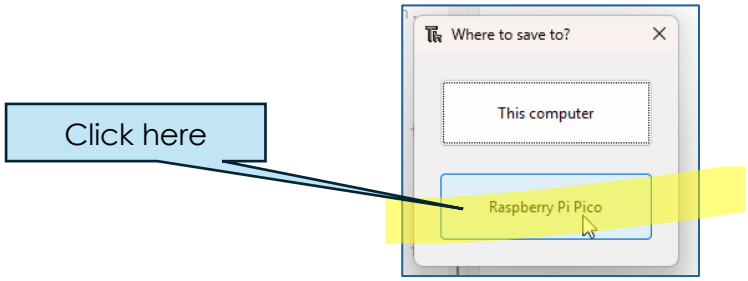
Now, we need to save this **onto the Pico.**

Click "**File**", then select "**Save As**"

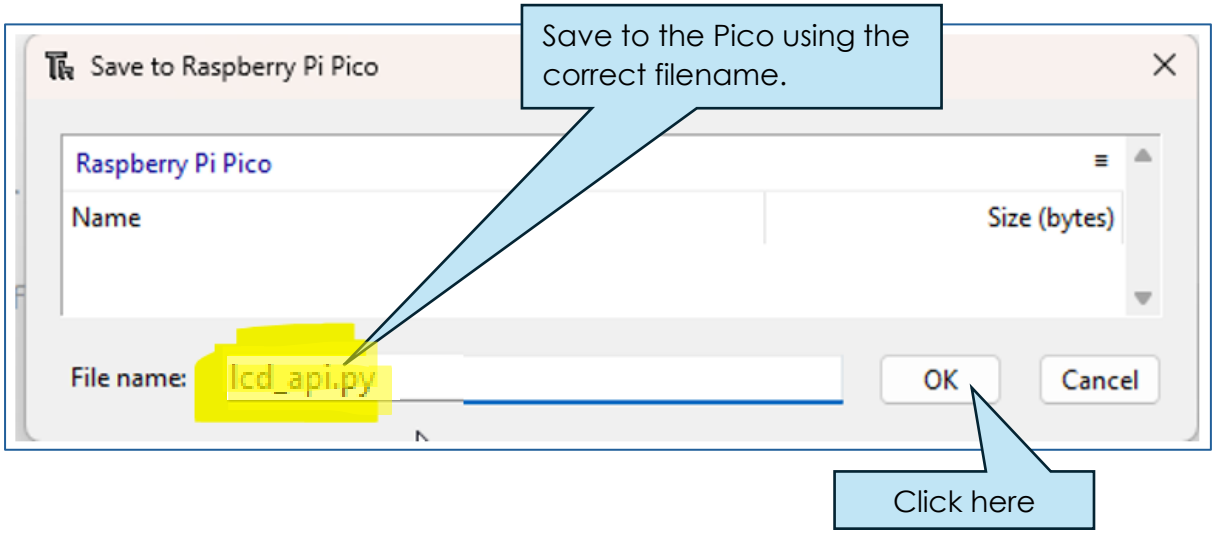


You will get a pop up window offering two choices.

Choose the 'Pico' option:

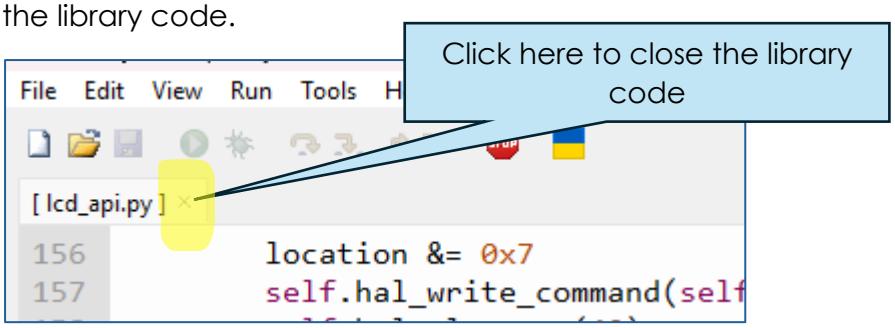


We need to save it using the correct file name or the code you write will not find this library. Save it as **lcd api.py**



The lcd_api.py file has now been saved to the Pico.

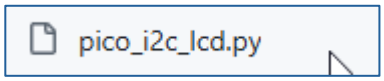
We should now close this open Thonny page to ensure we don't accidentally edit the library code.

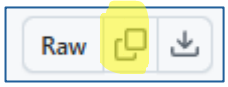


We now need to repeat this process for the second library file.

The pico_i2c_lcd.py library

Open this GitHub page: https://github.com/T-622/RPI-PICO-I2C-LCD

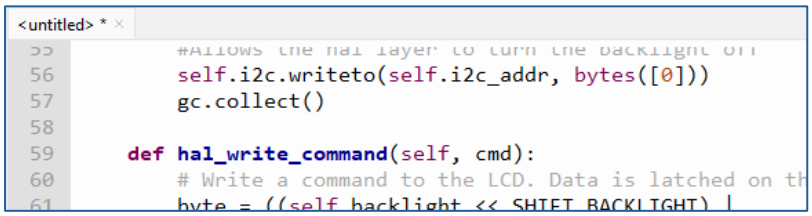
Open the **pico_i2c_lcd.py** file 

Copy the code 

Open a new page in Thonny 

Paste the code (CTRL V)

You should see the library code appear as a Thonny page like this:



We now need to save this library code to the Pico device.

Click "**File**", then select "**Save As**"

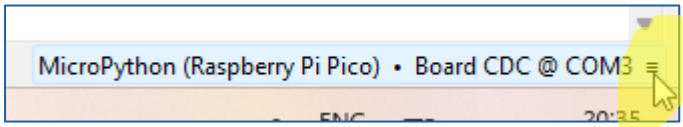
Select the "Pico" option.

Save it as **pico_i2c_lcd.py**

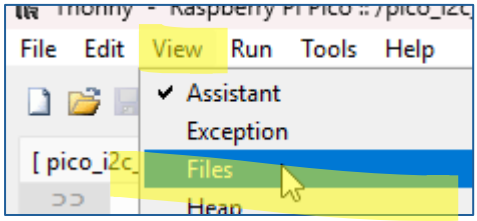
Checking what has been saved to the Pico

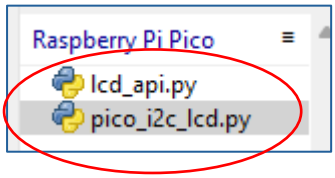
Connect the Pico to the PC.

Make sure Thonny has selected the Pico option in the bottom right corner.



In Thonny, click "**View**", then "**Files**"





You should see a window (bottom left) showing files saved to the Pico.

Scan to find out the address of the LCD

We now need to find out the 'address' of the LCD device we have just connected to our Pico. If we do not have the correct address, our Pico cannot 'talk' to the LCD display.

Open a new Thonny page.

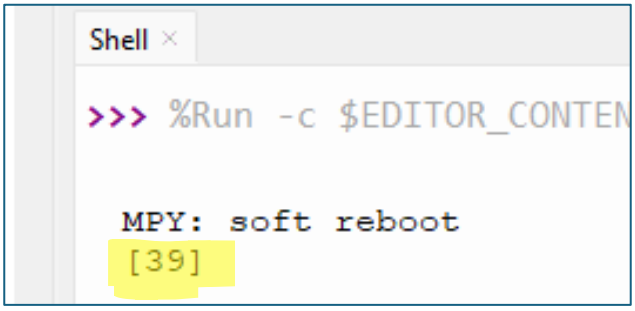
Copy/Paste the scan code below

Scan Code:

```
import machine
sda = machine.Pin(0)
scl = machine.Pin(1)
i2c = machine.I2C(0,sda=sda,scl=scl, freq=400000)
print(i2c.scan())
```

Run the program.

View the address in the shell.



You may get a different number value.

Remember it. We will be using it shortly.

We can now close the Thonny page with the scan code.

We do not need to save it.

Output Text to our LCD Display

Let's output a simple text message to the LCD display screen. It is always good practice to test connections before attempting a more complex program.

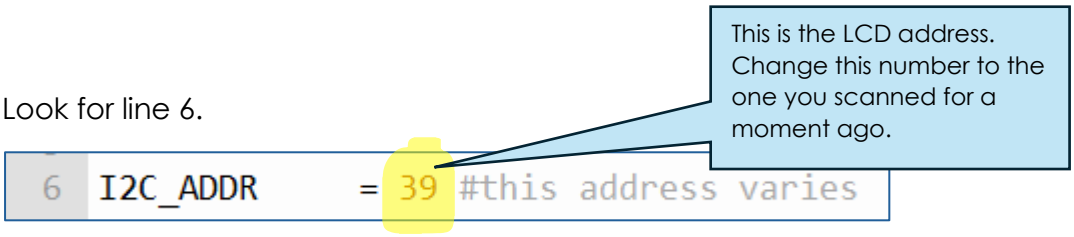
Open a new Thonny page.

Copy/paste the code below into Thonny.

Test code – Welcome Python Ninja

```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #this address varies
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

lcd.clear()
lcd.move_to(5,0)
lcd.putstr("Welcome")
lcd.move_to(3,1)
lcd.putstr("Python Ninja")
time.sleep(5)
lcd.clear()
```

Look for line 6.



Now, run this program.

Did your LCD display the text? Woohoo!

Troubleshooting

Things to check

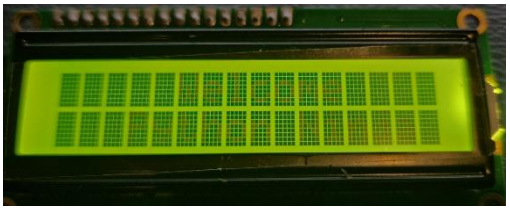
- Are the two signal wires connecting to the correct pins (see wiring diagram)
- Are the two power wires connecting to positive and negative (see wiring diagram)
- Have you set the ITC ADDR address to match the number you scanned? (code line 6)

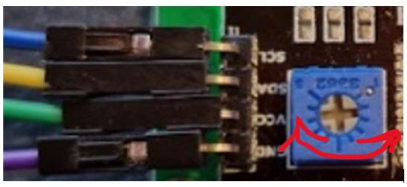
Adjust contrast

A common issue is that the older version of the LCD display (with the black converter on the back) sometimes has the contrast set too low or too high.

If you see a blank screen, or just a screen with these black boxes you might need to adjust the contrast.

You can adjust the contrast by turning this screw on the back.





Understanding the Code!

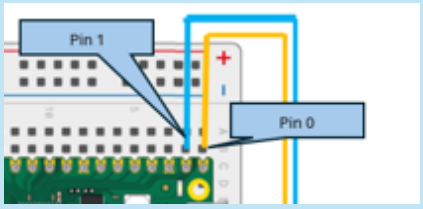
Let's identify some key commands.

```
1  import time
2  from machine import I2C
3  from lcd_api import L
4  from pico_i2c_lcd im t I2cLcd
5
6  I2C_ADDR     = 39 #this address varies
7  I2C_NUM_ROWS = 2
8  I2C_NUM_COLS = 16
9
10 i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
11 lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)
12
13 lcd.clear()
14 lcd.move_to(5,0)
15 lcd.putstr("Welcome")
16 lcd.move_to(3,1)
17 lcd.putstr("Python Ninja")
18 time.sleep(2)
19 lcd.clear()
```

This number must match the output from the i2cscan

(your number may be different)

These pins map to the Pico pins we wired up earlier.



This sets the starting point of the "welcome" line of text.

Cell 5 on row 0.

This defines the text to output on line 0.

This sets the starting point of the "Python Ninja" line of text. Cell 3 on row 1.

This defines the text to output on line 1.

The command on line 14 sets cell 5 and row 0 as the starting position, before it outputs "Welcome"


```
14  lcd.move_to(5,0)
```

Cell

Row

The command on line 16 sets the starting point as the 3rd cell on row 1 (the 2nd row), before it outputs "Python Ninja".

Challenge – Hello World

Make the LCD display "Hello World".

Make it flash on and off 3 times.

Solution 1: Displays "Hello World" once

Solution code is provided for you below, but I recommend you attempt the challenge before looking at this code.

```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #this address varies
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

lcd.clear()
lcd.move_to(0,0)
lcd.putstr("HELLO")
time.sleep(2)

lcd.move_to(0,1)
lcd.putstr("WORLD")
time.sleep(5)
lcd.clear()
```

Solution 2: Displays "Hello World" flashing

The solution below uses a for loop to repeat a sequence of commands.

```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #this address varies
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

for x in range(3):
    lcd.clear()
    lcd.move_to(0,0)
    lcd.putstr("HELLO")
    time.sleep(1)

    lcd.move_to(0,1)
    lcd.putstr("WORLD")
    time.sleep(2)
    lcd.clear()
```

Challenge – Scrolling text

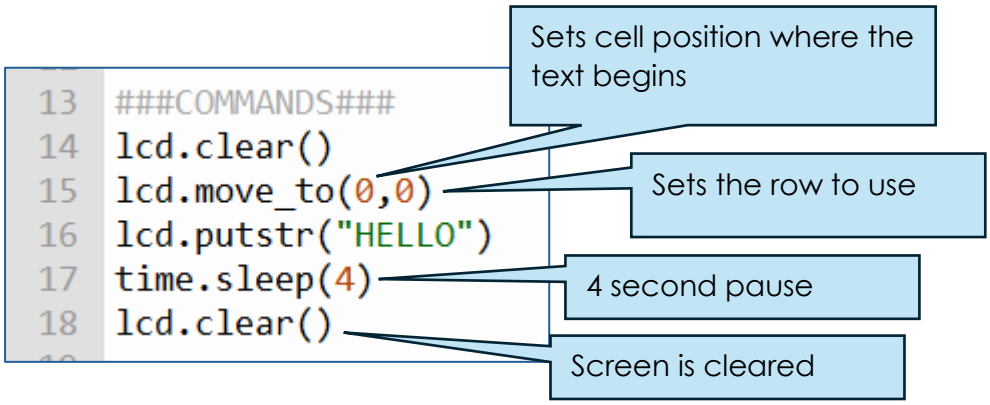
Use this test code to make the word "HELLO" appear on the LCD Display.

Remember, when using this test code, you will need to change the code on line 7 to match the LCD address you scanned for.

Test Code: displays "HELLO"

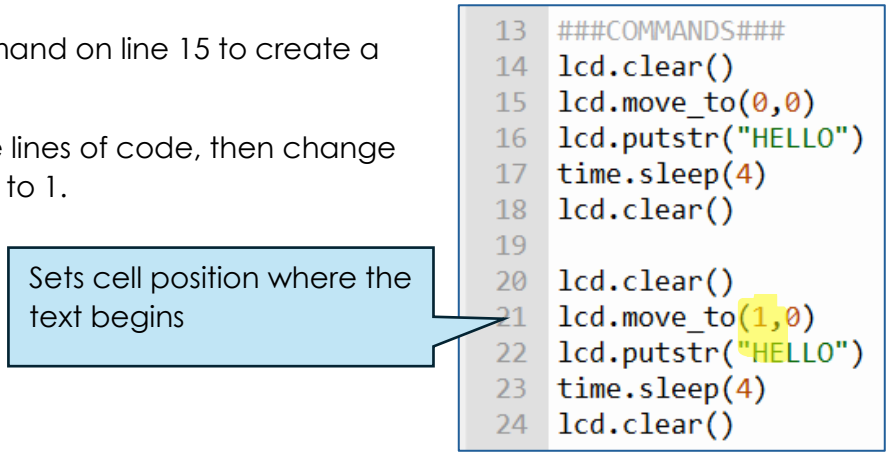
```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #change this to your LCD Address
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

###COMMANDS###
lcd.clear()
lcd.move_to(0,0)
lcd.putstr("HELLO")
time.sleep(4)
lcd.clear()
```

Understanding the Code:



We can use the command on line 15 to create a scrolling effect.

Copy and paste these lines of code, then change the starting cell from 0 to 1.



Let's paste another paragraph of code. Also, change the time.sleep(4) to time.sleep(1).

```
13  ###COMMANDS###
14  lcd.clear()
15  lcd.move_to(0,0)
16  lcd.putstr("HELLO")
17  time.sleep(1)
18  lcd.clear()
19
20  lcd.clear()
21  lcd.move_to(1,0)
22  lcd.putstr("HELLO")
23  time.sleep(1)
24  lcd.clear()
25
26  lcd.clear()
27  lcd.move_to(2,0)
28  lcd.putstr("HELLO")
29  time.sleep(1)
30  lcd.clear()
```

Your "HELLO" text should scroll across the screen, like this:

We could keep copy/pasting.

But this is repeating many very similar lines of code.

There is a more efficient way, using a For Loop.

This For Loop will loop 5 times.

The variable x has a starting value of 0.

It will increase by 1 on each loop (called incrementing).

```
13  ###COMMANDS###
14  for x in range(5):
15      lcd.clear()
16      lcd.move_to(x,0)
17      lcd.putstr("HELLO")
18      time.sleep(1)
19      lcd.clear()
20
```

So, we can use the X variable on line 16.

On the first loop, x = 0.

On the 2nd loop, x = 1, and so on.

Can you make a text message scroll across your LCD Display?

Solution:

```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #change this to your LCD Address
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)
```

```
###COMMANDS###
for x in range(10):
    lcd.clear()
    lcd.move_to(x,0)
    lcd.putstr("HELLO")
    time.sleep(0.5)
    lcd.clear()
```

Challenge - Output the Temperature

The Pico has a built-in temperature sensor. We can output these temperature readings to our LCD Display. This is a great way to learn how to use LCD displays because we can use the built in temperature sensor built into the Pico.

Step 1 - Make temp data appear in the Thonny Shell

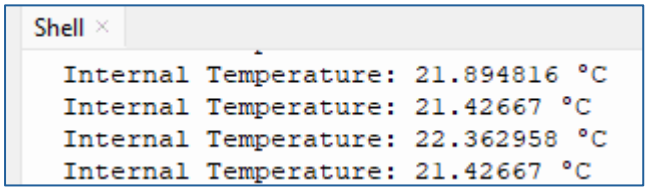
This test code is used to check the internal temperature sensor is outputting a reading. We are not displaying this data on the LCD screen yet. We should see it in the Thonny shell.

Test Code: Output temp to the shell

Copy/paste this code into Thonny and run it.

```
from machine import ADC
import time
# Internal temperature sensor is connected to ADC channel 4
temp_sensor = ADC(4)

while True:
    adc_value = temp_sensor.read_u16() # Read the raw ADC value
    voltage = adc_value * (3.3 / 65535.0)# Convert ADC value to voltage
    temp = 27 - (voltage - 0.706) / 0.001721 # Temperature calculation based on sensor characteristics
    print("Internal Temperature:", temp, "°C")
    time.sleep(1)
```

You should see it output temperature readings every second.



Put your finger onto the Pico device. The warmth should change the temperature readings being output.

We do not need so many decimal points in the output. Let's use string formatting to reduce this down to two decimal places.

Step 2 - Reducing the number of decimal places in the output

Using the test code above, we get an output with 6 decimal places.

```
Shell
Internal Temperature: 22.362958 °C
Internal Temperature: 22.362958 °C
```

We can reduce this down to 2 decimal places using string formatting.

This is the print command that outputs data to the shell.

```
10      print("Internal Temperature:", temp, "°C")
```

Change this print statement, using string formatting to limit the number of decimal places to two.

```
10      print("Internal Temperature:  {:.2f} ".format (temp, "°C"))
```

You should now get an output to the shell like this.

```
Shell
Internal Temperature:  22.36
Internal Temperature:  22.36
Internal Temperature:  22.36
```

Notice that we have list the °C part of the output.

Let's amend our print statement again.

```
10      print("Internal Temperature:  {:.2f} ".format (temp),"°C")
```

Your output to the shell should now look like this:

```
Shell
Internal Temperature:  22.36  °C
Internal Temperature:  22.36  °C
```

We have now developed a working program that outputs temperature to the Thonny shell.

Let's develop this so the temperature data is output to the LCD Display.

Step 3 – output the temp data onto the LCD as string

The LCD display can only read **string** values.

String = text data type.

The temperature value is in a **float** data type.

Float = number with a decimal point.

We can output **float** values to the LCD, but only if we convert the number to **string**. When we change the data type, this is called 'casting'.

Let's look back at the test code we used to output this text.

```
13  lcd.clear()
14  lcd.move_to(5,0)
15  lcd.putstr("Welcome")
16  lcd.move_to(3,1)
17  lcd.putstr("Python Ninja")
18  time.sleep(5)
19  lcd.clear()
```

This command outputs a string value. We can do something like this to output a variable.

Line 15 gives us a command we can use to output the temperature value.

Can you combine this with the temp sensor code so it outputs temp to the LCD?

We need this code from the Test code that displayed "Hello World" to the LCD.

Hello World Program Code

```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #this address varies
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)
```

We also need this code from the Test Code that output temp to the shell.

Temp to Shell program code

```
from machine import ADC
import time
# Internal temperature sensor is connected to ADC channel 4
temp_sensor = ADC(4)

while True:
    adc_value = temp_sensor.read_u16() # Read the raw ADC value
    voltage = adc_value * (3.3 / 65535.0)# Convert ADC value to voltage
    temp = 27 - (voltage - 0.706) / 0.001721 # Temp calc based on sensor characteristics
    print("Internal Temperature:  {:.2f} ".format (temp),"°C")
    time.sleep(1)
```

Line 10 uses the print command.

We cannot use this with the LCD Display.

To send code to the LCD Display we use **lcd.putstr()**

However, these LCD Screens cannot output number data types such as integer (whole number) or float (number with a decimal point).

Here is a very simple illustration of the problem.

We cannot use a command like this:

```
temp = 22.83
lcd.putstr(temp)
```

These i2c LCD displays cannot output a float value.

We need to cast the variable to string before outputting to the LCD display.

```
temp = 22.83
lcd.putstr("{}".format(temp))
```

Understanding the code:

This code has been simplified.

It is not reading from a temperature sensor and there is no while loop.

There is just a variable named **temp**.

The temp variable has the **float value 22.83** assigned to it.

Line 16 converts the float value to string before sending it to the LCD Display.

```
### libraries and variables for the LCD Display ###
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #this address varies
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

lcd.clear()
temp = 22.83
lcd.putstr("{}".format(temp))

```

Float value

Place holder

Replaces the placeholder with the value of the variable (converted to string)

Remember, you can use the programs we have been working on to find commands that help you achieve these objectives.

The test code on the next page brings together the libraries from the two programs we have used so far.

Hello World Program Code

```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #this address varies
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=m
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_RO
```

Temp to Shell program code

```
from machine import ADC
import time
# Internal temperature sensor is connected to ADC channel 4
temp_sensor = ADC(4)

while True:
    adc_value = temp_sensor.read_u16() # Read the raw ADC value
    voltage = adc_value * (3.3 / 65535.0)# Convert ADC value to voltage
    temp = 27 - (voltage - 0.706) / 0.001721 # Temp calc based on sensor characteristics
    print("Internal Temperature:  {:.2f} ".format (temp),"°C")
    time.sleep(1)
```

Use the test code on the next page as your starting point. Add new commands to output temp data to the LCD Display.

Can you develop the test code below further?

- Make it read from the temp sensor
- Use a while loop
- Output the value to the LCD Display as string

Test Code: includes libraries and variables for the temp sensor and LCD display

This test code does not yet read from the temp sensor or output results to the LCD display.

It does include the libraries so this program can read from the internal temperature sensor, and it can output data to the LCD display.

Your job is to develop the code under the ###COMMANDS### heading so the LCD display shows the temperature every second.

```
### libraries and variables for the LCD Display ###
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #this address varies
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

### libraries and variables for internal temp sensor ###
from machine import ADC
temp_sensor = ADC(4)

###COMMANDS###
```

Solution:

```
### libraries and variables for the LCD Display ###
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 39 #this address varies
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

### libraries and variables for internal temp sensor ###
from machine import ADC
temp_sensor = ADC(4)

###COMMANDS###
while True:
    #read the temp from the sensor#
    adc_value = temp_sensor.read_u16() # Read the raw ADC value
    voltage = adc_value * (3.3 / 65535.0)# Convert ADC value to voltage
    temp = 27 - (voltage - 0.706) / 0.001721 # Temperature calculation based on sensor characteristics
    #output the temp to the LCD display
    lcd.clear()
    lcd.move_to(0,0)
    lcd.putstr("{:.2f} degrees C".format(temp))
    time.sleep(1)
    lcd.clear()
```

Ok, now we have displayed a sensor value onto the LCD Display, let's move onto a new challenge. We can create our own custom characters and emoji's to output to the LCD display.

Custom Characters: Display shapes and emoji's

Instead of outputting text, we can output custom characters.

Try this code to see what I mean.

Test Code: Display one custom character

```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd
```

```
I2C_ADDR    = 39 # this value varies (scan to find yours)
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)


lcd.custom_char(0, bytearray([
0x00,
0x04,
0x0E,
0x04,
0x1F,
0x04,
0x0A,
0x11

]))


lcd.clear()

lcd.move_to(0,0)
lcd.putchar(chr(0))
time.sleep(10)

lcd.clear()
```

Remember, your LCD address is likely to be different to mine. You need to amend the test code command here:

```
7  I2C_ADDR      = 39
```

When you run the above test code should output a stick figure on row 0.

After 10 seconds the screen clears.

This line of code controls where the custom character is displayed.

```
30  lcd.move_to(0,0)
```

Cell Row

If we wanted the custom character to appear on the 2nd row of the LCD display, we would use this command.

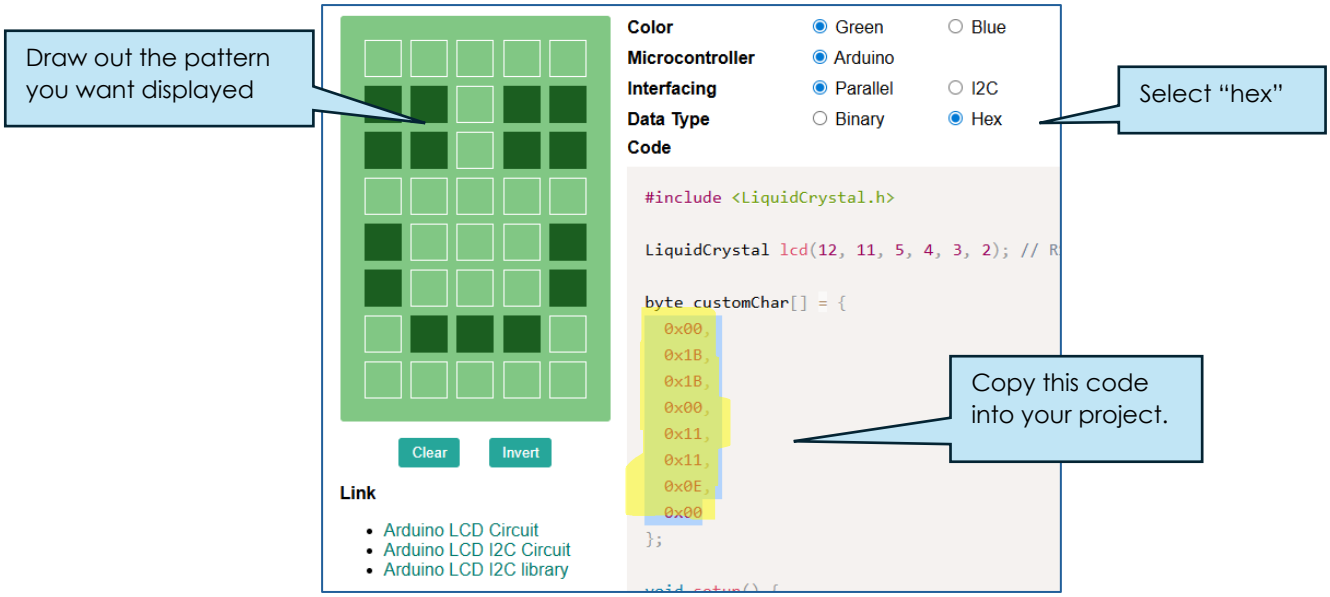
```
30  lcd.move_to(0,1)
```

Cell Row

Challenge – Display a new custom character

You can output your own custom characters using this website.

https://maxpromer.github.io/LCD-Character-Creator/

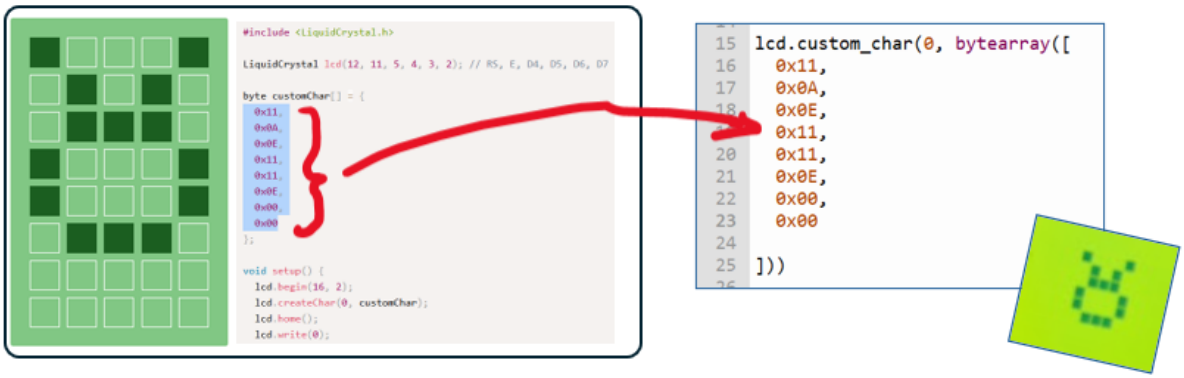


When you run Thonny, you should see the output has changed.

Use the maxpromer website to design your own custom character.

Replace the relevant lines of code in the test code.

Run your program to see your own custom character display in the LCD Screen.



If for some reason the maxpromer website is not working, search for "LCD Custom Character Generator".

Troubleshooting

- Have you set the LCD address on line 7 `I2C_ADDR = 39`
- Did you select the "Hex" button on the maxpromer website?

- Did you copy the correct lines of code from the maxpromer site, ensuring the code highlighted below is not affected?

```
15 lcd.custom_char(0, bytearray([
16   0x00,
17   0x1B,
18   0x1B,
19   0x00,
20   0x11,
21   0x11,
22   0x0E,
23   0x00
24
25 ]))
```

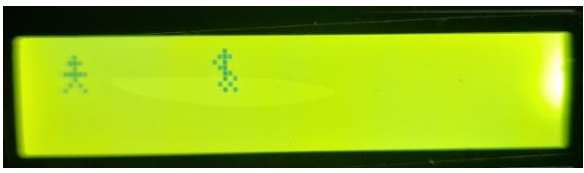
Solution:

```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd


I2C_ADDR     = 39 # this value varies (scan to find yours)
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)


lcd.custom_char(0, bytearray([
  0x11,
  0x0A,
  0x0E,
  0x11,
  0x11,
  0x0E,
  0x00,
  0x00

]))


lcd.clear()

lcd.move_to(0,0)
lcd.putchar(chr(0))
time.sleep(10)

lcd.clear()
```

Remember, you need to change the address value on line 7 to match the number you scanned for.

```
7  I2C_ADDR      = 39
```

Challenge - Display two custom characters on the 1st row.



We can use the maxpromer website to generate two custom characters.

Develop the test code to output two characters.

Make both of them output onto the first row of the LCD screen.

The code explained:

```
15 lcd.custom_char(0, bytearray([
16     0x00,
17     0x04,
18     0x0E,
19     0x04,
20     0x1F,
21     0x04,
22     0x0A,
23     0x11
24
25 ]))
26
27 lcd.custom_char(1, bytearray([
28     0x11,
29     0x0A,
30     0x0E,
31     0x11,
32     0x11,
33     0x0E,
34     0x00,
35     0x00
36
37 ]))
```

This is character 0

The code for the 2nd custom character must have a different number

```
39 lcd.clear()
40
41 lcd.move_to(0,0)
42 lcd.putchar(chr(0))
43 lcd.move_to(5,0)
44 lcd.putchar(chr(1))
45 time.sleep(10)
46
47 lcd.clear()
```

This makes the 1st character appear on the left side of the top row of the LCD.

This makes the 2nd character appear after the 5th cell of the top row of the LCD.

This outputs the 2nd custom character.

Solution (note – there are some deliberate bugs in this code!:

```
import time
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd


I2C_ADDR     = 39 # this value varies (scan to find yours)
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)


lcd.custom_char(0, bytearray([
    0x00,
    0x04,
    0x0E,
    0x04,
    0x1F,
```

```
    0x04,
    0x0A,
    0x11

]))

lcd.custom_char(1, bytearray([
    0x11,
    0x0A,
    0x0E,
    0x11,
    0x11,
    0x0E,
    0x00,
    0x00

]))

lcd.clear()

lcd.move_to(0,0)
lcd.putchar(chr(0))
lcd.move_to(2,0)
lcd.putchar(chr(0))
time.sleep(1)

lcd.clear()
```

How to Display several custom characters

This test code (from NerdCave) uses functions to output several custom characters.

It uses utime instead of time. This is almost identical.

Test Code:

```
import utime
from machine import I2C, Pin
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd


I2C_ADDR     = 39
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

def greeting():

    lcd.clear()
    lcd.move_to(0,0)
    lcd.putstr("Lets train")
    lcd.move_to(0,1)
    lcd.putstr("NINJA...")
    utime.sleep(2)
    lcd.clear()



def customcharacter():

  #character
  lcd.custom_char(0, bytearray([
  0x0E,
  0x0E,
  0x04,
  0x1F,
  0x04,
  0x0E,
  0x0A,
  0x0A

        ]))

    #character2
  lcd.custom_char(1, bytearray([
    0x1F,
  0x15,
  0x1F,
  0x1F,
  0x1F,
  0x0A,
  0x0A,
  0x1B

        ]))



  #smiley
  lcd.custom_char(2, bytearray([
  0x00,
  0x00,
  0x0A,
  0x00,
  0x15,
```

```
 0x11,
 0x0E,
 0x00

     ]))

 #heart
 lcd.custom_char(3, bytearray([
  0x00,
 0x00,
 0x0A,
 0x15,
 0x11,
 0x0A,
 0x04,
 0x00

     ]))

    #note
 lcd.custom_char(4, bytearray([
  0x01,
 0x03,
 0x05,
 0x09,
 0x09,
 0x0B,
 0x1B,
 0x18

     ]))
   #celcius
 lcd.custom_char(5, bytearray([
 0x07,
 0x05,
 0x07,
 0x00,
 0x00,
 0x00,
 0x00,
 0x00

     ]))



greeting()
customcharacter()
lcd.move_to(0,0)
lcd.putstr("Custom Character")
lcd.move_to(0,1)
lcd.putchar(chr(0))
lcd.move_to(4,1)
lcd.putchar(chr(1))
lcd.move_to(8,1)
lcd.putchar(chr(2))
lcd.move_to(12,1)
lcd.putchar(chr(3))
lcd.move_to(15,1)
lcd.putchar(chr(4))
```

Source: NerdCave YouTube channel and GitHub (example 1)

Sources and Further Reading:

NerdCave YouTube tutorial and GitHub resources:

https://www.youtube.com/watch?v=bXLgxEcT1QU

https://github.com/T-622/RPI-PICO-I2C-LCD

Freenove page inc. link to tutorial:

https://store.freenove.com/products/fnk0079

Scrolling Text:

https://randomnerdtutorials.com/micropython-i2c-lcd-esp32-esp8266/