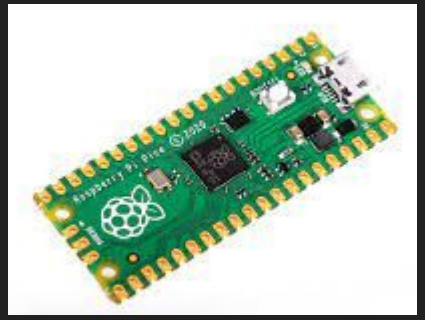
0. Meet the Parts



Raspberry Pi Pico

LED

Jumper Wires

Button

Buzzer

Breadboard

Potentiometer

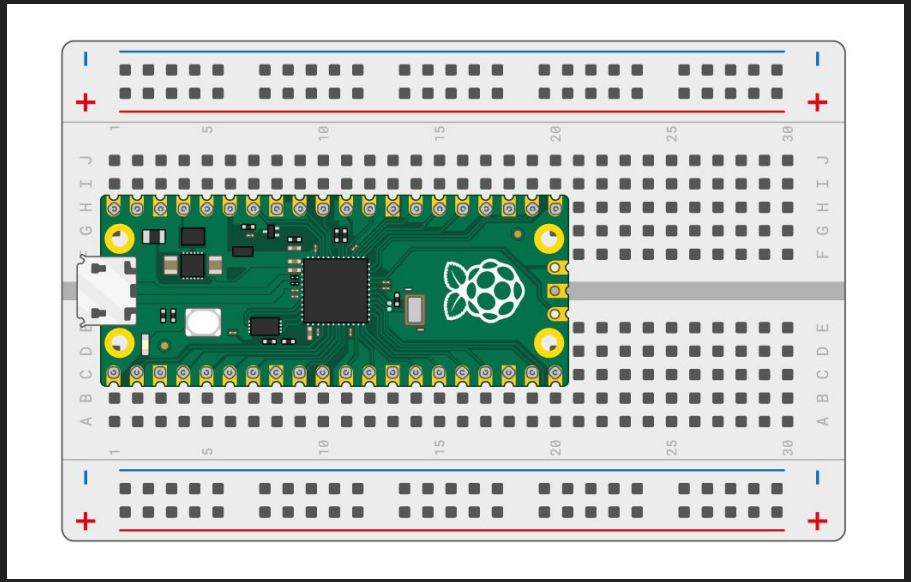
Resistors

OLED Display

Raspberry Pi Pico

- A Pi Pico is a microcontroller, which is essentially a programmable device for controlling other electronic devices
- A self contained system with its own processor, memory, IO peripherals, all integrated on one chip

1. Connecting Pico to your laptop



Pico & anThonny

2. Working with the Internal LED

```
1 from machine import Pin
2
3 led = Pin(25, Pin.OUT)
4
5 led.on()
```

```
1  from machine import Pin
2  from time import sleep
3
4  led = Pin(25, Pin.OUT)
5
6  led.on()
7
8  sleep(2)
9
10 led.off()
```

```
1  from machine import Pin
2  from time import sleep
3
4  led = Pin(25, Pin.OUT)
5
6
7  # make the LED blink!
8
9  for step in range(10):
10     led.on()
11     sleep(0.5) # on for 0.5s
12     led.off()
13     sleep(0.5) # off for 0.5s
```

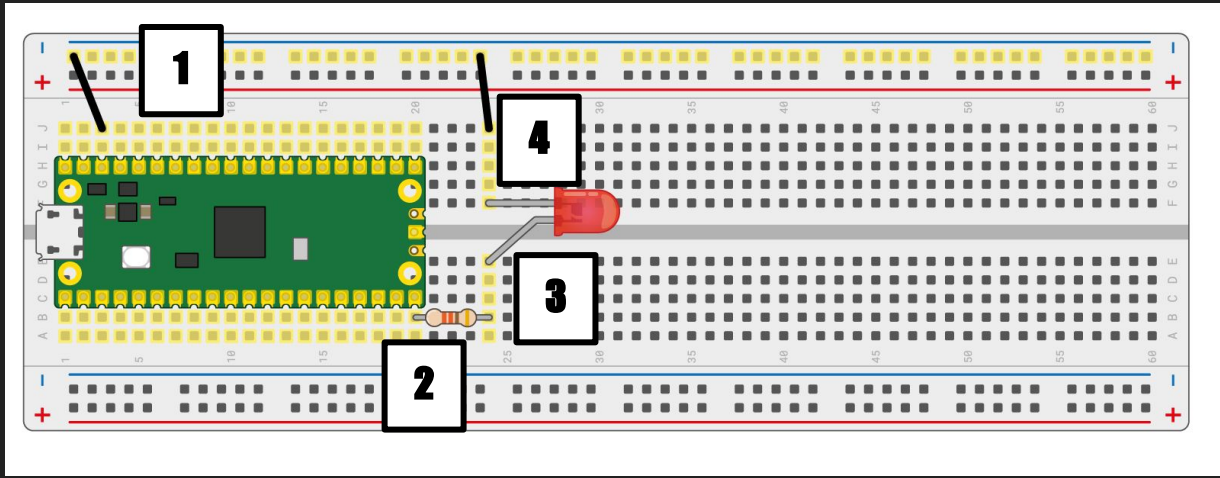
3. Working with an External LED

1. Wire the Ground Pin of Pico to the rightmost column on the breadboard (now the entire column is negative!)

2. Connect one end of Resistor to GP15

3. Connect second end of Resistor to same row as LED's positive (longer) terminal

4. Connect LED's negative terminal (shorter end) to the ground (negative column)




```
from machine import Pin
from time import sleep

led2 = Pin(15, Pin.OUT)
# connect your resistor to Pin 15

led2.on()

sleep(2)

led2.off()
```

Play around with the external LED!

- Make the external LED blink
- Make the internal LED and external LED alternate

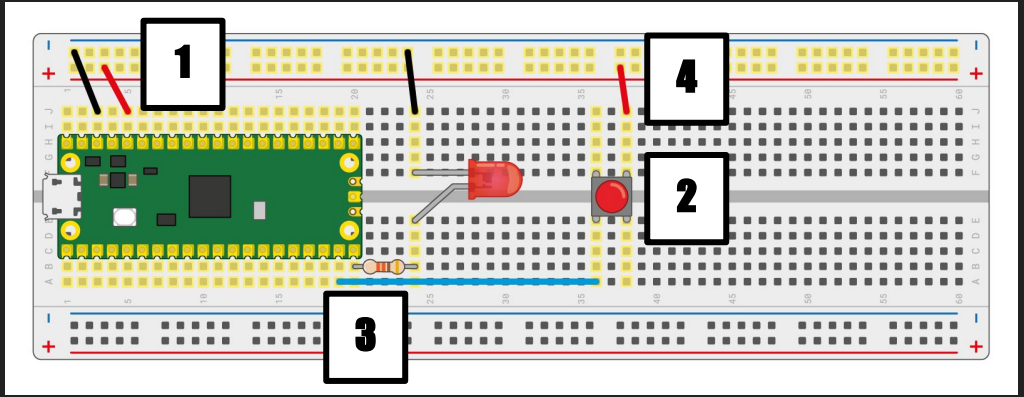
4. Connecting a Button!

1. Wire the 3v3 port to the inner column of the rightmost section of the breadboard (now the inner column is + and outer is -)

2. Plug in the button across the middle section of the board

3. Connect the top row of the button to Pin 14

4. Connect the bottom row of the button to the + column



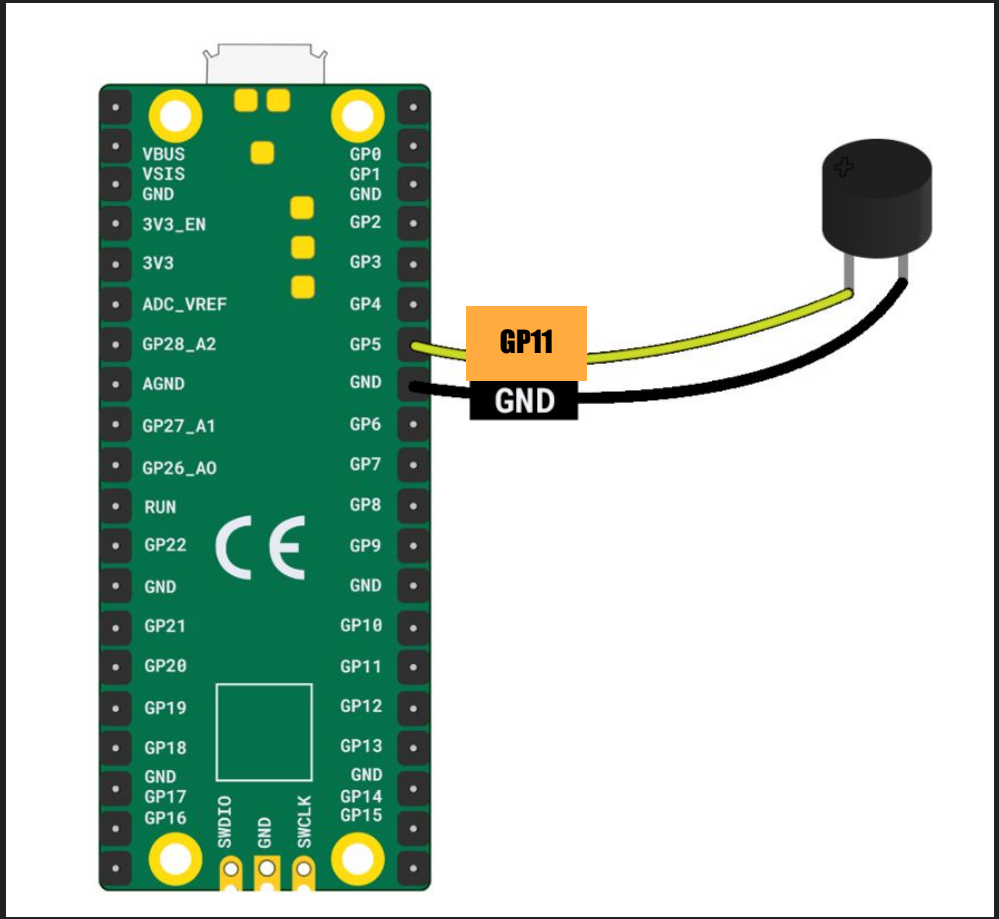

```
from machine import Pin
from time import sleep

led1 = Pin(25, Pin.OUT)
led2 = Pin(15, Pin.OUT)
# resistor connected to Pin15

button = Pin(14, Pin.IN, Pin.PULL_DOWN)
# button top connected to Pin14

while True:
    if button.value():
        print("LIGHTING UP!")
        led2.on()
    else:
        led2.off()
```

5. Connecting a Buzzer!

0. Keep everything from before intact!

1. Connect positive terminal of buzzer (longer end) to Pin 11

2. Connect negative terminal of buzzer to Ground (the column or a Ground pin)



```
led1 = Pin(25, Pin.OUT)
led2 = Pin(15, Pin.OUT)
# resistor connected to Pin15

button = Pin(14, Pin.IN, Pin.PULL_DOWN)
# button top connected to Pin14

buzzer = Pin(11, Pin.OUT)
# buzzer + connected to Pin11

print("Listening for Button Press ... ")
while True:
    if button.value():
        print("LIGHTING UP!")
        led1.on()
        led2.on()
        buzzer.on()
    else:
        led1.off()
        led2.off()
        buzzer.off()
```

Play around with all the components so far!

- Can you make the buzzer play different sounds? A song maybe?
- Add rave lighting (blinking LEDs) to your song?

6. Working with an OLED display

The I2C Protocol & SSD1306: Overview

The Hardware

GND - A power pin for a connection to the ground (negative terminal)

VDD - A power pin for a connection to the positive terminal

SCK - The OLED’s serial clock, deals with the timing information (in signals)

SDA - The OLED’s serial data, used to transfer data between the OLED and Pi Pico

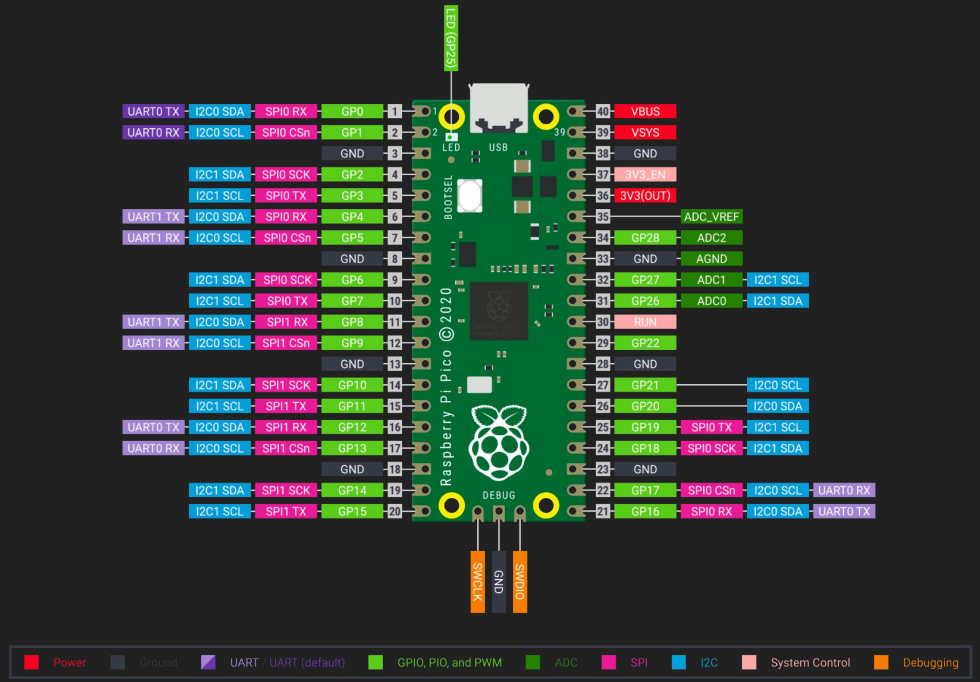
Relationship Problems: How do we communicate?

We use IIC! Or I2C/I²C for short.

What is it?

Inter-integrated circuit. Essentially, once the connections between the Pi Pico and OLED are made, I2C takes over and dictates what data is sent between the two devices, as well as how frequently.

Look at the Pi Pico diagram handed to you! How many I2C connections can be made?

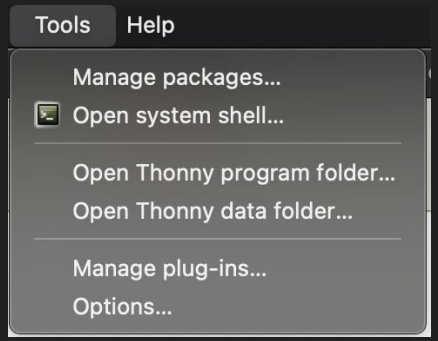


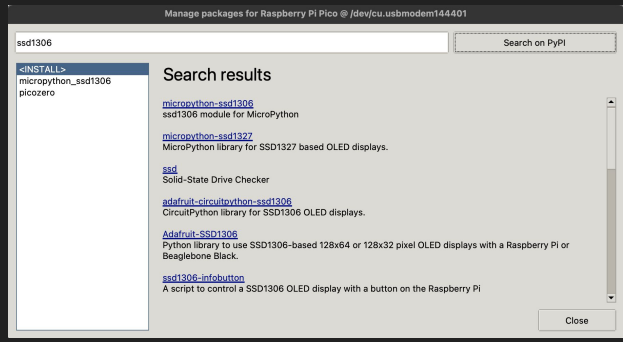
Some more interface details: SSD1306

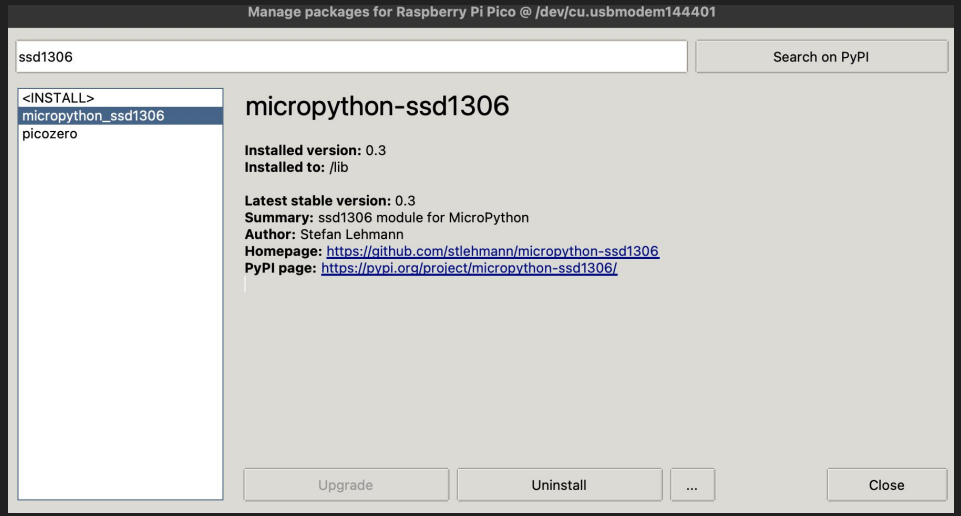
While I2C takes care of the communication issues, how do we actually display stuff?

The answer is SSD1306! It is a library that allows you to turn pixels on and off on the OLED, giving you the power to create any image you want on a 128x64 screen!

We first need to install this library on our Pi Pico

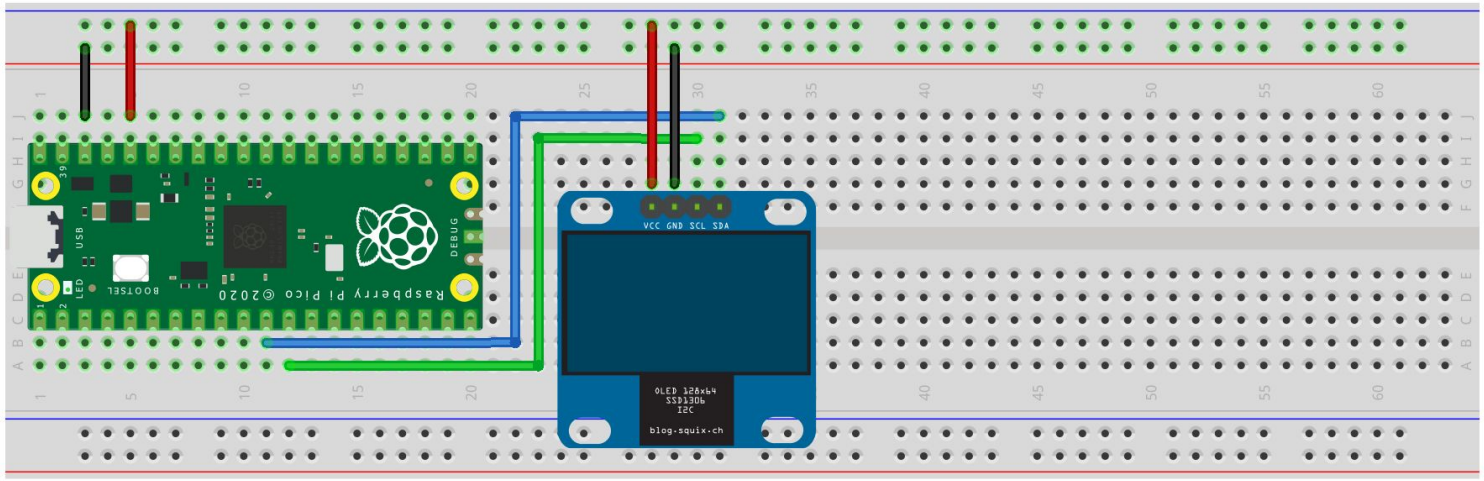






Making the connections

Raspberry Pi Pico + SSD1306 I2C OLED	
GP8 (pin 11)	SDA
GP9 (pin 12)	SCL
3V3	VCC
GND	GND



Let’s start drawing!

```
3  from machine import Pin, SoftI2C
4  import ssd1306
5  from time import sleep
6
7  # The specification for the OLED Display we are using
8  WIDTH = 128
9  HEIGHT = 64
10
11 connection = SoftI2C(sda = Pin(0), scl = Pin(1))
12 oled = ssd1306_SSD1306_I2C(WIDTH, HEIGHT, connection)
```

```
14 # Clears the screen
15 oled.fill(0)
16 oled.show()
17
18 sleep(1)
19
20 oled.fill(1)
21 oled.show()
```

Playing around a bit more!

```
23  # Clears the screen, and turns on the pixel at (32, 32)
24  oled.fill(0)
25  oled.pixel(32, 32, 1)
26  oled.show()
27
28  # Draws a line from (12, 13) to (28, 55)
29  oled.fill(0)
30  oled.line(12, 13, 28, 55, 1)
31  oled.show()
32
33  # Draws a rectangle at (28, 12) of width 10 and height 15
34  oled.fill(0)
35  oled.rect(28, 12, 10, 15, 1) # or oled.fill_rect(28, 12, 10, 15, 1)
36  oled.show()
```

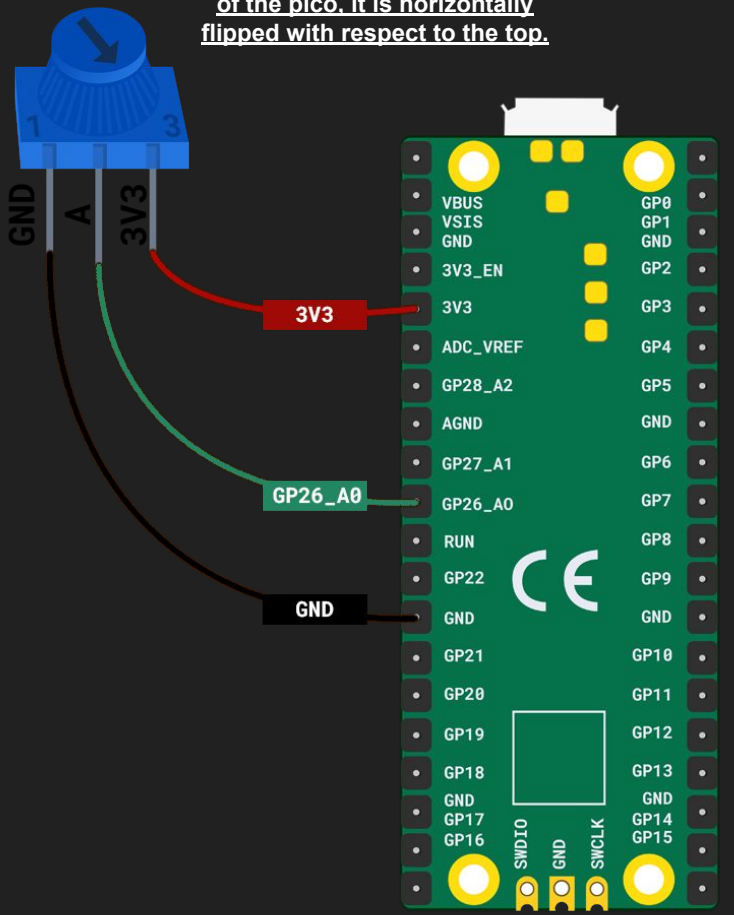
8. Working with a potentiometer

A potentiometer has three pins:

1. Ground/Power
2. Analog Signal Out
3. Power/Ground

Use F-M jumper wires to connect the potentiometer to the breadboard, and use the female ends for the legs of the potentiometer

```python
from picozero import Pot # Pot is short for Potentiometer
from time import sleep

dial = Pot(0) # Connected to pin A0 (GP_26)

while True:
    print(dial.value)
    sleep(0.1) # slow down the output
```



Inspecting the Range

```
from picozero import Pot
from time import sleep

dial = Pot(2)
values = []

for i in range(100):
    reading = dial.value
    values.append(reading)
    sleep(0.05)

print(min(values), max(values))
```


```
1  from picozero import Pot
2  from time import sleep
3
4  dial = Pot(2)
5  values = []
6
7  for i in range(100):
8      reading = dial.value
9      values.append(reading)
10     sleep(0.05)
11
12 print(min(values), max(values))
```

9. Pong

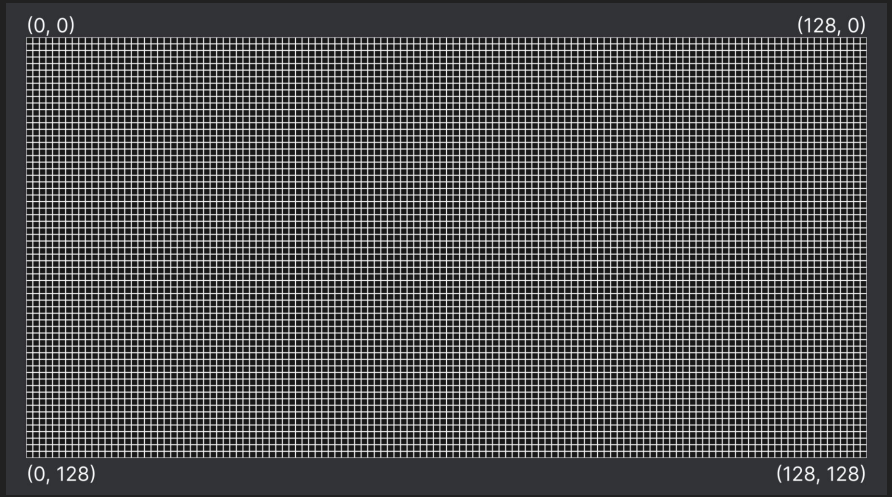
Exercise: Helper Function Setup

```
38  # return the sign of x (+1 or -1)
39  def sgn(x):
40      pass
41
42  # the input thing is in the range [minin, maxin], the output should be in
43  # the range [minout, maxout] with linear scaling
44  def remap(thing, minin, maxin, minout, maxout):
45      pass
46
47  # the input thing is in any range, the output should be thing clamped (truncated)
48  # to the range [minout, maxout]
49  def clamp(thing, minout, maxout):
50      pass
51
52  # assume that the I2C SDA connection for an OLED is present at pin GP(sda)
53  # and the SCL connection is present at the GP(sda + 1), return an SSD1306 object
54  def oled_connect(sda):
55      pass
```

Exercise: Potentiometer

Complete the following function using the data from our earlier inspection of the potentiometer's range to get the y position of a pixel based on the potentiometer's input.

The pixel should be able to "move" or occupy every vertical position possible on the screen (and ideally not leave the screen).




```
57  # the pot input is a potentiometer object. based on the coordinate system,
58  # think about the range of this function's returned value
59  def get_y_position(pot):
60      pass
```

Controlling a Pixel: Definitions

```
62  WIDTH = 128
63  HEIGHT = 64
64
65  FRAME_RATE = 60
66  PADDLE_DIMS = (8, 16)
67  BALL_DIMS = (5, 5)
68
69  OLED_ATTACH = 0
70  POT_IN = 0
71
72  # Replace these values with your potentiometer's range,
73  # and use them in your get_y_position implementation
74  POTENTIO_MIN = 0
75  POTENTIO_MAX = 1
```

(put these at the top of your file, just after the imports)

Controlling a Pixel

Upon running this code, if everything has been implemented correctly, you should see a pixel on the OLED's first column, which you can move using the potentiometer.

Next, we will extend this single pixel to be our paddle.

Exercise: Have your get_y_position implementation use the PADDLE_DIMS definition, where the first element is the width of a paddle and the second element is the height.

```
77  oled = oled_connect(OLED_ATTACH)
78  pot = Pot(POT_IN)
79
80  # this is our game loop
81  while True:
82      oled.fill(0)
83      pos = get_y_position(pot)
84
85      oled.pixel(0, pos, 1)
86      oled.show()
87      sleep(1 / FRAME_RATE)
```

Paddles

```
89  # a paddle can be represented as the x & y position of
90  # its top-left pixel
91  paddle_l = (0, get_y_position(pot))
92
93  # what should the coordinates be for the right paddle, if we
94  # want it to be placed at the center of the last column?
95  paddle_r = (???, ???)
96
97  # ...game loop
```

```
 99  # the oled input is the ssd1306 object, and the paddle is
100  # the (x, y) tuple containing the paddle's position
101  # use the PADDLE_DIMS definition here
102  def draw_paddle(oled, paddle):
103      pass
```

Replace the pixel drawing code in the game loop to draw both paddles using this function. Make sure to update paddle_l before drawing it

Introducing the Ball

- The ball is the most complex object in the game
- It has a velocity, is expected to collide (and bounce off of) walls and paddles, and detect when a point is scored.
- We represent the ball as (x, y, xv, yv), where xv and yv are x-velocity and y-velocity respectively
- We create a reset_ball helper function, used to initialise the ball, and to reset it whenever a point is scored.

```
105 def reset_ball():
106     yv = random.randint(-3, 3)
107     while yv == 0:
108         yv = random.randint(-3, 3)
109 
110     # The ball is placed at the center of the screen
111     return ((WIDTH - BALL_DIMS[0]) // 2, (HEIGHT - BALL_DIMS[1]) // 2, 2, yv)
112 
113 # write a function to draw the input ball,
114 # use BALL_DIMS
115 def draw_ball(ball):
116     pass
117 
118 # we will fill this in later
119 def update_ball(ball, paddles):
120     pass
121 
122 # ...
123 
124 ball = reset_ball()
125 
126 # ... game loop
```

Updated Game Loop

```
128 # ...
129 while True:
130     oled.fill(0)
131     pos = get_y_position(pot)
132     paddle_l = (0, pos)
133 
134     draw_paddle(paddle_l)
135     draw_paddle(paddle_r)
136 
137     update_ball(ball, [paddle_l, paddle_r])
138     draw_ball(ball)
139 
140     oled.show()
141     sleep(1 / FRAME_RATE)
```

update_ball: Wall Collisions

```
143 # definitions...
144
145 score = (0, 0)
146
147 # helpers...
148
149 def update_ball(ball, paddles):
150     global score
151     x, y, xv, yv = ball
152
153     # A: paddle collisions (later)
154
155     # B: horizontal wall collisions
156     reset = x < 0 or x >= WIDTH - ???
157
158     if x < 0:
159         score = (score[0], score[1] + 1)
160
161     if x >= WIDTH - ???:
162         score = (score[0] + 1, score[1])
163
164     if reset:
165         return reset_ball()
166
167     # C: vertical wall bounces
168     if y < 0:
169         y = 0
170         yv *= -1
171         return (x, y, xv, yv)
172
173     if y > HEIGHT - BALL_DIMS[1]:
174         y = HEIGHT - BALL_DIMS[1]
175         yv *= -1
176         return (x, y, xv, yv)
177
178     # D: no collisions
179     return (round(x + xv), round(y + yv), xv, yv)
```

rect_intersection

Returns true if rectangles A and B, expressed as (x, y, w, h), intersect. Takes in parameters x_off and y_off to offset rectangle A before comparing.

```
56  def rect_intersection(A, B, x_off = 0, y_off = 0):
57      # no intersection if any of the widths or heights are 0
58      if A[2] == 0 or A[3] == 0 or B[2] == 0 or B[3] == 0:
59          return False
60
61      # if A is completely to the right of B or B is completely to the right of A
62      if A[0] + x_off > B[0] + B[2] or B[0] > A[0] + x_off + A[2]:
63          return False
64
65      # if A is completely below B or B is completely below A
66      if A[1] + y_off > B[1] + B[3] or B[1] > A[1] + y_off + A[3]:
67          return False
68
69      return True
```

Paddle Collision

```
197 def update_ball(ball, paddles):
198     global score
199     x, y, xv, yv = ball
200 
201     # A: paddle collisions
202     ball_bb = (x, y, BALL_DIMS[0], BALL_DIMS[1])
203 
204     for paddle in paddles:
205         paddle_collision = False
206         test_bb = (paddle[0], paddle[1], PADDLE_DIMS[0], PADDLE_DIMS[1])
207 
208         if yv != 0 and rect_intersection(ball_bb, test_bb, 0, yv):
209             paddle_collision = True
210             while not rect_intersection(ball_bb, test_bb, 0, sgn(yv)):
211                 y += sgn(yv)
212                 ball_bb = (x, y, BALL_DIMS[0], BALL_DIMS[1])
213             yv *= -1
214 
215         if not paddle_collision and xv != 0 and rect_intersection(ball_bb, test_bb, xv):
216             paddle_collision = True
217             while not rect_intersection(ball_bb, test_bb, sgn(xv)):
218                 x += sgn(hv)
219                 ball_bb = (x, y, BALL_DIMS[0], BALL_DIMS[1])
220             xv *= -1
221 
222         if paddle_collision:
223             return (x, y, xv, yv)
224 
225     # ...
```

Line 218 has an error, hv should be xv

Exercise: Score Display

Write a function to display the score, and add it to the game loop.

10. Making up an Opponent (Pong AI)