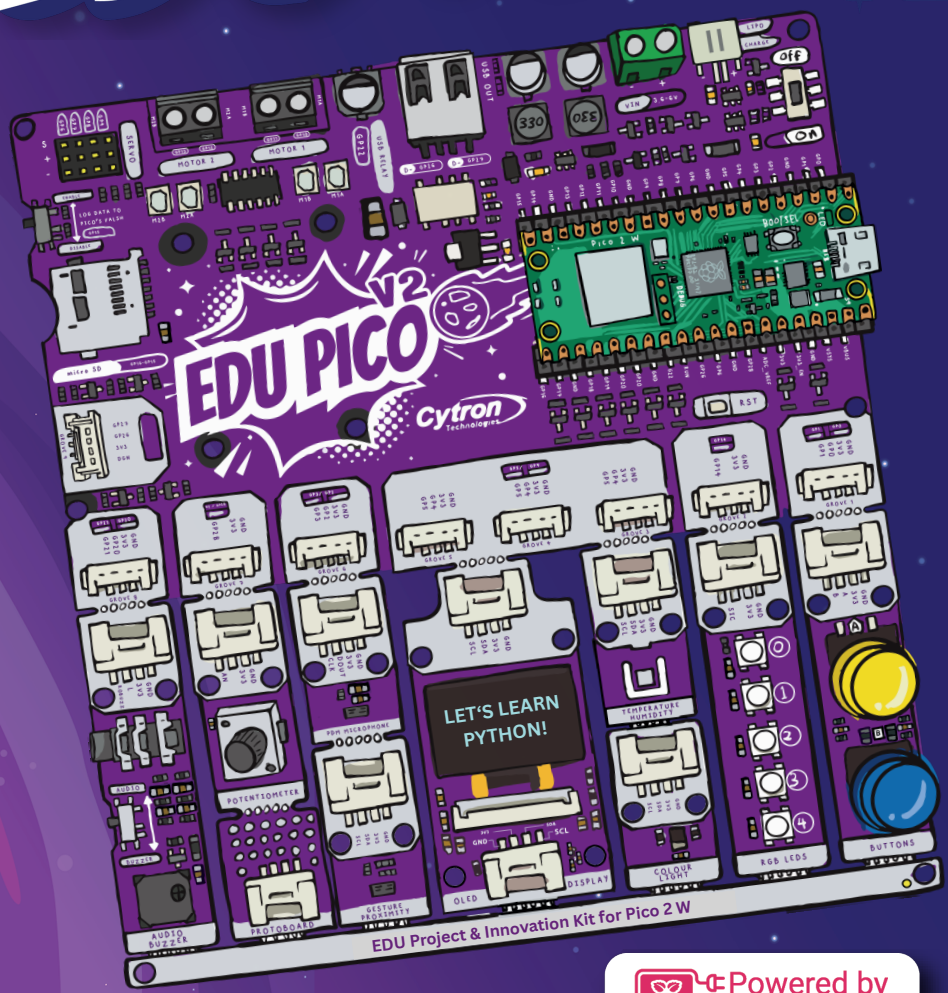
Cytron Technologies

Explore STEM & Coding with EDU PICO V2



Powered by Raspberry Pi

Author's Note

It is with great excitement that I introduce this book, **Explore STEM and Coding with EDU PICO 2**. This updated guide is designed to accompany the EDU PICO 2 kit, which is also powered by the Raspberry Pi Pico 2 W, and proudly brought to you by Cytron Technologies.

A special acknowledgement goes to SC Lim, our Project Manager, for his steady guidance and leadership in ensuring the successful development of both the EDU PICO 2 kit and this second edition of the guide.

To Salam & Noqman, the Designers of EDU PICO 2, for to shape its technical foundation, your expertise has made EDU PICO 2 a powerful and accessible tool for STEM education. And to Suhana, our Illustrator, for once again bringing the content to life through visuals that make complex ideas easier and more enjoyable to understand.

I would like to extend my sincere gratitude to our dedicated reviewers — Livendran, Anna, Priya, and Khairul for their valuable insights and detailed feedback throughout this edition's development. Your input has helped refine and strengthen the learning experience for our readers.

To our amazing trainees and team members behind the scenes, thank you for your contributions, time, and enthusiasm in making EDU PICO 2 possible.

Finally, this book is dedicated to all students and educators who will embark on their learning journey with EDU PICO 2. May it spark curiosity, creativity, and a lifelong love for learning through hands-on exploration of programming and electronics.

With the new improvement and integration of AI prompts, I can't wait to see the amazing projects you'll innovate with the EDU PICO 2!

Happy Learning!
Adrian Teo

Explore STEM and Coding with EDU PICO 2 Project & Innovation Kit

Second Printing, June 2026

Published by Cytron Technologies

Copyright © 2026 Cytron Technologies

All rights reserved. No part of this book may be reproduced or distributed in any manner without the prior written permission of the copyright owner.

The content of this publication has not been approved by the United Nations and does not reflect the views of the United Nations or its officials or Member States. For more information about the United Nations Sustainable Development Goals, visit here:
https://www.un.org/sustainabledevelopment/

To request permissions, contact the publisher at education@cytron.io.

Declaration

The Author and Publisher have made every effort to ensure the accuracy of the information in this book. However, they do not assume any liability and hereby disclaim responsibility for any loss or damage resulting from errors or omissions in this book, whether arising from negligence, accident, or any other cause.

Published by:

Cytron Technologies Sdn. Bhd.
1, Lorong Industri Impian 1,
Taman Industri Impian,
14000 Bukit Mertajam,
Penang, Malaysia.

Tel: +604-5480668
Fax: +604-5480669

www.cytron.io

Printed in Malaysia.

Contents

Hello EDU PICO 2

Chapter 1: Programming with CircuitPython ..... 10 - 26
Text-based Coding with Thonny and CircuitPython.

Chapter 2: Water Drinking Reminder ..... 27 - 45
Buttons and Buzzer.

Chapter 3: Gesture Reaction Game ..... 46 - 67
OLED and Gesture Sensor.

Chapter 4: Colour Detection Game ..... 68 - 86
RGB LEDs and Colour Sensor.

Chapter 5: Automated Waste Bin ..... 87 - 106
Servo Motor and Proximity Sensor.

Chapter 6: Noise Pollution Monitoring ..... 107 - 127
PDM Sound Sensor and Potentiometer.

Chapter 7: Smart Classroom ..... 128 - 150
USB Relay and DC Motor.

Chapter 8: Climate Control Greenhouse ..... 151 - 178
Light Sensor and Temperature Humidity Sensor.

Chapter Summary

The EDU PICO 2 is an all-in-one Raspberry Pi Pico 2 W learning kit, specially designed for beginners who wish to start in text-based Python programming using hardware.

This book offers 8 chapters with 14 activities, covering all 14 modules and functionalities of the EDU PICO 2. Aside from that, the guidebook also features 7 captivating SDG related projects that deepens your knowledge in combining modules in a single project.

Chapter 1: Programming with CircuitPython

In this chapter, we learn to:

- install Thonny IDE.
- program our first CircuitPython program using Thonny IDE.
- download program to EDU PICO 2.
- save, open, and edit Python .py files in EDU PICO 2.
- import CircuitPython libraries.

Activities: CircuitPython syntax - Indentation, variables, data types, casting, comments, operators, if.. else (conditions), for loops, while loops, functions, and string format.

Chapter 2: Water Drinking Reminder (Button and Buzzer)

In this chapter, we learn to:

- use input buttons to interact with Thonny's console.
- use a piezo buzzer to produce sound.

Introduction: Button and Buzzer.
AI Task #1: Print "A" when button A is pressed and "B" when button B is pressed.
AI Task #2: Make the buzzer play a short melody of "Twinkle Twinkle Little Star".

Activities: Build a water drinking reminder.
Challenge: Program drinking reminder to accept user input for the duration.

Chapter 3: Gesture Reaction Game (OLED and Gesture Sensor)

In this chapter, we learn to:

- display text on the OLED display.
- use gesture sensor as input.

Introduction: OLED and Gesture Sensor.
AI Task #1: Sweep the text "Hello EDU PICO" across the screen from left to right.
AI Task #2: Use hand gestures to control the text movement on the OLED display.

Activities: Build a hearing gesture reaction game with the gesture sensor and buzzer.
Challenge: Program a tone testing feature for better gaming experience.

Chapter 4: Colour Detection Game (RGB LEDs and Colour Sensor)

In this chapter, we learn to:

- program EDU PICO 2 to light up RGB LEDs in different colours.
- read colour data with colour sensor.

Introduction: RGB LEDs and Colour Sensor.
AI Task #1: Make the RGB LEDs show a running rainbow colour effect.
AI Task #2: Use the colour sensor to detect red, green, or blue and show that colour on the RGB LED module.

Activities: Build a colour detection game.
Challenge: Improve the game by programming a colour hint indicator using a single RGB LEDs.

Chapter 5: Automated Waste Bin (Servo Motor and Proximity Sensor)

In this chapter, we learn to:

- control a servo motor using pulse width modulation (PWM).
- read distance value using the PAJ7620 proximity sensor.
- read data with Thonny's plotter.
- construct a Trashbot smart bin with card box accessories.

Introduction: Servo Motor and Proximity Sensor.
AI Task #1: Use Button A to trigger the servo motor to move from 0° to 180°.
AI Task #2: Use the proximity sensor to trigger and control the buzzer's beeping interval.

Activities: Build an Automated Waste Bin (Trashbot Smart Bin).
Challenge: Upgrade Trashbot to include light and sound, making it more interactive.

Chapter 6: Noise Pollution Monitoring System (Potentiometer and Sound Sensor)

In this chapter, we learn to:

- program EDU PICO 2 to read analog values from a potentiometer.
- measure noise in dB with the PDM sound sensor.
- construct a physical noise level meter with a card accessory.

Introduction: Potentiometer and PDM Sound Sensor.
AI Task #1: Use the potentiometer to control the brightness of the RGB LED strip.
AI Task #2: Use the sound sensor to turn the LED on and off when the sound level exceeds a specific threshold.

Activities: Build a room environment noise pollution monitoring system.
Challenge: Program a servo motor to serve as a sound level meter.

Chapter 7: Smart Classroom (DC Motor and Relay)

In this chapter, we learn to:

- program EDU PICO 2 to control a DC motor – spinning direction and speed control.
- turn ON and OFF a USB switch relay.

Introduction: DC Motor and USB Relay.
AI Task #1: Use the potentiometer to control the speed and direction of the DC motor.
AI Task #2: Use Button A to toggle the relay ON and OFF (like a light switch).

Activities: Build a smart classroom with a gesture sensor, USB relay, DC motor, and OLED.

Chapter 8: Climate Control Greenhouse (Light and Humidity Temperature Sensor)

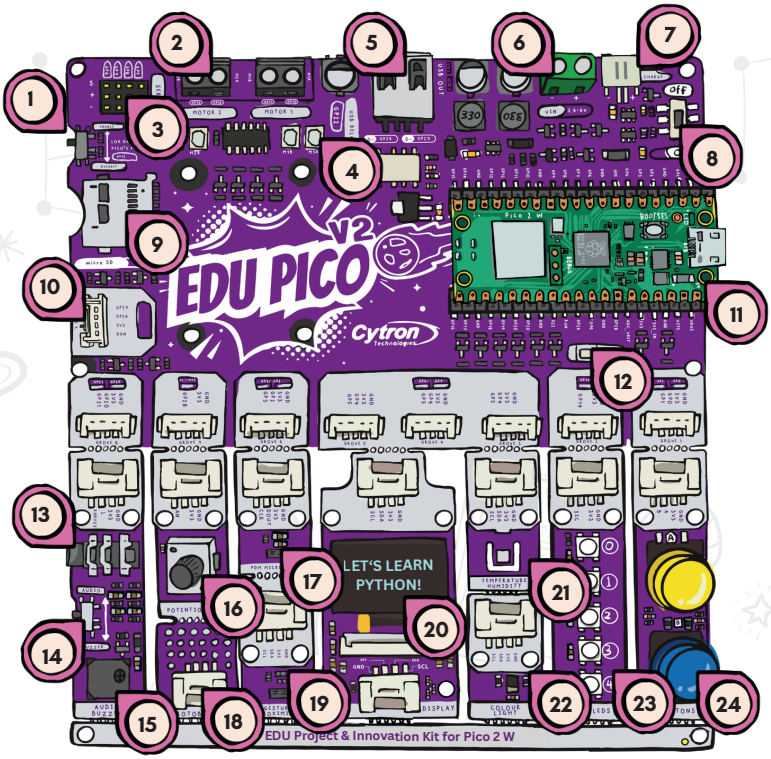
In this chapter, we learn to:

- program light sensor to measure ambient brightness.
- program AHT20 sensor for humidity and temperature measurement.
- program EDU PICO 2's Raspberry Pi Pico 2 W into a WiFi access point

Introduction: Light Sensor, Humidity and Temperature Sensor.
AI Task #1: Use the light sensor to automatically turn the RGB LED ON and OFF.
AI Task #2: Display temperature and humidity live readings on the OLED display.

Activities: Build a climate control greenhouse using temperature and humidity sensor, light sensor, DC and servo motor, RGB LEDs, and OLED.
Bonus: Control the USB relay through a webpage using the Raspberry Pi Pico 2 W acting as a WiFi Access Point (AP).

HELLO EDU PICO 2



1. Log Data to Pico's Flash Switch
2. DC Motor Terminals
3. Servo Ports
4. DC Motor Test Buttons
5. USB Relay
6. Vin Terminal
7. LiPo Battery Connector
8. On/Off Switch
9. Micro SD Card Socket
10. Additional Grove Port
11. Raspberry Pi Pico 2 W
12. Reset Button
13. Stereo Audio Jack
14. Audio/Buzzer Switch
15. Piezo Buzzer
16. Potentiometer
17. PDM Microphone
18. Protoboard
19. Gesture & Proximity Sensor (PAJ7620)
20. OLED Display (SSD1315)
21. Temperature & Humidity Sensor (AHT20)
22. Colour & Light Sensor (OPT4048)
23. RGB LEDs
24. Buttons

CHAPTER 1

Programming with CircuitPython

Start Here - Thonny IDE

Introduction to CircuitPython

Hello EDU PICO

CircuitPython Syntax

- Indentation
- Variables
- Data Types
- Casting
- Comments
- Operators
- If.. Else (Conditions)
- For Loops
- While Loops
- Functions
- String Format



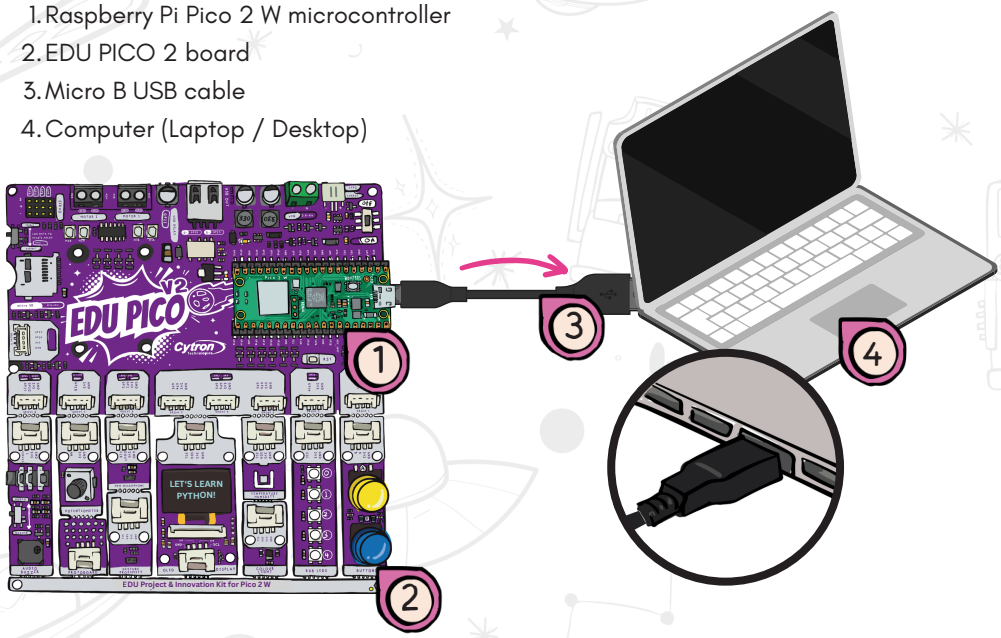
Start Here - Thonny IDE

EDU PICO 2 is an all-in-one learning tool that allows you to practice using electronics and CircuitPython programming using the Raspberry Pi Pico 2 W microcontroller. It's a special board with all pre-connected sensors to help you learn more seamlessly. This means that you can start experimenting and building projects immediately without knowing a lot about electronics.

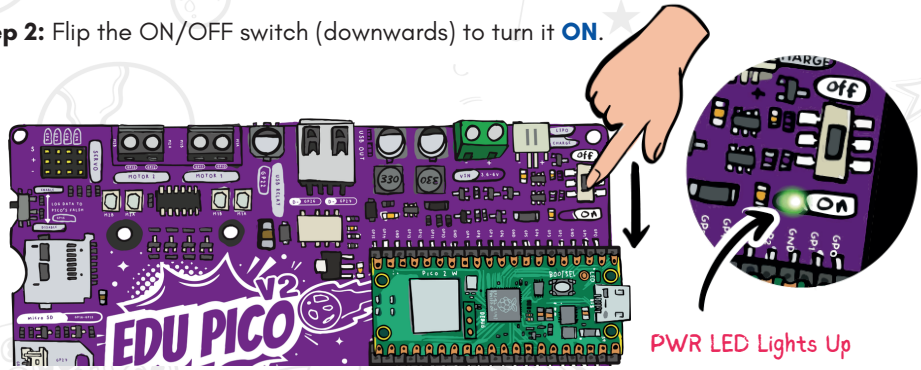
What You Need to Start?

Step 1: Connect the EDU PICO 2 to your computer's USB port.

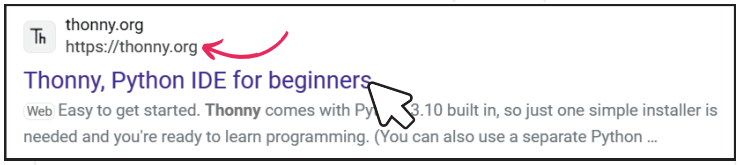
1. Raspberry Pi Pico 2 W microcontroller
2. EDU PICO 2 board
3. Micro B USB cable
4. Computer (Laptop / Desktop)

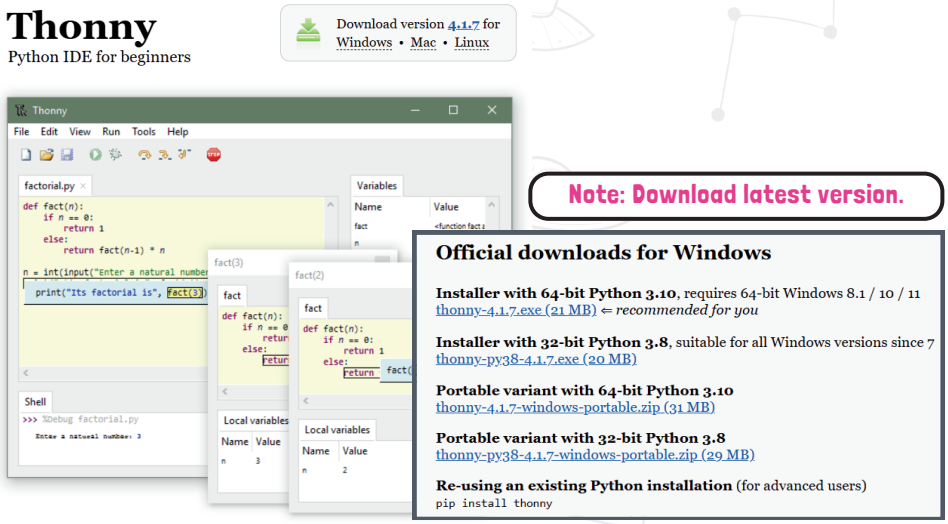


Step 2: Flip the ON/OFF switch (downwards) to turn it **ON**.



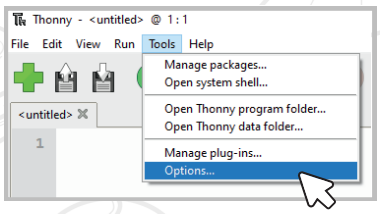
Step 3: Download and install **Thonny IDE** on your computer.





Step 4: Let's configure Thonny IDE's theme and font so that it matches the theme of the guidebook. This will make sure you can go through the guidebook smoothly later.

Select **Tools > Options > Theme & Font** as shown in the figure below. Adjust the IO font, UI theme, Syntax theme, and Editor font accordingly.

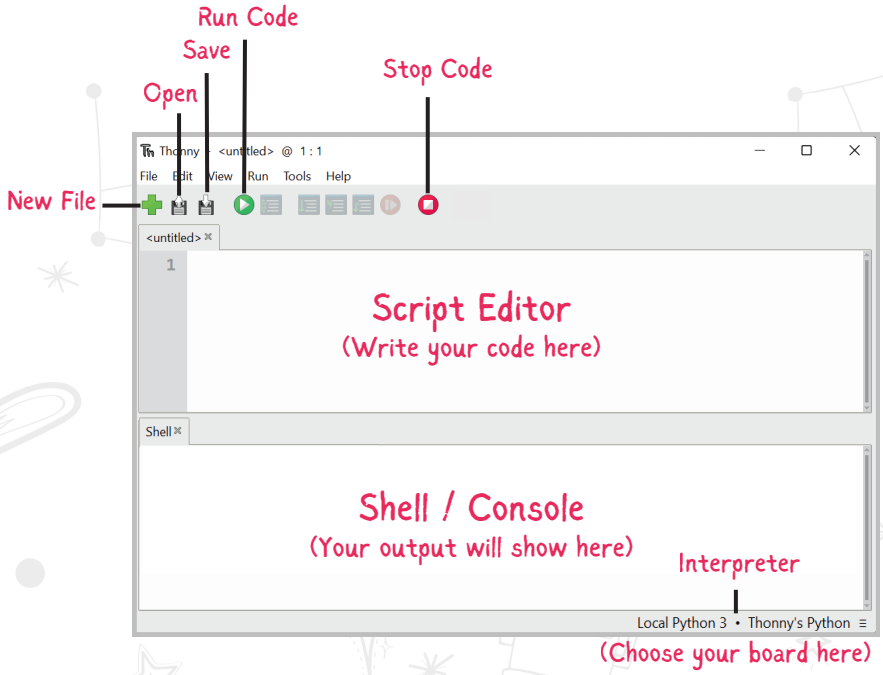




Step 5: Select **OK**, and restart Thonny to make sure the style elements take effect.

Now you're all set for your first program!

Thonny Integrated Development Environment (IDE)



Why Thonny IDE?

Thonny IDE runs on systems including Windows, MacOS, and Linux. It provides a dedicated mode meant for MicroPython, which includes CircuitPython that we will be using throughout this module.

Thonny IDE is also created specifically for beginners where its interface has intentionally removed extra features that might be overwhelming or distracting to new learners.

Syntax errors are a common issue for beginners, which is why Thonny IDE makes it easy for learners to highlight and identify these errors easily.

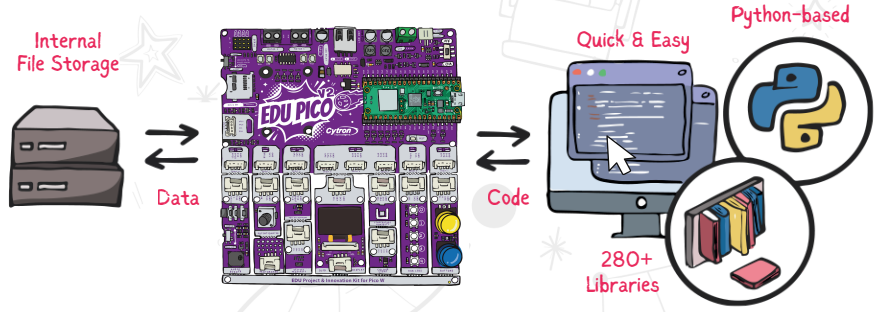
Lastly, Thonny IDE was originally designed for teaching Python and emphasizes learning and understanding code. Hence why it is an excellent choice for educators and students to practice programming microcontrollers with Python.

Introduction to CircuitPython

CircuitPython is a programming language based on Python. Specifically designed to simplify for experimenting and learning to code with microcontroller boards. In this module, we will learn CircuitPython programming with EDU PICO 2 and Raspberry Pi Pico 2 W.

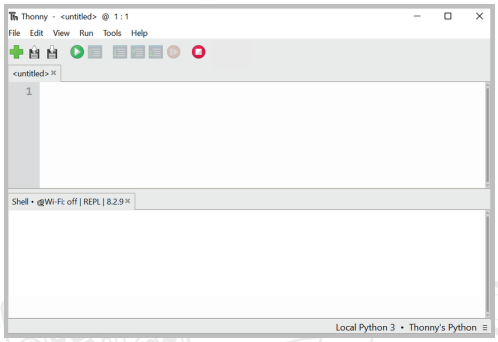
Why CircuitPython?

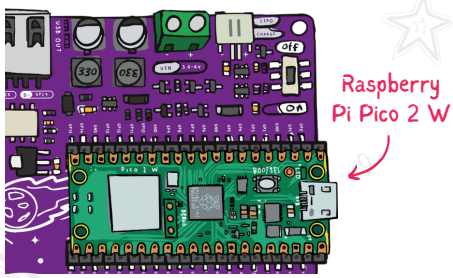
- **Easy & Beginner Friendly**: Create, edit, save, and run your code without compiling, downloading, or uploading needed. Perfect for learning text-based programming.
- **File Storage**: Internal storage designed for data-logging, playing audio clips, and file interaction. It also allows you to edit your code anytime since it is stored on the disk drive.
- **Strong Hardware Support**: Built-in features like audio I/O, digital I/O pins, hardware buses (I2C, SPI, UART), and supports over 280+ libraries all written in Python.
- **Python**: CircuitPython is based on Python, the fastest-growing programming language. It simply adds hardware support to all of Python's amazing features.



Preparing EDU PICO 2 CircuitPython Firmware

Step 1: Launch Thonny IDE and make sure your computer has internet connection.

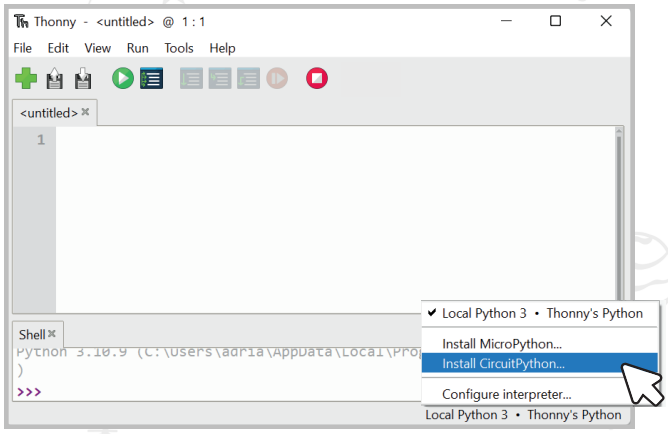




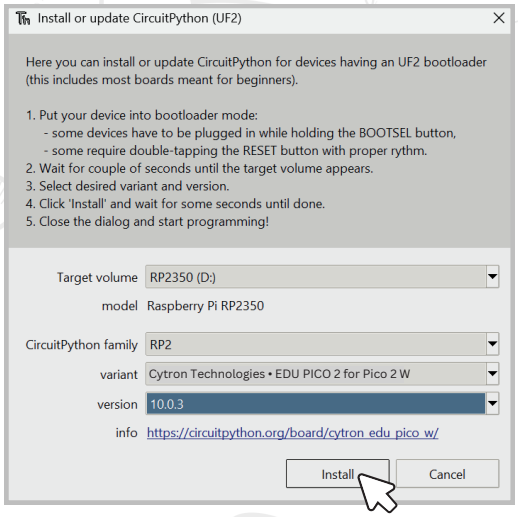
Step 2: Press and hold the reset (RST) and BOOTSEL buttons, then release the RST button, but continue to hold the **BOOTSEL button** until the **RP2350** drive appears.



Step 3: Click on the bar at the bottom right corner and select **Install CircuitPython** from the drop-down menu.

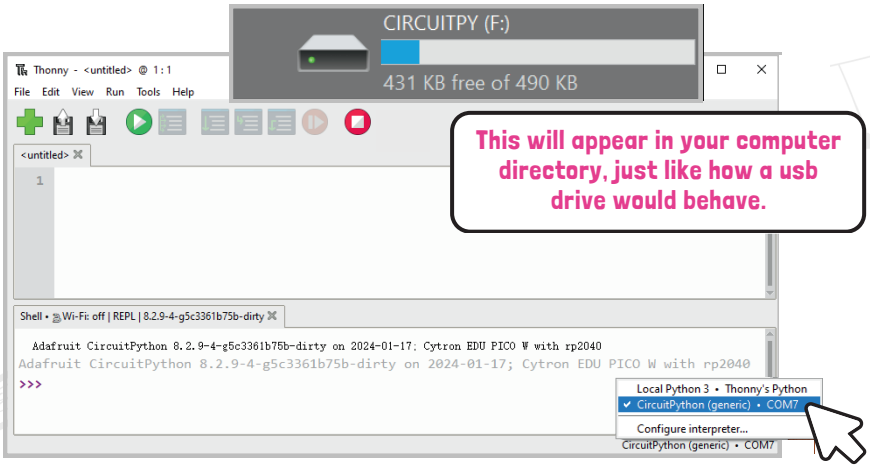


Step 4: Select **RP2350** for the target volume, and **Cytron Technologies - EDU PICO 2 for Pico 2 W** for the variant. Then select the latest stable version and Click **Install**.



Step 5: Wait for the download and installation to be done. Close all the pop-up windows.

Step 6: After successful installation, the **CircuitPython** drive will appear on your computer. Click the bottom right corner of Thonny and select the **CircuitPython** option as shown.



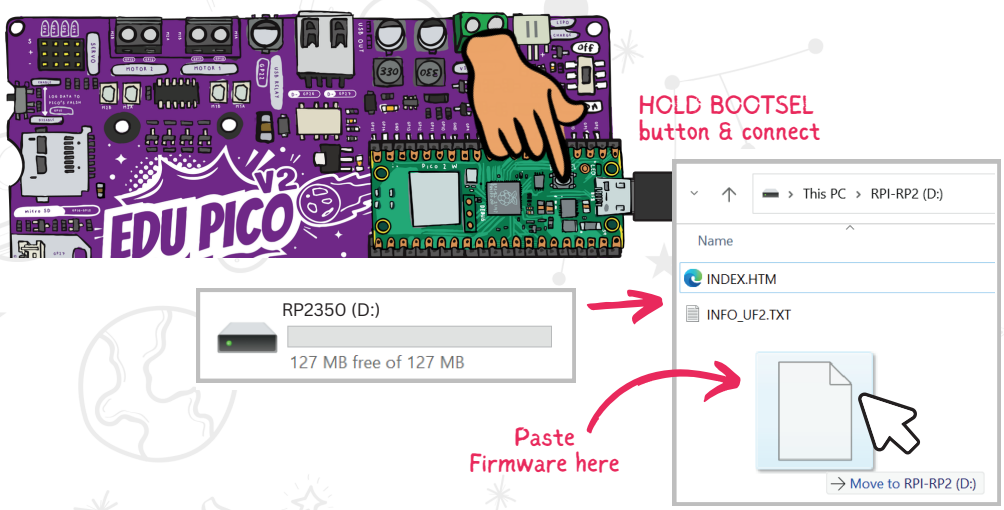
Download EDU PICO 2 Firmware

Alternatively, you can download the EDU PICO 2 CircuitPython firmware from:

- **https://circuitpython.org/board/cytron_edu_v2_pico_2w/**

Copy the firmware into the **RP2350** Drive to install the firmware.

(Make sure to HOLD BOOTSEL button when connecting the Pico to your computer USB port)

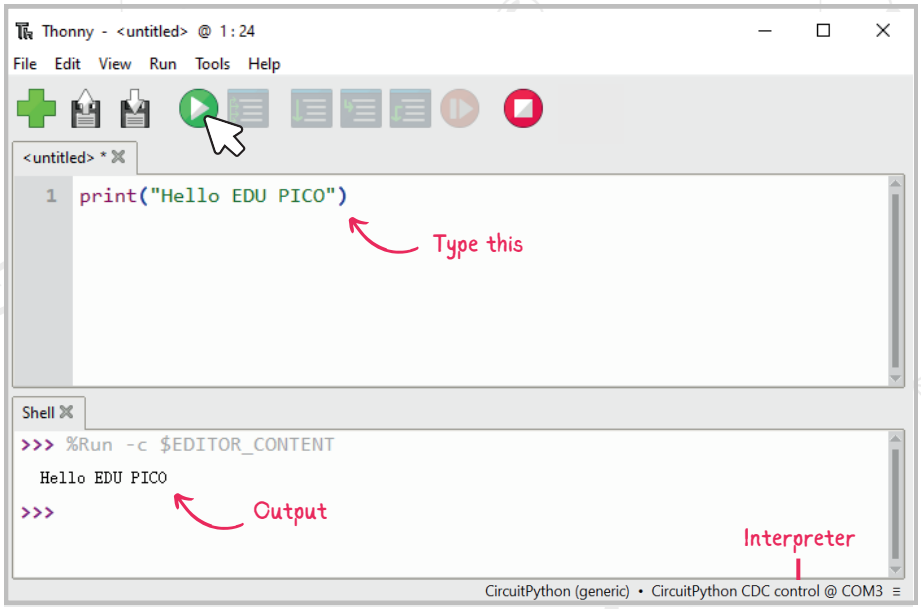


Go ahead and launch the Thonny IDE and you're all set for your first EDU PICO 2 program!

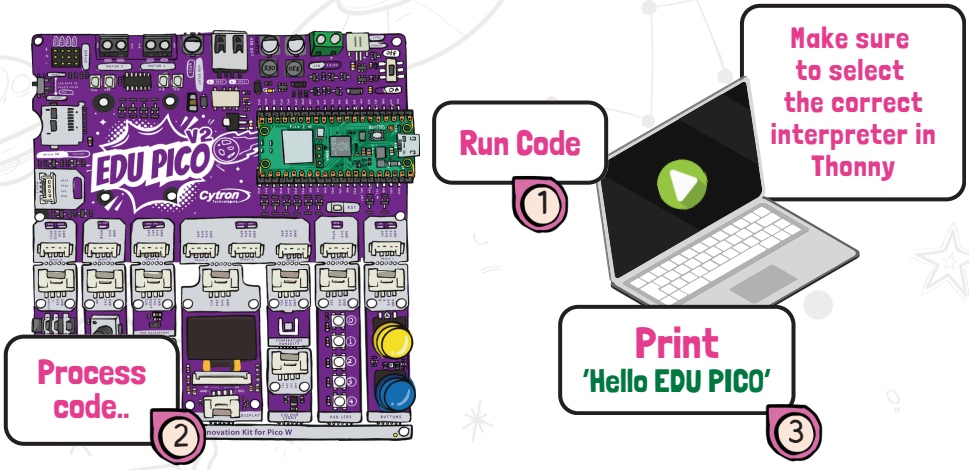
Hello EDU PICO

Hello hello, do you read me EDU PICO? Let's write our first code to the microcontroller.

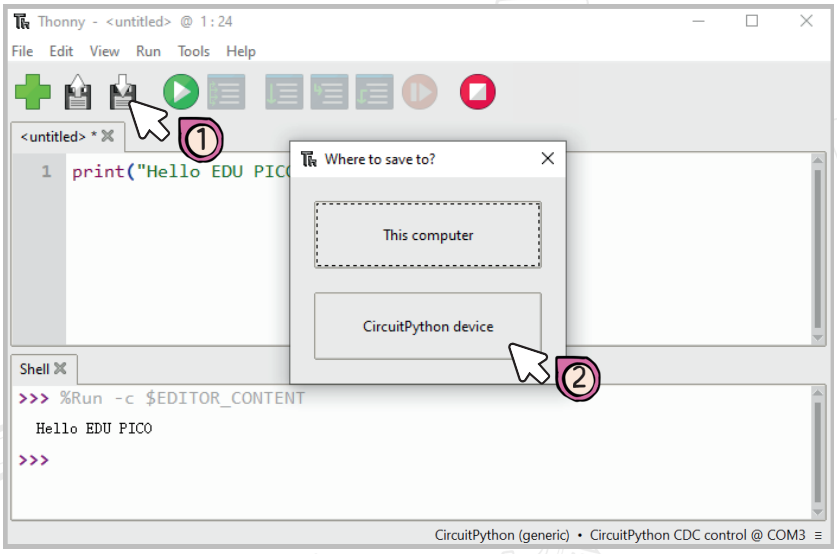
Step 1: Program the EDU PICO to print "Hello EDU PICO" text at the shell.



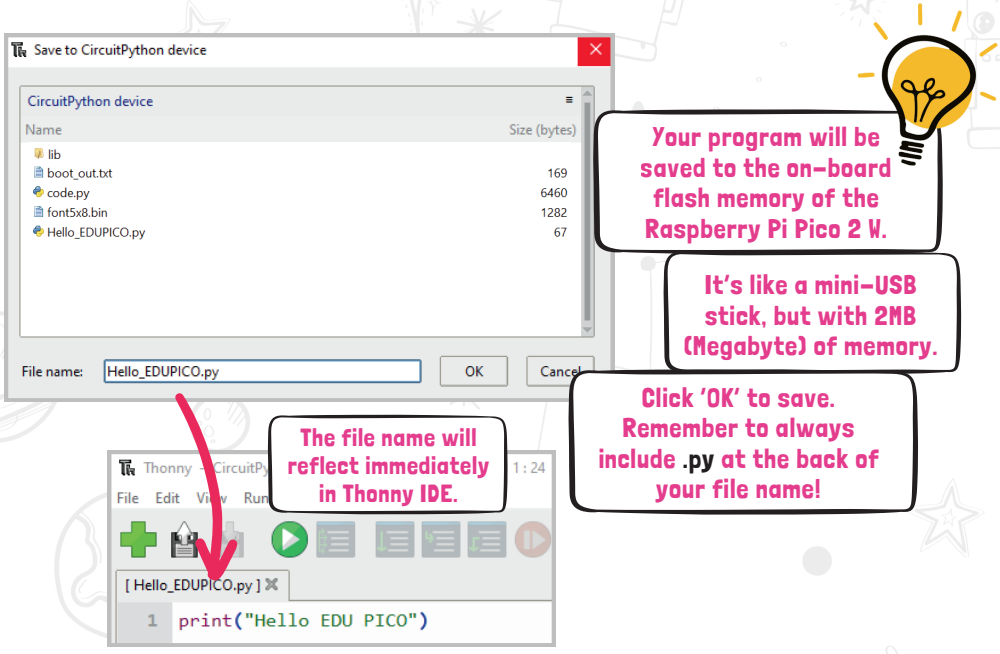
Step 2: Click the Green Button to run the code and Red Button to stop.



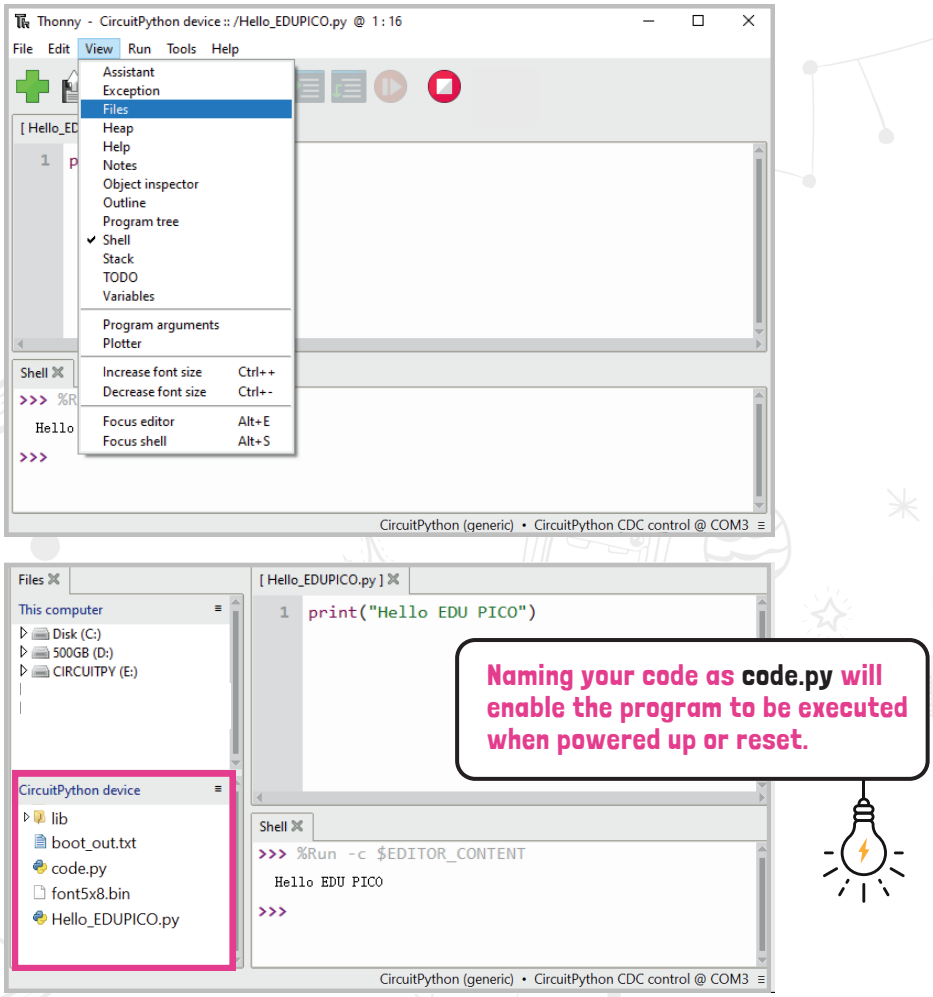
Step 3: Click **Save** and save your code into the CircuitPython device.



Step 4: Name your file as **Hello_EDUPICO.py**, then click OK to save.



Step 5: To check the code that you have already saved in your CircuitPython device, go to **View > Files**.



You will notice two sections under **Files**, the first is to your computer, and the second is to the CircuitPython device that is currently connected to your computer.

This allows you to navigate your codes inside the Raspberry Pi Pico 2 W easily; simply open the code by double-clicking the file.

To prevent **code.py** from running automatically when the EDU PICO 2 is powered on, simply delete or rename the **code.py** file.

THE MORE YOU KNOW

Execute Code when Powered Up or Reset

Sometimes you would prefer to have your code run automatically when the EDU PICO 2 is powered up. To achieve that, saving the correct code file name is important.

Once the board is powered up, CircuitPython will continuously look for code files with the following names in sequence: **code.txt**, **code.py**, **main.txt** and **main.py**, the board will then execute the first code it finds.

However, the file name **code.py** is recommended while using CircuitPython.
It is also important to keep in mind that having multiple file names mentioned earlier would create confusion and may prevent the board from executing the correct code.

Naming Program File

In programming, it is crucial to be mindful whenever we name our code files. One of the common mistakes that should be avoided is when naming your code with a similar name as your library.

When you use a similar name for your code and library, it can be challenging to differentiate between the two for your code that is calling for the particular library. This will usually result in execution errors in your code.

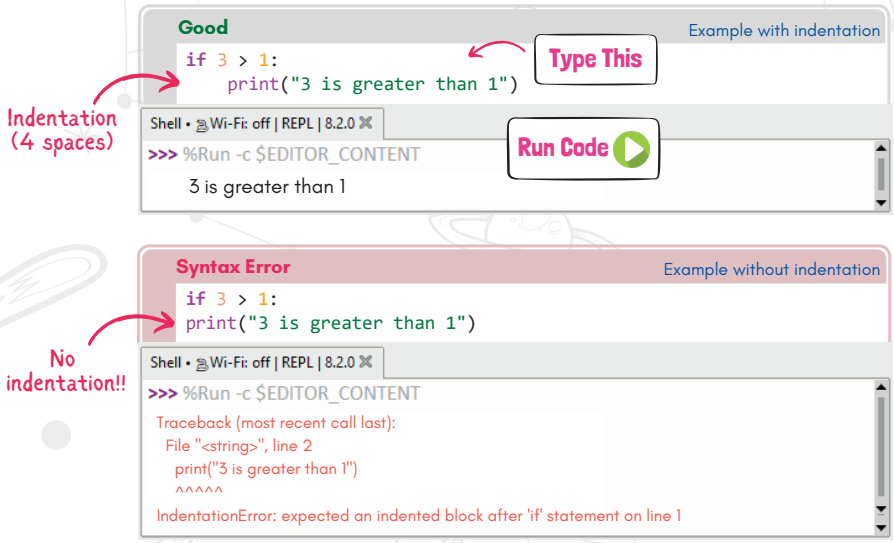
Hence, it is always recommended to use a different name for your code to avoid any naming conflicts with the CircuitPython library.

For an example, naming your file as neopixel.py will result in an error because the Neopixel library is already being named as neopixel.mpy.

In the next section, we will learn the basic syntax of CircuitPython.

Indentation

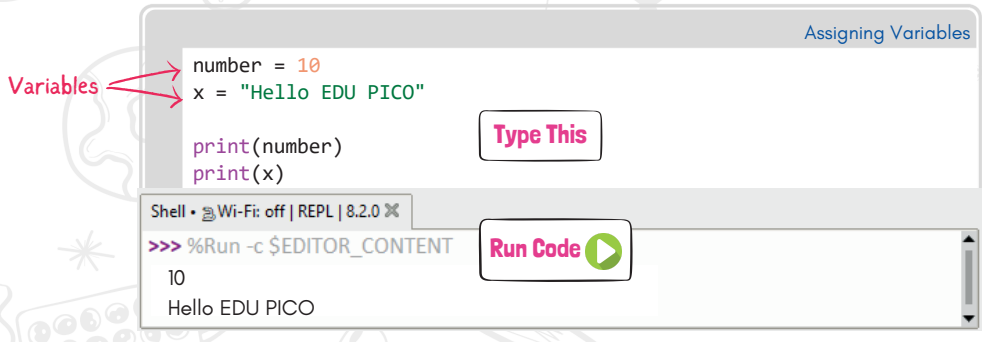
Indentation refers to the spaces at the beginning of a code line. You have the freedom to decide the number of spaces to use. While the most popular choice is four (4) spaces, its essential to use at least one to ensure readability.



While indentation in other programming languages is only for readability, in Python, it is very important.

Variables

A variable is created when you assign a value to it. In this case, the name **number** and **x** are defined as variables.



Data Types

Understanding data types is essential in programming. Variables can store data of various types, each with its distinct functionalities. Here are a few popular data types that are set when you assign a value to the variable:

Example	Data Type
x = “Hello EDU PICO”	String
x = 100	Integer
x = 100.5	Float
x = True	Boolean
x = [“red”, “green”, “blue”]	List
x = range(5)	Range
x = {1: “red”, 2: “green”}	Dictionary

Casting

One way to assign specific data type to a variable is through casting.

Specify a Variable Type

```
x = str("Hello EDU PICO")          #string
x = int(100)                       #integer
x = float(100.5)                   #float
x = bool(5)                        #boolean
x = list(("red", "green", "blue")) #list
x = range(5)                       #range
x = dict(1="red", 2=”green”)       #dictionary
```

Comments

Use comments to provide documentation within your code. Begin your comment with a **#**, and Python will interpret the remainder of the line as a comment.

Comment

```
#This is a comment, normally used for writing notes.
print("Hello, World!")

"""
This is a multiline
comment.
"""
```

You can also comment on multiple lines by using triple quotes (""" """) shown above.
Note that writing comments within your script does not affect the functionality of your code.

Operators

Operators play an important role when executing operations on variables and values.

In this section, we will explore three main types of operators: arithmetic, assignment and comparison.

1. Arithmetic Operator

ARITHMETIC_OPERATOR.py

Type This

```
x = 10
y = 3

print(x + y)  #addition
print(x - y)  #subtraction
print(x * y)  #multiplication
print(x / y)  #division
print(x % y)  #modulus
print(x ** y) #exponentiation
print(x // y) #floor division
```

Run Code

Shell

```
>>> %Run -c $EDITOR_CONTENT
13
7
30
3.3333333333333335
1
1000
3
```

2. Assignment Operator

ASSIGNMENT_OPERATOR.py

Type This

```
x = 8

x += 3  #Same as x = x + 3 = 8 + 3
print(x)
x -= 3  #Same as x = x - 3 = 11 - 3
print(x)
x *= 3  #Same as x = x * 3 = 8 * 3
print(x)
x /= 3  #Same as x = x / 3 = 24 / 3
print(x)
```

Run Code

Shell

```
>>> %Run -c
11
8
24
8.0
```

3. Comparison Operator

COMPARISON_OPERATOR.py

Type This

```
x = 10
y = 3

print(x == y)   #Equal
print(x != y)   #Not equal
print(x > y)    #Greater than
print(x < y)    #Less than
print(x >= y)   #Greater than or equal to
print(x <= y)   #Less than or equal to
```

Shell

```
>>> %Run COMPARISON_OPERATOR.
False
True
True
False
True
False
```

Run Code

COMPARISON_OPERATOR_2.py

Flip the x and y value and you will receive a different set of output.

```
x = 3
y = 10

print(x == y)   #Equal
print(x != y)   #Not equal
print(x > y)    #Greater than
print(x < y)    #Less than
print(x >= y)   #Greater than or equal to
print(x <= y)   #Less than or equal to
```

Shell

```
>>> %Run COMPARISON_OPERATOR_
False
True
False
True
False
True
```

Run Code

If.. Else (Conditions)

Now that you have learned about operators, it's time to put them into action. Conditions are used in various ways, with the most common being if statements and loops. In the following example, we use two variables **x** and **y**, to check whether **x** is greater than **y**.

If_Else.py — Example for if.. else

```
x = 10
y = 3

if x > y:
    print("x is greater than y")
else:
    print("x is not greater than y")
```

Type This

Shell

```
>>> %Run If_Else_1.py
 x is greater than y
```

Run Code

If_Elif_Else.py — Example for elif

```
x = 3
y = 10

if x > y:
    print("x is greater than y")
elif x == y:
    print("x is equal to y")
else:
    print("y is greater than x")
```

Shell

```
>>> %Run If_Elif_2.py
 y is greater than x
```

In the first example, if **x** is greater than **y**, the code will print "x is greater than y" in the shell.

The second example uses **if.. elif.. else** to check the first two conditions. As a result, it prints "y is greater than x" in the shell since both conditions were not met.

Remember to indent the statement blocks below the if...elif...else conditions to prevent syntax error.

For Loop

A for loop is a control flow statement that allows you to execute a block of code repeatedly for a fixed number of times. It is commonly used to iterate over sequences such as lists, sets, dictionaries, and strings.

For_Loop_1.py — Print each colour in a colour list

```
colours = ["red", "green", "blue"]
for x in colours:
  print(x)
```

Type This

Shell

```
red
green
blue
```

Run Code

For_Loop_2.py — Print number ranging from 0 to 2

```
for x in range(3):
  print(x)
```

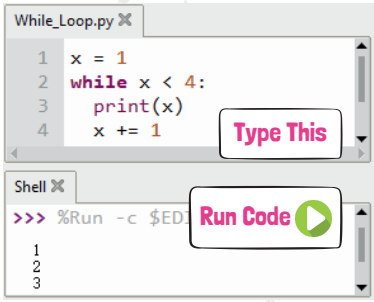
Shell

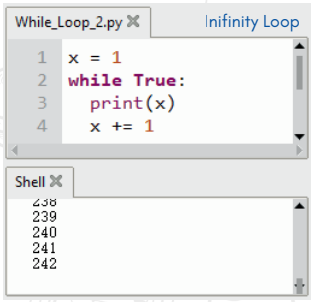
```
0
1
2
```

In the second example, the **range()** function returns a sequence of numbers, starting from **0** by default, and increments by **1** (by default), until it reaches a total number count of **3**, starting from **0**, **1**, and **2**.

While Loops

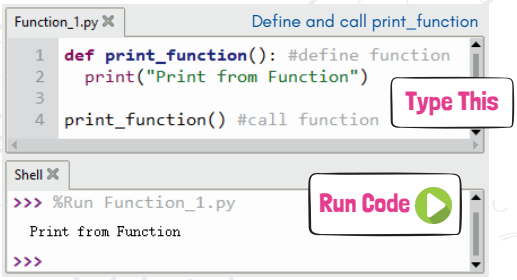
A while loop is a control flow statement that allows a block of code to execute repeatedly as long as a condition is true. The following code uses a while loop to continuously check if **x** is less than **4**. If a condition is true, the code block inside the loop executes and prints the value of **x** starting from **1** and increments it by **1** (assignment operator). This process repeats until **x** is no longer less than **4**.

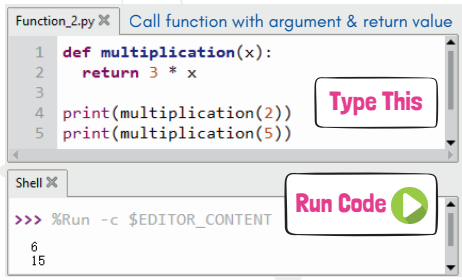




Functions

A function is a block of code which only runs when it is called. It is normally used to break down a program into smaller, more manageable pieces, making it easier to read and understand. Furthermore, you can pass data into the function for processing and return data from the function as a result.





The example on the right has a multiplication function with **multiplication(x)**. When the function is called, the values **2** and **5** are used inside the function to be multiplied by **3** and return the result of **6** and **15**.

String Format

When writing your code, it is important to maintain data readability when printing on the shell console or the OLED display.

In this section, we will introduce you to the **format()** method which makes the printing process simpler. To control the values, simply add placeholders (curly brackets {} in the text), which will allow us to format and print specific parts of a string to your liking.

Below are three examples of how this works:

Print_Format.py

Type This

```
1 temperature = 26
2 print(f"The temperature is {temperature} C")
```

Shell

Run Code

```
>>> %Run -c $EDITOR_CONTENT
 The temperature is 26 C
```

To print more values, you can simply add more placeholder {} with the **format()** method.

Print_Format_2.py

Type This

```
1 colour = "red"
2 quantity = 3
3 cost = 35
4 shirt = "I have {} {} coloured shirts that cost RM{:.2f} each."
5 print(shirt.format(quantity, colour, cost))
```

Shell

Run Code

```
>>> %Run Print_Format_2.py
 I have 3 red coloured shirts that cost RM35.00 each.
```

You can also use index numbers (a number within the curly brackets {0}) to guarantee that values are accurately placed in the designated placeholders as shown below:

Print_Format_3.py

Type This

```
1 colour = "red"
2 quantity = 3
3 cost = 35
4 shirt = "I have {0} {1} coloured shirts that cost RM{2:.2f} each."
5 print(shirt.format(quantity, colour, cost))
```

{0} {1} {2}

Shell

Run Code

```
>>> %Run Print_Format_3.py
 I have 3 red coloured shirts that cost RM35.00 each.
```

CHAPTER 2

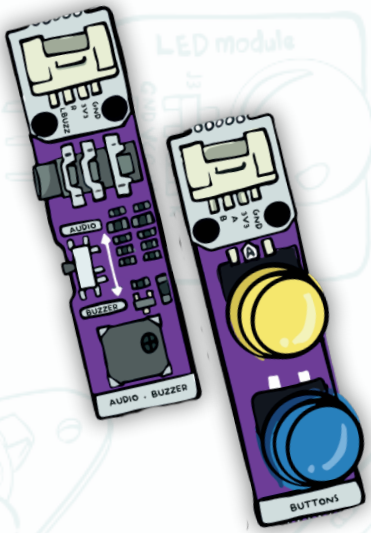
Water Drinking Reminder

Buttons & Buzzer

Introduction to **Buttons**

Introduction to **Buzzer**

Project: Water Drinking Reminder



Hello, Makers!

In this chapter, we will cover two of the most basic components on the EDU PICO 2, the push buttons and the piezo buzzer module.

The button and buzzer are among the most widely used components in real-life applications. A common example would be the buttons on our mobile phones and the buzzer in our microwave!

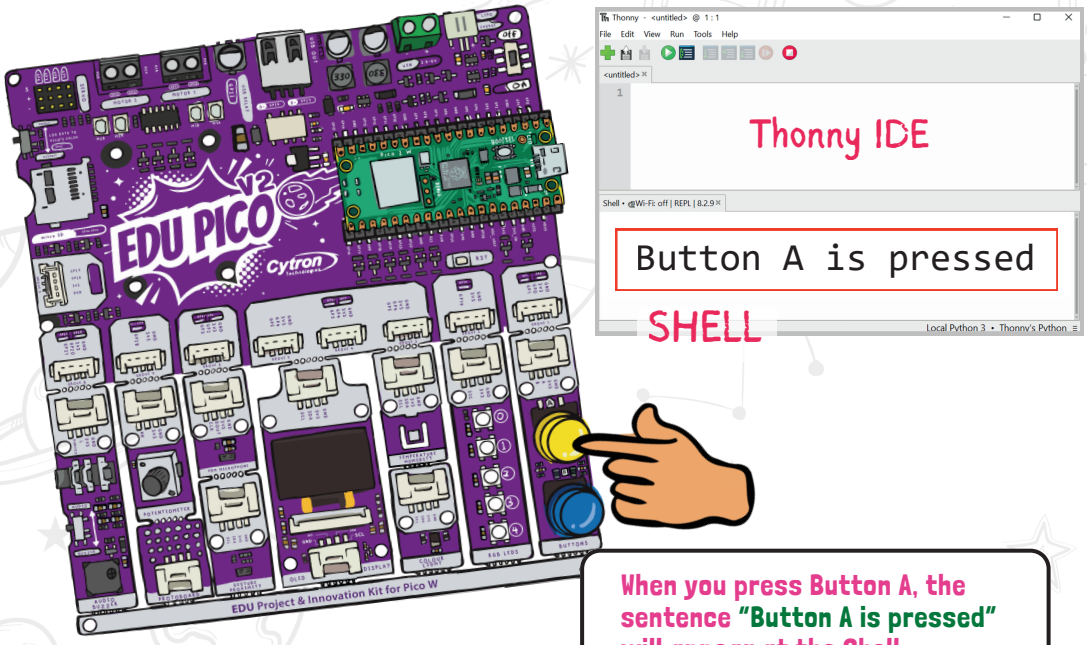
Throughout this chapter, we will explore not only the basic functionality of these components but also practice combining them in a water-drinking reminder project.

Introduction to Button

Buttons are essential electronic components in modern technology that allow us to input commands and interact with various devices. They come in all shapes and sizes, from the tiny buttons on your calculator to the larger ones on your computer keyboard.

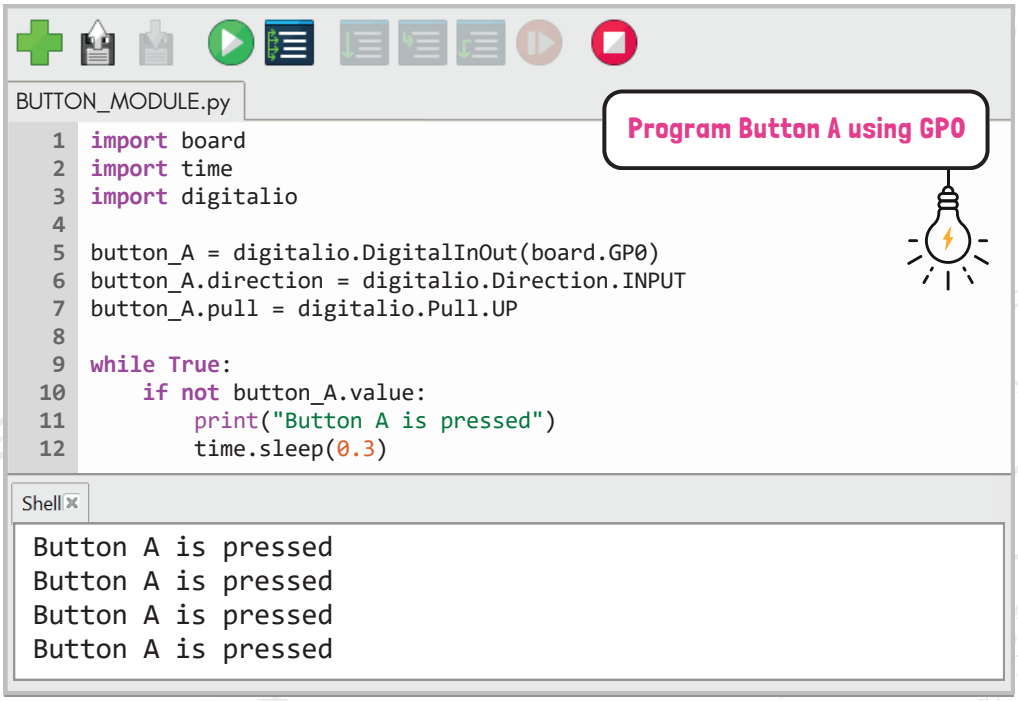
How Does This Activity Work?

- **Libraries:** board, time, digitalio.
- **Button Configuration:** Button A (Yellow) to ***GP0*** as digital input.
- **Output:**
  - If Button A is pressed, the code prints "Button A is pressed" at the shell console.



When you press Button A, the sentence "Button A is pressed" will appear at the Shell.

Code



What the Code Does </>

Import Necessary Libraries

Libraries

```
1 import board
2 import time
3 import digitalio
```

These lines of code are used to import the necessary modules and libraries that enable the EDU PICO to understand the functions of button and time.
The imported modules are as follows:

Line 1: Provides a way to reference the GPIO pins on the EDU PICO.

Line 2: Provides time-related functions such as **time.sleep()** for introducing delays.

Line 3: Allows interaction with digital input/output pins.

Initialize Hardware Components

Button Pin Configuration

```
5 button_A = digitalio.DigitalInOut(board.GP0)
6 button_A.direction = digitalio.Direction.INPUT
7 button_A.pull = digitalio.Pull.UP
```

Line 5: Set up a connection between the EDU PICO and a button. It specifies pin ***GP0*** to be used for the button.

Line 6: Informs the EDU PICO whether **button_A** is an input or output. In this case, it's set as input which will be used to read the state of the button, either logic high (3.3V) or low (0V).

Line 7: Configures the internal pull-up of the button pin. This keeps the reading at logic high (3.3V) when the button is not pressed.

Enter a Continuous Loop

Infinite Loop

```
9 while True:
```

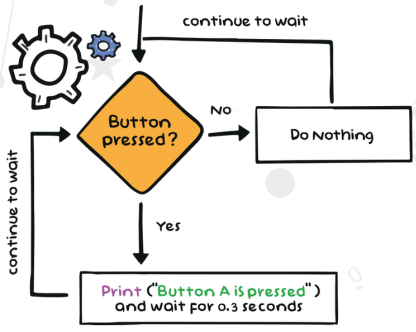
while True initiates an infinite loop. The code inside this loop will run repeatedly as long as the condition remains true.

Read Button Value

```
10     if not button_A.value:
11         print("Button A is pressed")
12         time.sleep(0.3)
```

Line 10 - 12: Checks the value of button A. **button_A.value** returns True if the button is not pressed (logic high) and False if the button is pressed (logic low).

If the condition is True, it prints the message "Button A is pressed" and is delayed for **0.3** seconds.



Activity with AI

Prompt Framework: Buttons

Task: Print "A" when button A is pressed and "B" when button B is pressed.

Step 1: Teach the AI

Copy and paste your working code from **Page 29** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Button A = GP0, Button B = GP1

Here is my current code:
<PASTE YOUR CODE FROM PAGE 29>"

Step 2: Ask the AI

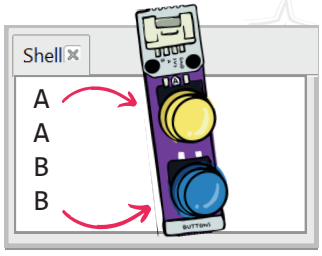
Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

Prompt #2:
"Add Button B. Print 'A' when A is pressed, 'B' when B is pressed."

Step 3: Check the Answer

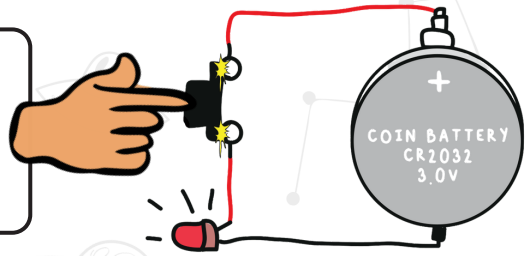
Copy and paste the code in Thonny, then run the code. Check if "A" is printed in the shell console when Button A is pressed, and "B" is printed on the shell console when Button B is pressed.

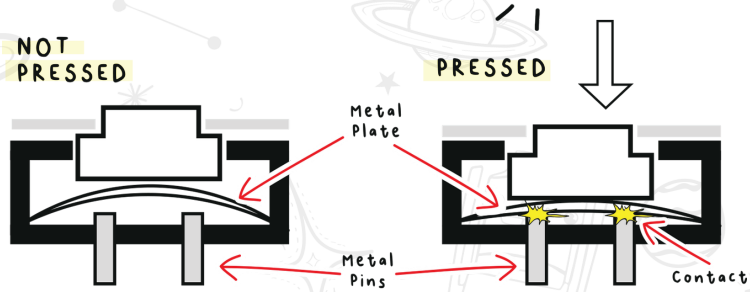


THE MORE YOU KNOW

Buttons

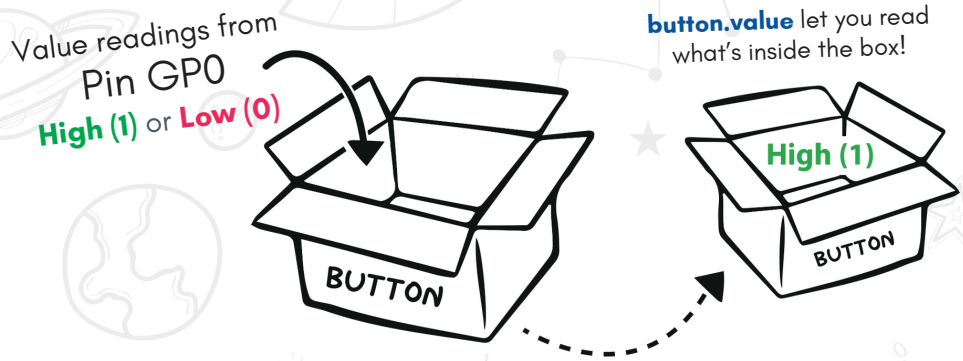
To turn on the LED, simply press the button to close the circuit. Releasing the button opens or disconnects the circuit and turns off the LED.





Variables

A variable is like a box that can hold different values. Think of it like a container that can store numbers, words, or anything else. You can give this box a name, like **x** or **button**, and then put different information inside it as needed.

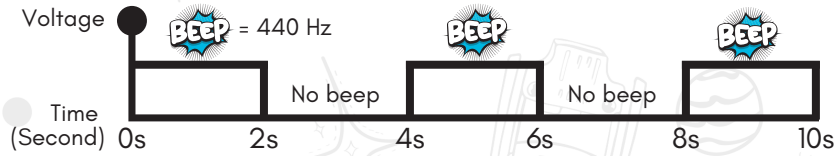


Introduction to Buzzer

Want to make some noise with your EDU PICO? In this section, you will learn how to activate your EDU PICO built-in piezo buzzer. While you're at it, go ahead and hook up the EDU PICO to a headset or a speaker at the buzzer module too!

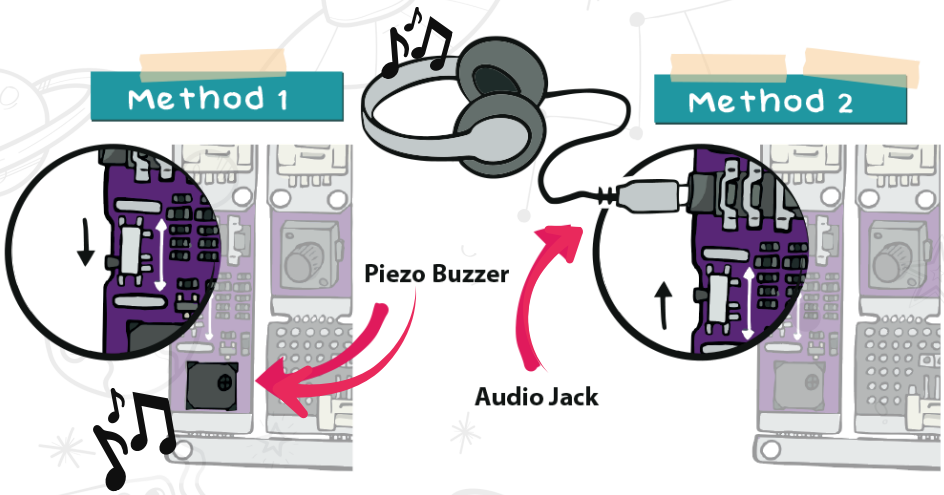
How Does This Activity Work?

- **Libraries:** board, simpleio.
- **Audio / Buzzer Configuration**: Buzzer to ***GP21***.
- **Output**: When the code is executed, the EDU PICO buzzer module will continuously beep at a **2** second interval with a frequency of 440 Hz.



- **Action Required**: Flip the buzzer module switch downwards to enable buzzer output (Method 1 illustration as shown below).

2 Methods to use the Audio Buzzer Module



Code

BUZZER_MODULE.py

```
1  import board
2  import simpleio
3
4  buzzer = board.GP21
5
6  while True:
7      simpleio.tone(buzzer, 440, 1)
8      simpleio.tone(buzzer, 0, 2)
```

You will need to import **simpleio** library for the code to work!



Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Libraries

```
2  import simpleio
```

The **simpleio** module offers a set of functions that simplify the input and output operations for the EDU PICO, making it easier to work with hardware components. In this case, it allows the user to generate various tones with the EDU PICO's buzzer.

Buzzer Pin

```
4  buzzer = board.GP21
```

Line 4: Assigns ***GP21*** to the variable **buzzer** to control the buzzer module.

Tone Frequency

```
6  while True:
7      simpleio.tone(buzzer, 440, 1)
8      simpleio.tone(buzzer, 0, 2)
```

Line 6 - 8: Generates a tone on the buzzer pin at a frequency of **440 Hz** (which corresponds to the musical note A4) for a duration of **1** second and generates a "silent" tone (**0 Hz**) on the buzzer for **2** seconds. The code will run continuously.

Activity with AI

Prompt Framework: Buzzer

Task: Make the buzzer play a short melody of "*Twinkle Twinkle Little Star*".

Step 1: Teach the AI

Copy and paste your working code from **Page 34** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Here is my current code:
<PASTE YOUR CODE FROM PAGE 34>"

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

Prompt #2:
"Make the buzzer play the melody "Twinkle Twinkle Little Star" once when the program starts. Use short beeps for each note."

Step 3: Check the Answer

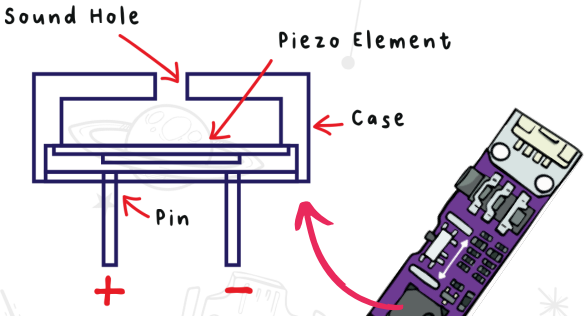
Copy and paste the code in Thonny, then run the code. Check if the buzzer plays the "Twinkle Twinkle Little Star" melody once when the program starts.

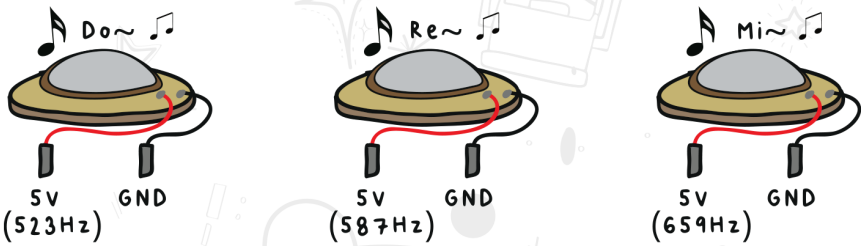
For learning continuation, you can ask the AI to play the melody when Button A is pressed! Make sure to Teach the AI about buttons using the Button code from Page 29.

THE MORE YOU KNOW

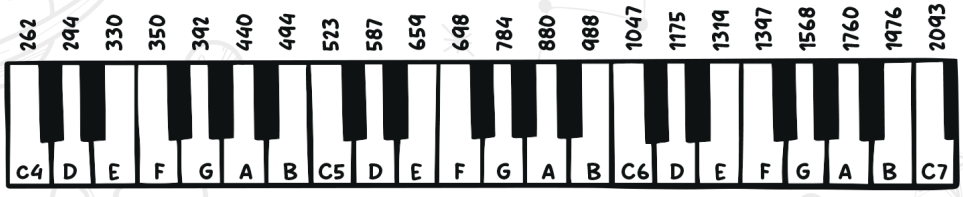
Piezo Buzzer

A piezo buzzer is frequently used to generate sound through the vibration of a piezo element when an electric signal flows across it. By altering the frequency of the electric signal, the rate of vibrations adjusts accordingly. As a result, the piezo buzzer generates sound with a distinct tone.





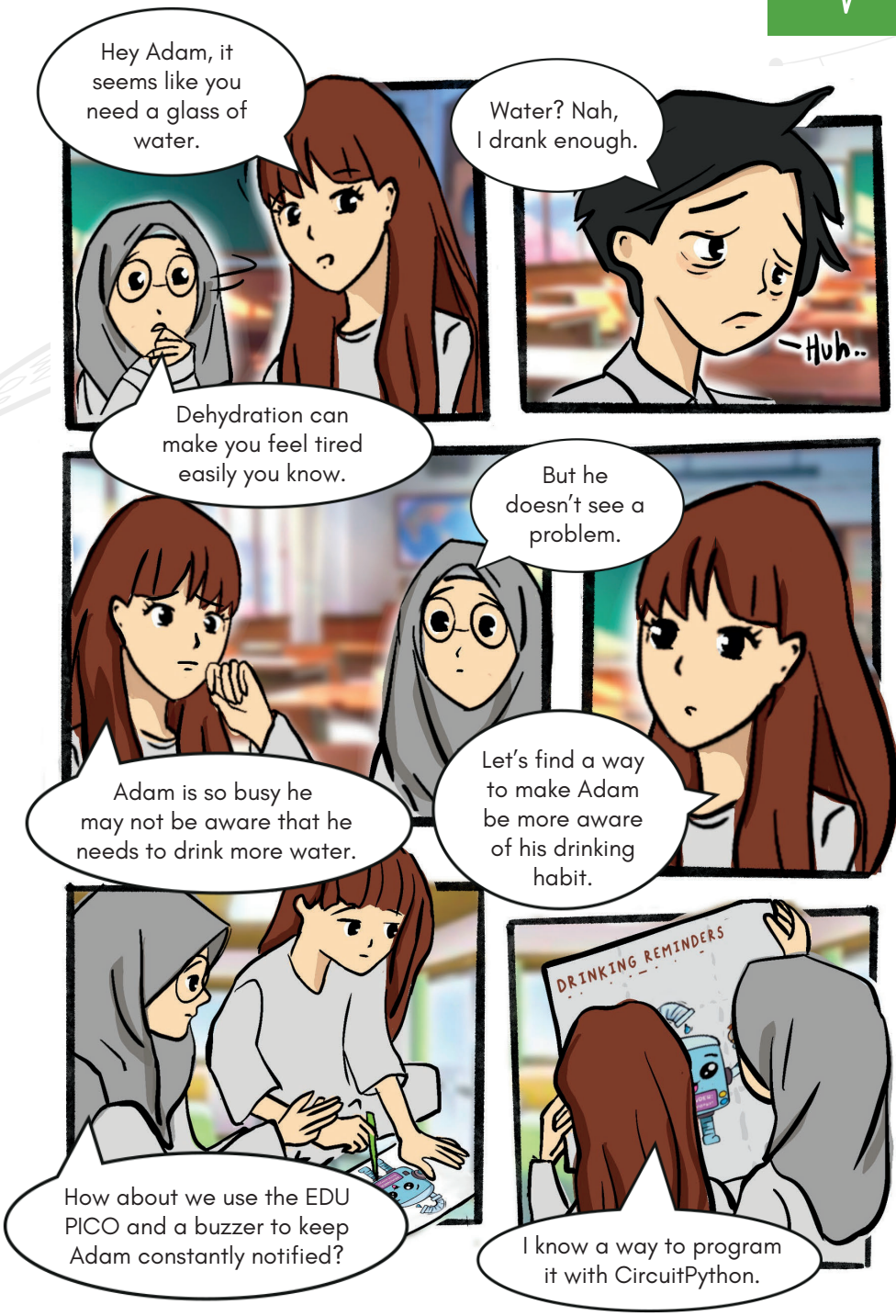
The placement of a musical note along the vertical lines indicates the specific tone to be played. As the note is positioned higher, the pitch or frequency of the sound increases.



You can program EDU PICO to play other songs by following music notes.

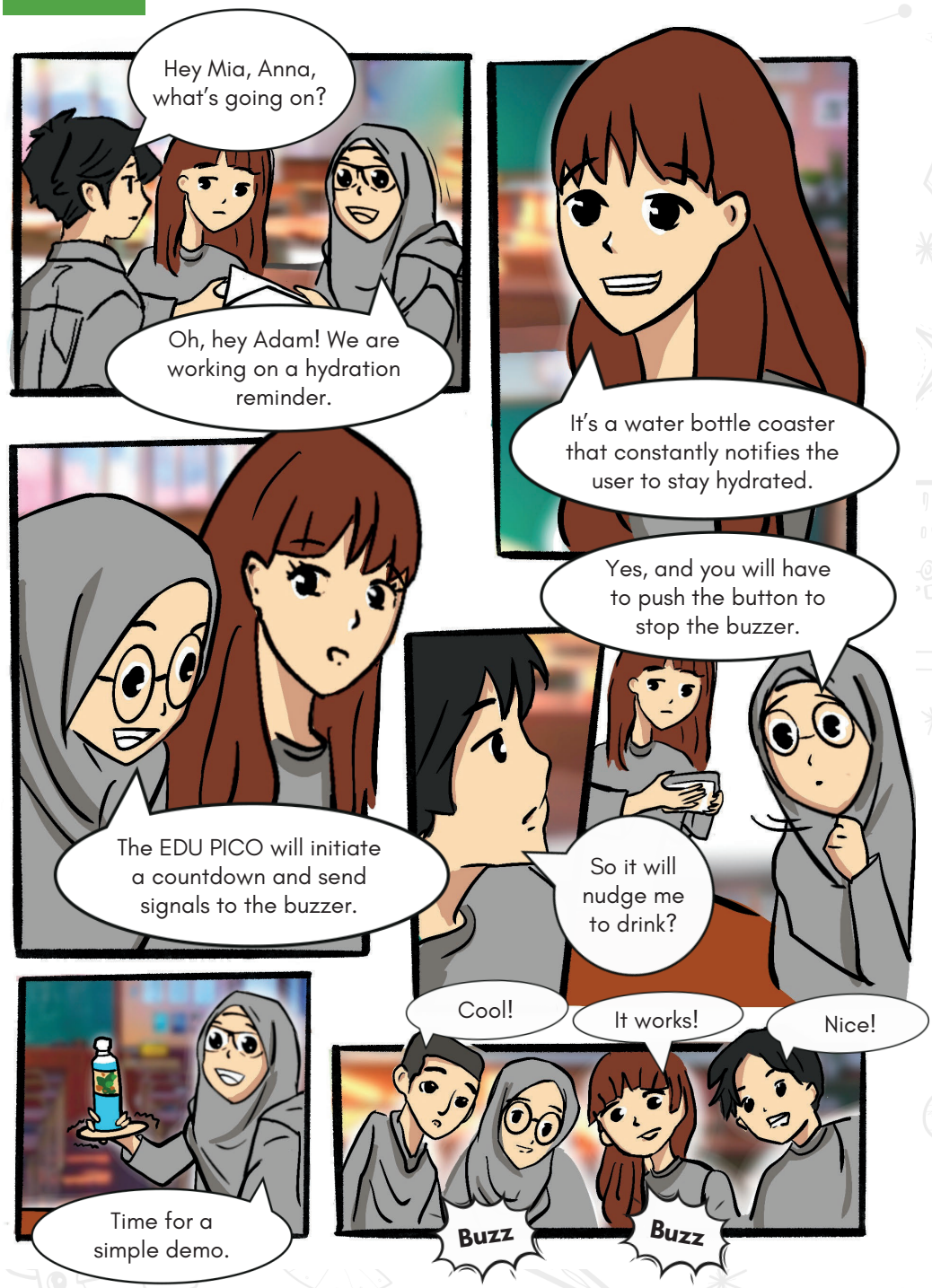
PROLOGUE: WATER DRINKING REMINDER

3 GOOD HEALTH AND WELL-BEING



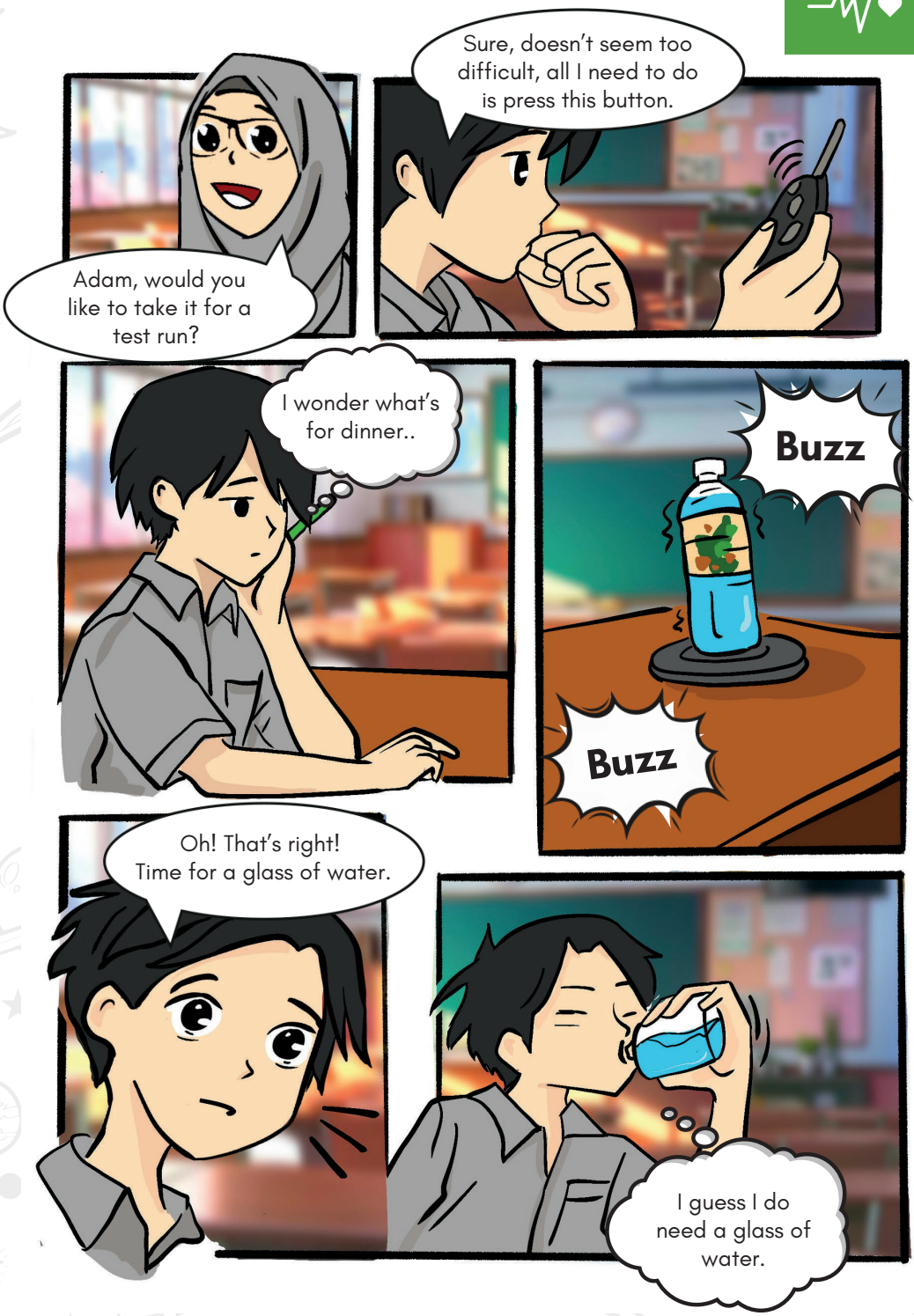
3 GOOD HEALTH AND WELL-BEING

PROLOGUE: WATER DRINKING REMINDER



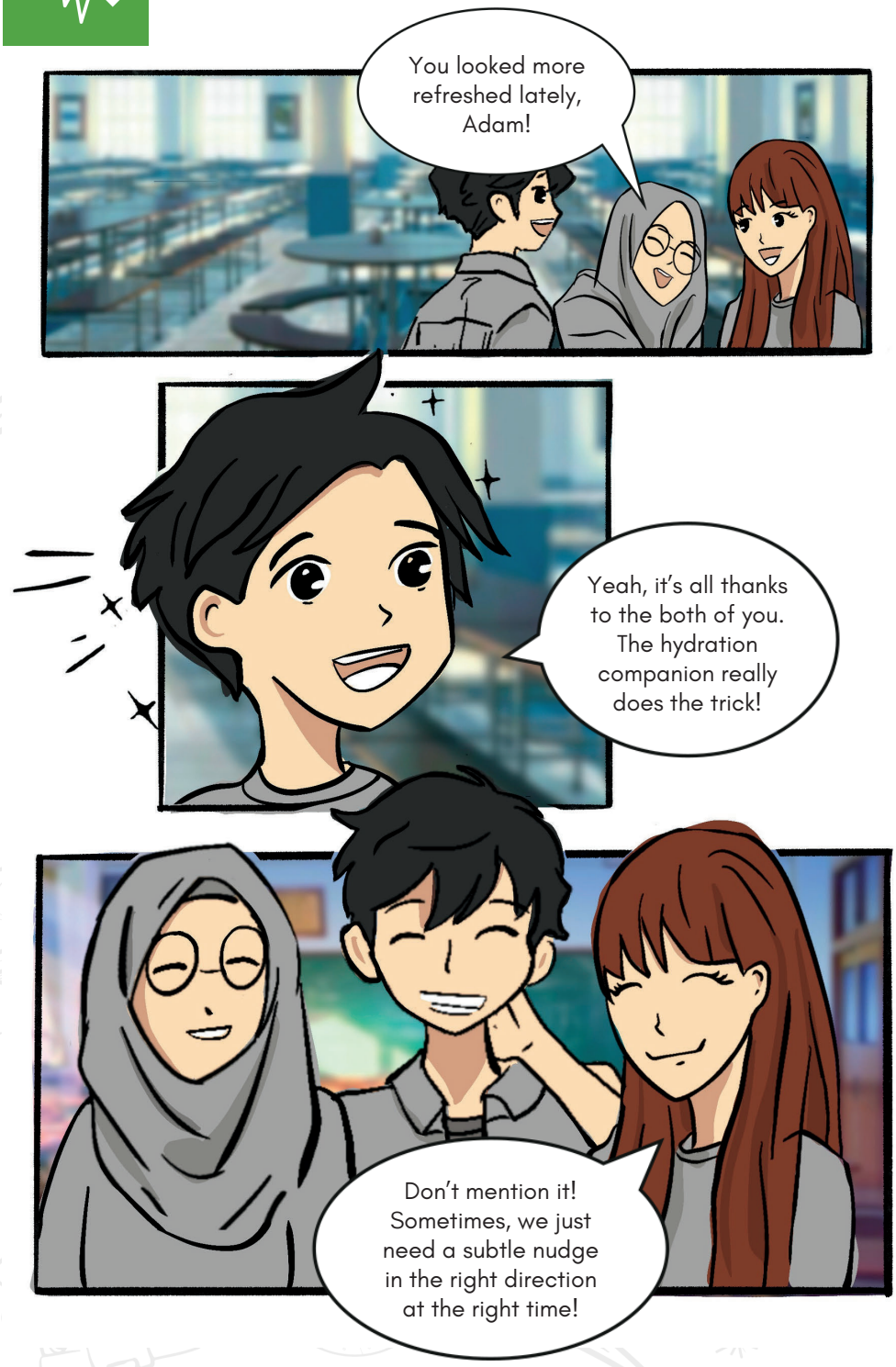
PROLOGUE: WATER DRINKING REMINDER

3 GOOD HEALTH AND WELL-BEING



3 GOOD HEALTH AND WELL-BEING

PROLOGUE: WATER DRINKING REMINDER



Water Drinking Reminder

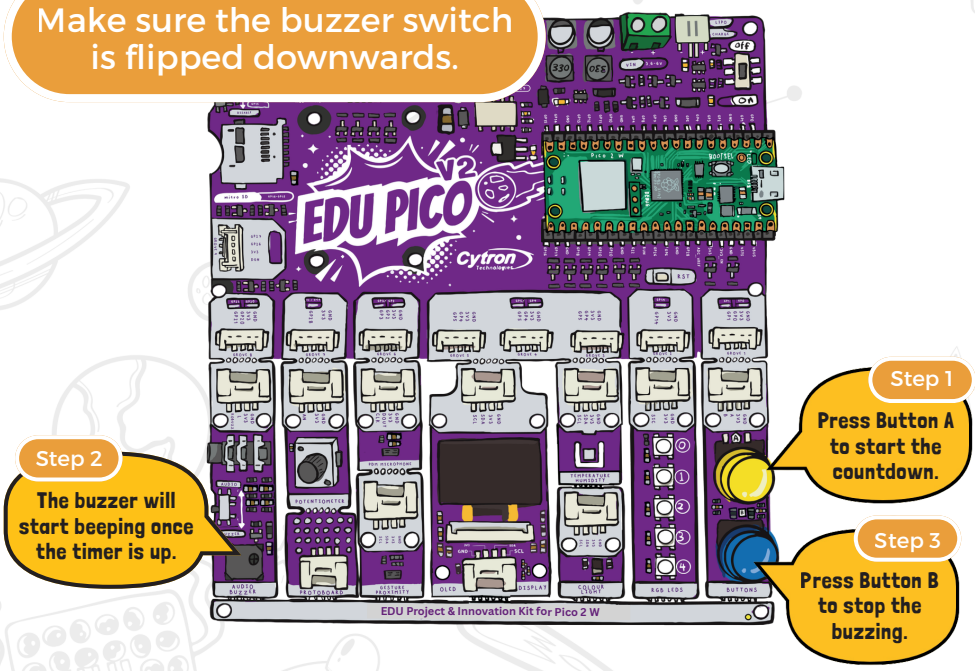
Hydration Companion

Have an Adam in your life? Let's help them solve their dehydration issue by building a water drinking reminder with EDU PICO! Don't worry, we will keep your first project simple. Ultimately, you will learn how to combine codes from multiple components. Let's go!

How Does This Activity Work?

- **Libraries:** board, digitalio, time, simpleio.
- **Buttons Configuration:** Button A (Yellow) = ***GP0***, Button B (Blue) = ***GP1***.
- **Input:**
  - Press Button A to activate the 5 second (default) countdown.
  - Press Button B to reset the buzzer.
- **Output:**
  - The buzzer will start beeping once the countdown reaches zero.
  - The program will print the reminder text and timer status to the user at the shell console.

Make sure the buzzer switch is flipped downwards.



Code



PROJECT_WATER_DRINKING_REMINDER.py

```
import board, digitalio, time, simpleio

button_start = digitalio.DigitalInOut(board.GP0)
button_start.direction = digitalio.Direction.INPUT
button_start.pull = digitalio.Pull.UP

button_stop = digitalio.DigitalInOut(board.GP1)
button_stop.direction = digitalio.Direction.INPUT
button_stop.pull = digitalio.Pull.UP

buzzer_pin = board.GP21
buzzer_status = False

print("Press Button A to start")

while True:
    if not button_start.value:
        duration = 5
        end_time = time.monotonic() + duration
        print("You have activated the water drinking reminder")

        while time.monotonic() < end_time:
            pass

        print("Reminder to Drink Water! Press Button B to reset")
        buzzer_status = True

    if not button_stop.value:
        print("Timer reset")
        buzzer_status = False
        time.sleep(0.3)
        print("Press Button A to start")

    if buzzer_status:
        simpleio.tone(buzzer_pin, 800, 0.5)
        simpleio.tone(buzzer_pin, 0, 0.5)
```

Shell

```
Press Button A to start
You have activated the water drinking reminder
Reminder to Drink Water! Press Button B to reset
Timer reset
```

Click the Green Button  to run the code and Red Button  to stop.

What the Code Does

Import Necessary Libraries

Libraries

```
1 import board, digitalio, time, simpleio
```

This line imports the necessary libraries for the code to utilize buttons, time and sound functions.

Initialize Hardware Components

Buttons

```
3 button_start = digitalio.DigitalInOut(board.GP0)
4 button_start.direction = digitalio.Direction.INPUT
5 button_start.pull = digitalio.Pull.UP
6
7 button_stop = digitalio.DigitalInOut(board.GP1)
8 button_stop.direction = digitalio.Direction.INPUT
9 button_stop.pull = digitalio.Pull.UP
```

Line 3 - 5: Initialize **button_start** (Yellow Button) for starting the reminder.

Line 7 - 9: Initialize **button_stop** (Blue Button) for stopping or resetting the alarm.

Buzzer

```
11 buzzer_pin = board.GP21
12 buzzer_status = False
```

Line 11 - 12: Prepare the buzzer and set the variable **buzzer_status** to false, which means the buzzer is off at the beginning of the code.

Initial Message

```
14 print("Press Button A to start")
```

Line 14: Print an initial message to the console: "Press Button A to start"

Enter a Continuous Loop

button_start

```
16 while True:
17     if not button_start.value:
18         duration = 5
19         end_time = time.monotonic() + duration
20         print("You have activated the water drinking reminder")
21
22         while time.monotonic() < end_time:
23             pass
24
25         print("Reminder to Drink Water! Press Button B to reset")
26         buzzer_status = True
```

Line 17: Check if Button A **button_start** is pressed.

Line 18 - 19: If the button is pressed, set a **duration** of **5** seconds for the reminder timer and calculate the **end_time** by adding **duration** to the current time using **time.monotonic()**.

Line 20: Print "You have activated the water drinking reminder".

Line 22 - 23: Enter an inner loop that continues until the current time is less than **end_time**. This inner loop actively wait for the reminder duration to pass.

Line 25: After the reminder duration has passed, print "Reminder to Drink Water! Press Button B to reset".

Line 26: Set **buzzer_status** variable to True.

button_stop

```
28     if not button_stop.value:
29         print("Timer reset")
30         buzzer_status = False
31         time.sleep(0.3)
32         print("Press Button A to start")
```

Line 28 - 31: Check if Button B **button_stop** is pressed. If it is, reset the timer and set **buzzer_status** to False. A brief delay of **0.3** seconds is introduced to prevent multiple rapid button presses, this is also known as the switch bounce effect.

Line 32: Print "Press Button A to start" to indicate the timer waiting for user input.

buzzer_on

```
34      if buzzer_status:
35          simpleio.tone(buzzer_pin, 800, 0.5)
36          simpleio.tone(buzzer_pin, 0, 0.5)
```

Line 34 - 36: If **buzzer_status** is True, activate the buzzer using **simpleio.tone()** to produce **800 Hz** tone for **0.5** seconds, followed by a **0 Hz** (silent) tone for **0.5** seconds. This alarm is used to signal the user when its time to drink water.

CHALLENGE - USER INPUT DURATION

In the original code, the timer runs for only 5 seconds by default. In this challenge, you are required to modify the code to allow the user to customize the duration of the water-drinking reminder. Check out the hint below to guide you with the modification.




```
if not button_start.value:
    user_input = input("Enter the reminder duration (in seconds): ")
    user_duration = int(user_input)
    if user_duration >= 1:
        duration = user_duration
        print(f"Reminder duration set to {duration} seconds")
    else:
        print("Reminder to Drink Water! Press Button B to reset")
        buzzer_status = False
        continue

        end_time = time.monotonic() + duration
        print("You have activated the water drinking reminder")

        while time.monotonic() < end_time:
            pass

        print("Reminder to Drink Water! Press Button B to reset")
        buzzer_status = True
```

This code allows you to set the duration manually through the shell. The code runs when Button A is pressed.

CHAPTER 3

Gesture Reaction Game

OLED & Gesture Sensor

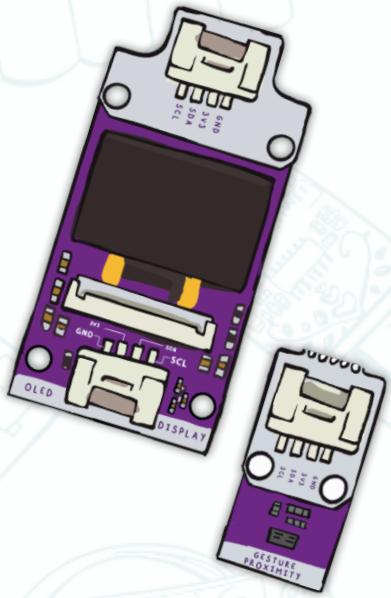
Introduction to OLED Display

Introduction to Gesture Sensor

Project: Gesture Reaction Game

Hello, Makers!

Imagine being able to control appliances around you with just a wave of your hand, or programming the screen on your digital watch anytime. In this chapter, we will explore two exciting components: the gesture sensor and the OLED display. The gesture sensor allows us to detect movements in our surroundings and trigger actions based on those motions. Meanwhile, the OLED display provides a platform to display text and graphics, whether for data monitoring or as a user interface for a simple game!



Introduction to OLED

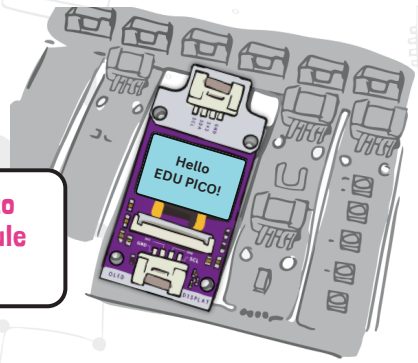
You can think of the OLED display as a very small, electronic billboard. Its like the display on your smartphone but much smaller. Like an electronic billboard, you can choose what to show on it. In this activity, we will learn how to code and print on the EDU PICO's OLED display module.

How Does This Activity Work?

- **Libraries:** board, busio, time, adafruit_ssd1306, **font5x8.bin** ← Located in CircuitPython Root Directory
- **OLED I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Output:**
  - Invert OLED display at initialization with a bluish-white background.
  - Print "Hello EDU PICO" text in the middle of the OLED in black colour:
    - "Hello" with a coordinate of x = 50, y = 20.
    - "World" with a coordinate of x = 13, y = 35.

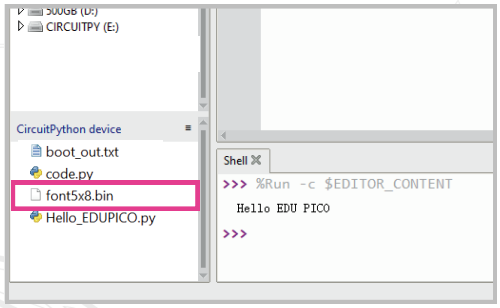
I2C (Inter-integrated Circuit) is a communication method that lets electronic components talk to each other by sharing a common connection with unique addresses.

Use the I2C protocol to connect the OLED module with the EDU PICO.



Navigating font5x8.bin

It is vital to make sure your CircuitPython root directory contains the font5x8.bin file.
The file is a binary file that contains a bitmap font used by the framebuf library to render text on the OLED display.



Code



OLED_MODULE.py

```
1  import board, busio, time
2  import adafruit_ssd1306
3
4  i2c = busio.I2C(board.GP5, board.GP4)
5  oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
6
7  oled.fill(0)
8  oled.invert(True)
9  oled.text("Hello", 50, 20, 1)
10 oled.text("EDU PICO!", 13, 35, 1, size = 2)
11 oled.show()
```

Click the Green Button  to run the code and Red Button  to stop.

What the Code Does </>

Import Necessary Libraries

Libraries

```
2 import adafruit_ssd1306
```

adafruit_ssd1306 enables operation with SSD1306 OLED display.

Initialize Hardware Components

Configure OLED I2C Pins

```
4 i2c = busio.I2C(board.GP5, board.GP4)
5 oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
```

Line 4: Initialize I2C communication bus using the **busio.I2C** class and assign ***GP5*** for SCL (Serial Clock Line) and ***GP4*** for SDA (Serial Data Line).

Line 5: Configure the OLED display with a resolution of **128x64** pixels while forming the connection through I2C communication.

Enter a Continuous Loop

Configure OLED Pins

```
7  oled.fill(0)
8  oled.invert(True)
9  oled.text("Hello", 50, 20, 1)
10 oled.text("EDU PICO!", 13, 35, 1, size = 2)
11 oled.show()
```

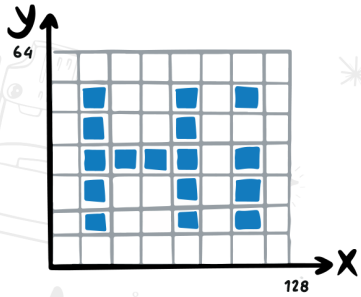
Line 7: Clear the OLED display by filling it with black (**0** represents black font).

Line 8: Invert the background colour to appear in white and text in black.

Line 9: Display the text "Hello" with the coordinates of x = **50** and y = **25**.

Line 10: Display the text "EDU PICO!" with the coordinates of x = **13** and y = **35**. The size of the text is enlarged to size = 2.

Line 11: Update the OLED display to reflect the latest changes.



The ssd1306 OLED display have 128 pixels from left to right (X) and 64 pixels from top to bottom (Y).

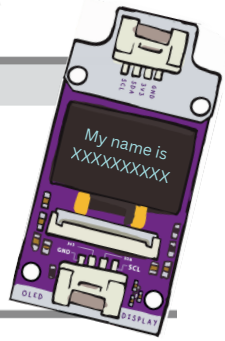
In summary, this code initializes the OLED display and print "Hello EDU PICO!" text in black font continuously.

MINI ACTIVITY

Modify the code to allow the user to type their name in the shell console and print the name on the OLEDdisplay. Replace the original code from line 7 to 11 with the code provided below:

QUICK TIPS

```
while True:
    oled.fill(0)
    oled.text("My name is", 35, 25, 1)
    user_input = input("Name: ")
    oled.text(user_input, 40, 40, 1)
    oled.show()
```



Activity with AI

Prompt Framework: OLED Display

Task: Sweep the text "Hello EDU PICO" across the OLED display from left to right.

Step 1: Teach the AI

Copy and paste your working code from **Page 48** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Here is my current code:
<PASTE YOUR CODE FROM PAGE 48>"

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

Prompt #2:
"Make the text "EDU PICO!" sweep smoothly across the screen from left to right and loop forever."

Step 3: Check the Answer

Copy and paste the code in Thonny, then run the code. Check if the text "EDU PICO!" moves smoothly across the OLED display. The text should continuously sweep from left to right.

THE MORE YOU KNOW

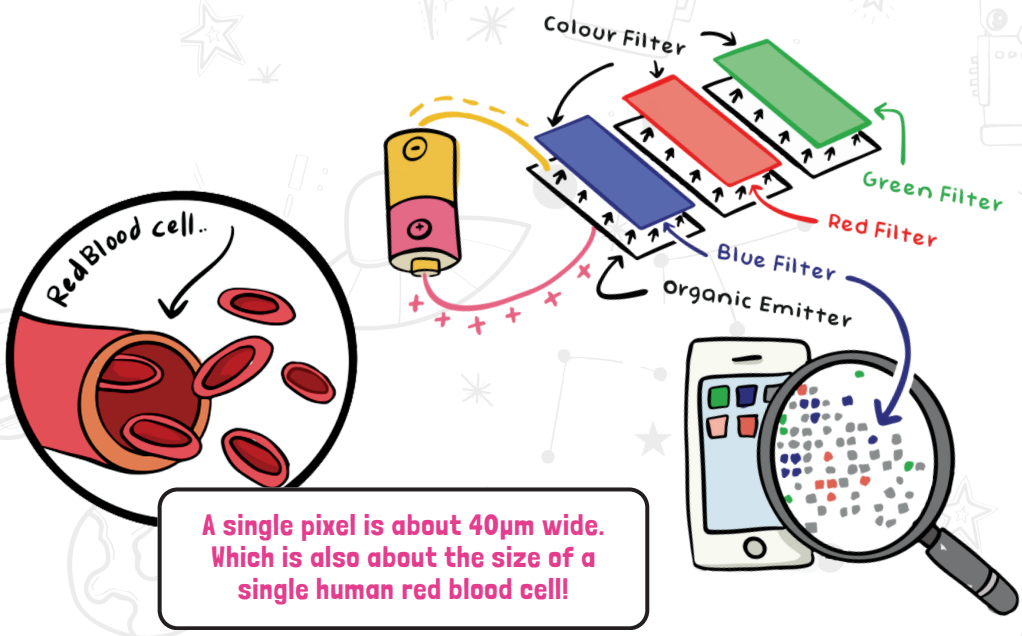
OLED Pixel

OLED (Organic Light-Emitting Diodes) are becoming the most common display technology in our devices. Most smartphones and many TVs today use OLED technology.

The name OLED originates from the organic compounds in its pixels. These compounds are primarily made of carbon and hydrogen, and form organic emitters that emit light when excited by an electrical current.

The organic emitters emit white light initially, which passes through a filter to create colours (Red, Green, or Blue). By adjusting the electrical current in each organic emitter, we can control the colour of a single OLED pixel.

Note: The OLED display included with the EDU PICO can only emit in blue colour.

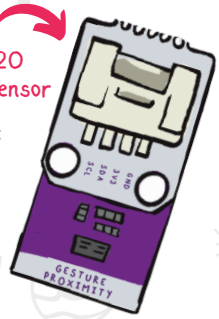


Introduction to Gesture Sensor

In this section, we'll explore how PAJ7620 gesture sensor can be used to detect hand gestures. The PAJ7620 is a tiny electronic device that can "see" hand gestures and measure the amount of light around it. It's like having a tiny robot eye!

How Does This Activity Work?

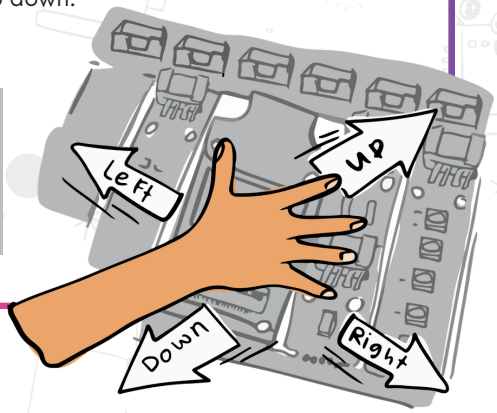
- **Libraries:** board, busio, edupic2_paj7620.
- **PAJ7620 I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.

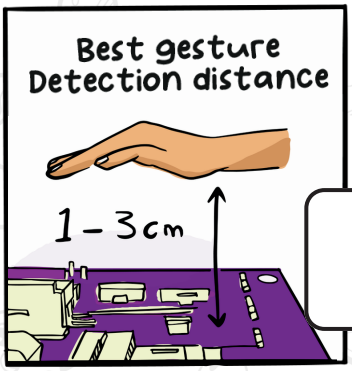


- **Input:** By performing various hand gestures in front of the sensor, the code will print the corresponding direction to the shell console:
  - "left" when moving from right to left.
  - "right" when moving from left to right.
  - "up" when moving from down to up.
  - "down" when moving from up to down.
- **Output:**

```
>>> %Run -c $EDITOR_CONTENT

  left
  right
  up
  down
```





Move your hand across the sensor slowly to ensure the sensor can detect the gesture.

Code

GESTURE_SENSOR_MODULE.py

```
1  import board, busio
2  import edupico2_paj7620
3
4  i2c = busio.I2C(board.GP5, board.GP4)
5  sensor = edupico2_paj7620.PAJ7620(i2c)
6
7  while True:
8      gesture = sensor.gesture()
9      if gesture & sensor.UP:
10         print("up")
11     if gesture & sensor.DOWN:
12         print("down")
13     if gesture & sensor.LEFT:
14         print("left")
15     if gesture & sensor.RIGHT:
16         print("right")
17     if gesture & sensor.CW:
18         print("clockwise")
19     if gesture & sensor.CCW:
20         print("counter-clockwise")
```

Shell

```
counter clockwise
clockwise
right
left
up
down
```

Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Libraries

```
1 import board, busio
2 import edupico2_paj7620
```

Line 2: Imports the library for the PAJ7620 gesture sensor.

PAJ7620 Functions

```
4 i2c = busio.I2C(board.GP5, board.GP4)
5 sensor = edupico2_paj7620.PAJ7620(i2c)
```

Line 4 - 5: Initializes an I2C communication bus with pins ***GP5*** (SCL - Serial Clock) and ***GP4*** (SDA - Serial Data), then create an instance that represents the PAJ7620 sensor to allow the user to interact with it.

Main Loop

```
7  while True:
8      gesture = sensor.gesture()
9      if gesture & sensor.UP:
10         print("up")
11     if gesture & sensor.DOWN:
12         print("down")
13     if gesture & sensor.LEFT:
14         print("left")
15     if gesture & sensor.RIGHT:
16         print("right")
17     if gesture & sensor.CW:
18         print("clockwise")
19     if gesture & sensor.CCW:
20         print("counter-clockwise")
```

Line 8: Calls the sensor driver, **sensor.gesture()** to read the current gesture state.

Line 9 - 20: gesture & sensor.UP performs a bitwise AND between the gesture value and the sensor.UP value. Print "up" at the shell console if the gesture received is **TRUE**.

This pattern is repeated for DOWN, LEFT, RIGHT, CW (clockwise) and CCW (counter-clockwise) and prints the results in the shell console.

Activity with AI

Prompt Framework: OLED Display

Task: Use hand gestures to control the text movement on the OLED display.

Step 1: Teach the AI

Copy and paste your working code from **Page 53** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Here is my gesture code:
<PASTE YOUR CODE FROM PAGE 53>

Here is my OLED display code:
<PASTE YOUR CODE FROM PAGE 48>"

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

Prompt #2:
"Combine the gesture sensor and OLED display.
Move the text "EDU PICO!" on the screen according to my hand gestures:
Swipe Up to move text up
Swipe Down to move text down
Swipe Left to move text left
Swipe Right to move text right"

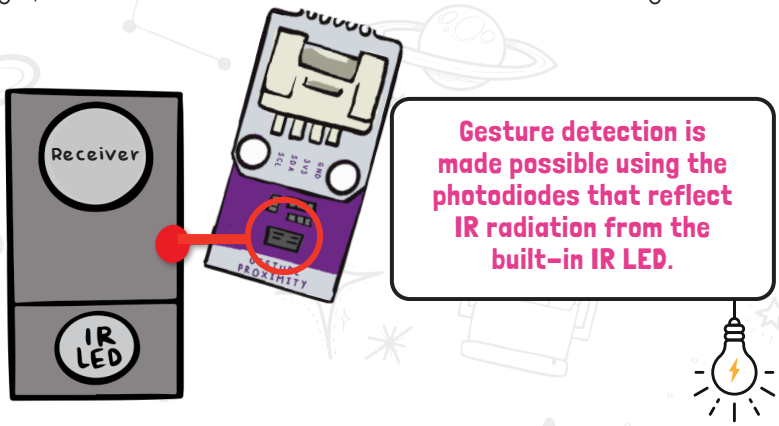
Step 3: Check the Answer

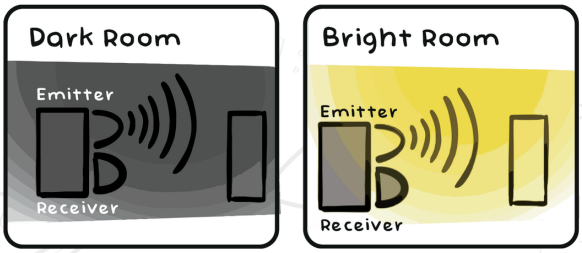
Copy and paste the code in Thonny, then run the code. Check if the text "EDU PICO!" moves on the OLED display in the same direction as your hand movement.

THE MORE YOU KNOW

PAJ7620 Gesture Sensor

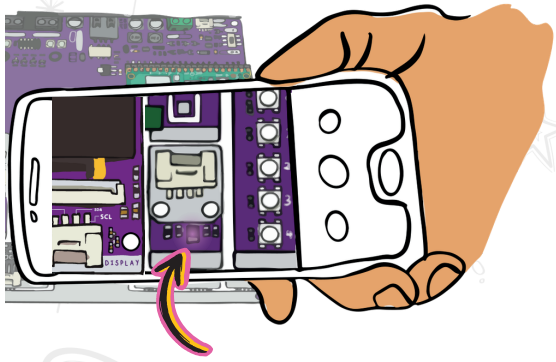
Imagine the PAJ7620 as a little camera with tiny sensors that can detect infrared light (which we can't see with our eyes). When you move your hand or an object in front of the sensor, the invisible light reflects on the object, allowing the sensor to detect the reflected infrared light, almost like a secret handshake that can't be seen through our naked eye.





The sensitivity of the sensor can be influenced by the room's brightness. In a well-lit room, there is a higher level of light reflection, which can be detected by the receiver.

Infrared light is invisible to the naked eye, however, you can view infrared light by simply looking at the infrared LED through a phone's rear camera. This is especially useful when you need to check whether the PAJ7620 sensor is still functioning properly.



3 GOOD HEALTH AND WELL-BEING

PROLOGUE: GESTURE REACTION GAME



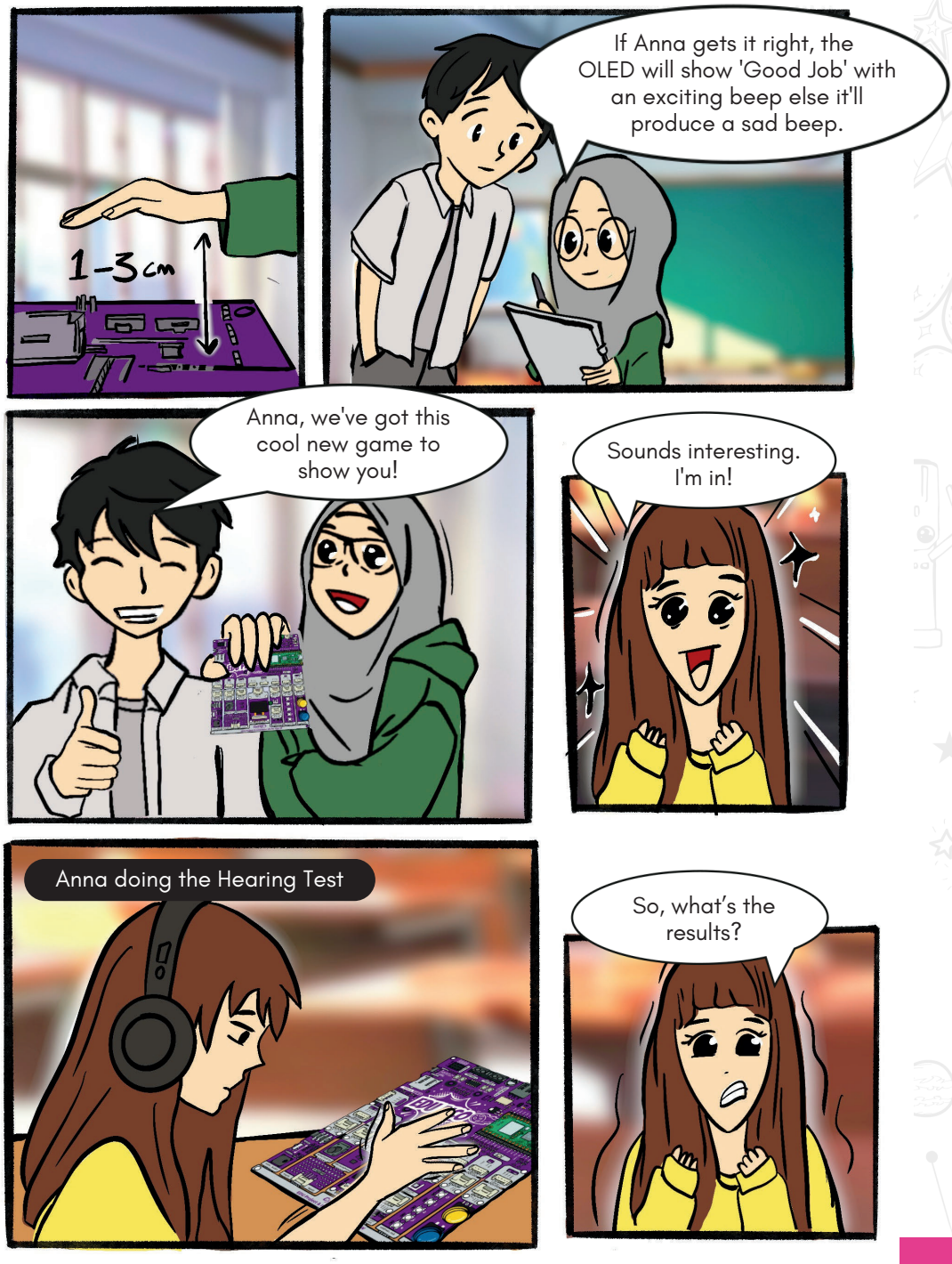
PROLOGUE: GESTURE REACTION GAME

3 GOOD HEALTH AND WELL-BEING



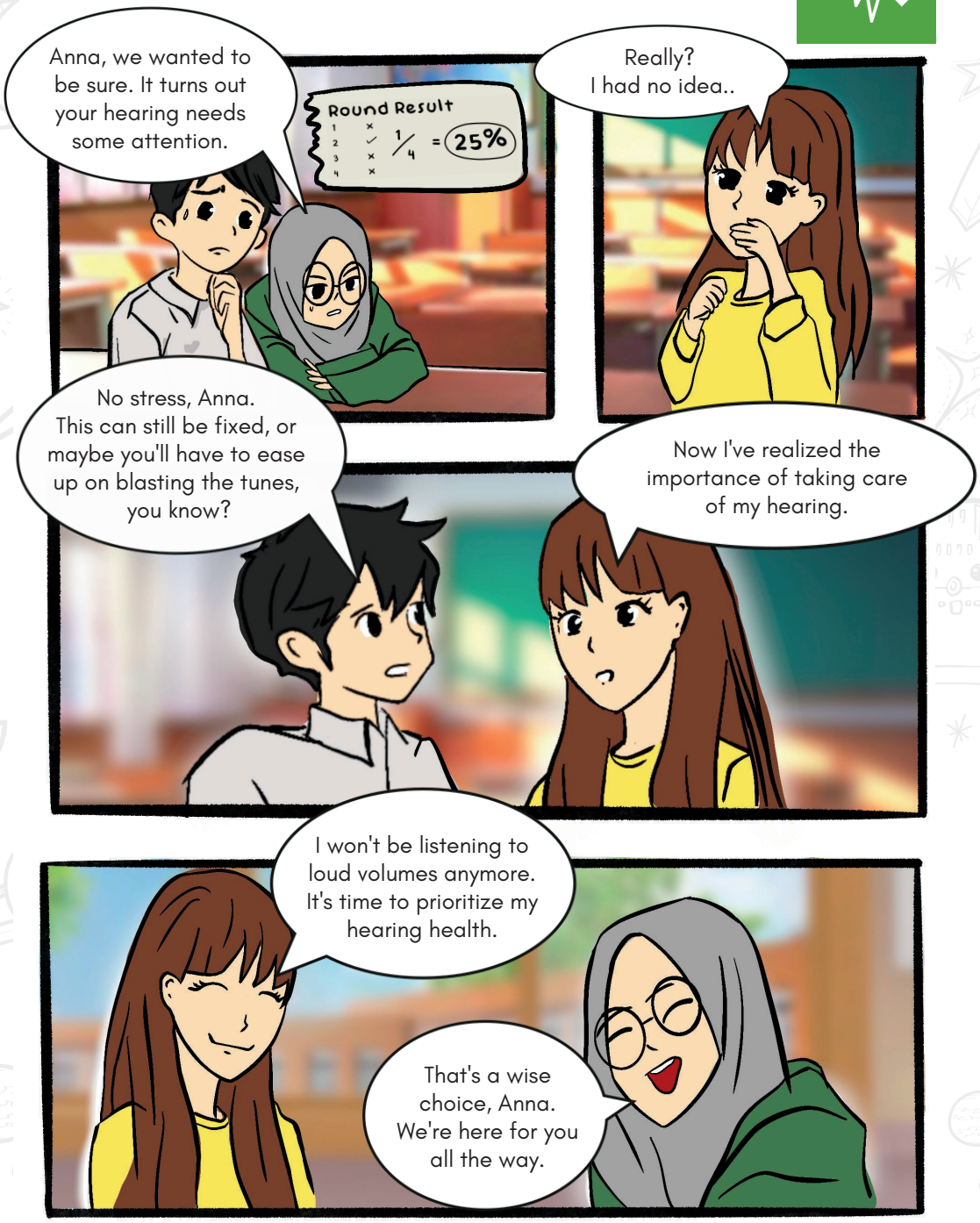
3 GOOD HEALTH AND WELL-BEING

PROLOGUE: GESTURE REACTION GAME



PROLOGUE: GESTURE REACTION GAME

3 GOOD HEALTH AND WELL-BEING



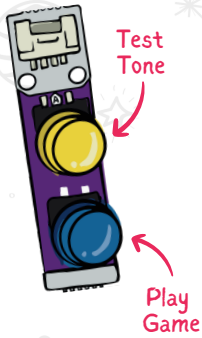
Gesture Reaction Game

Do-Re-Mi-Fa Arcade

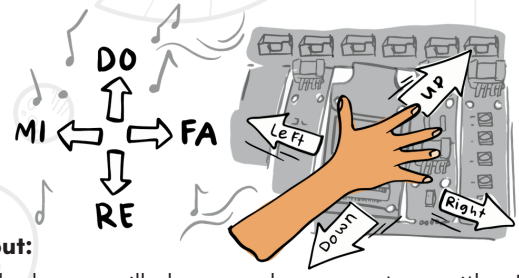
Let's build our first arcade game. But this won't be your typical game, instead, this is going to help players improve their hand-ear coordination. In this project, we'll learn to integrate OLED display, buzzer, and PAJ7620 gesture sensor, so be prepared for a slight increase in difficulty. Ready? Let's get started!

How Does This Activity Work?

- **Libraries:** board, digitalio, time, simpleio, busio, edupico2_paj7620 adafruit_ssd1306, random, font5x8.bin.
- **OLED & PAJ7620 I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Audio / Buzzer Configuration:** Buzzer to ***GP21***.
- **Button Configuration:**
  - button_tone (Yellow) to ***GP0*** as digital input.
  - button_start (Blue) to ***GP1*** as digital input.



- **Input:**
  - Pressing Button A will activate the test tone, playing **Do Re Mi Fa** in sequence.
  - Pressing Button B will start the game with a "Ready" and "Swipe!" countdown.
  - The player has to react to the random tone with either UP, DOWN, LEFT or RIGHT hand gestures.





- **Output:**
  - The buzzer will play a random game tone, either DO, RE, MI, or FA.
  - The OLED will print "Try again" with a sad beep if the player gives a wrong answer.
  - Getting the correct tone gesture will produce an exciting beep with a "Good Job" displayed on the OLED.

Code

PROJECT_HEARING_REACTION_GAME.py

```
1  import board, digitalio, time, simpleio, busio, random
2  import adafruit_ssd1306, edupico2_paj7620
3
4  tone_map = {"Up": 261, "Down": 293, "Left": 329, "Right": 349}
5  tone_keys = ["Up", "Down", "Left", "Right"]
6
7  buzzer = board.GP21
8  button_tone = digitalio.DigitalInOut(board.GP0)
9  button_start = digitalio.DigitalInOut(board.GP1)
10 button_tone.direction = digitalio.Direction.INPUT
11 button_start.direction = digitalio.Direction.INPUT
12
13 i2c = busio.I2C(board.GP5, board.GP4)
14 sensor = edupico2_paj7620.PAJ7620(i2c)
15 oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
16
17 def gamestart():
18     oled.fill(0)
19     oled.text("Ready", 50, 25, 1)
20     oled.show()
21     time.sleep(1)
22     oled.text("Swipe!", 50, 40, 1)
23     oled.show()
24
25 def get_gesture():
26     gesture = sensor.gesture()
27     if gesture & sensor.UP:
28         return "Up"
29     elif gesture & sensor.DOWN:
30         return "Down"
31     elif gesture & sensor.LEFT:
32         return "Left"
33     elif gesture & sensor.RIGHT:
34         return "Right"
35     return None
36
37 def flush_gesture():
38     sensor.gesture()
39     time.sleep(0.1)
40
41 while True:
42     oled.fill(0)
43     oled.text("Button A: Test Tone", 10, 25, 1)
44     oled.text("Button B: Play Game", 10, 40, 1)
45     oled.show()
```

Single Tab

The code from line 4 to 15 configures all the necessary stuff to make your project work!



```
    if not button_tone.value:
        oled.fill(0)
        oled.text("Playing Test Tone", 15, 35, 1)
Double  oled.show()
Tab     for key in tone_keys:
            simpleio.tone(buzzer, tone_map[key], 0.3)

    if not button_start.value:
        gamestart()
        tone_gesture = random.choice(list(tone_map))
        selected_tone = tone_map[tone_gesture]
        simpleio.tone(buzzer, selected_tone, 1)
        flush_gesture()

        while True:
            gesture_value = get_gesture()
Triple  if not gesture_value:
Tab         continue
            if gesture_value == tone_gesture:
                oled.fill(0)
                oled.text("Good Job!", 35, 30, 1)
                oled.show()
                for i in range(3):
                    simpleio.tone(buzzer, 1100, 0.1)
                    time.sleep(0.1)
                break
            else:
                oled.fill(0)
                oled.text("Try Again!", 35, 30, 1)
                oled.show()
                for i in range(3):
                    simpleio.tone(buzzer, 100, 0.1)
                    time.sleep(0.1)
```

Click the Green Button to run the code and Red Button to stop.

What the Code Does

Libraries

```
1 import board, digitalio, time, simpleio, busio, random
2 import adafruit_ssd1306, edupico2_paj7620
```

Line 1: To generate a random sequence of **DO RE MI FA** tones, you'll need to use the **random** library. This library enables you to generate random numbers for your code.
Line 2: We will also need the **edupico2_paj7620** library in order to operate the PAJ7620 gesture sensor.

Constant Variable and Dictionary

```
4  tone_map = {"Up": 261, "Down": 293, "Left": 329, "Right": 349}
5  tone_keys = ["Up", "Down", "Left", "Right"]
```

Line 4: **tone_map** matches with the gestures (Up/Down/Left/Right) to buzzer tone frequencies (Do-Re-Mi-Fa).

Line 5: **tone_keys** is a fixed order list, used when looping through tones in a specific order. In this case, the order starts from Do, Re, Mi, and Fa.

Define a Custom Function

gamestart(), get_gesture(), flush_gesture() functions

```
17 def gamestart():
18     oled.fill(0)
19     oled.text("Ready", 50, 25, 1)
20     oled.show()
21     time.sleep(1)
22     oled.text("Swipe!", 50, 40, 1)
23     oled.show()
24
25 def get_gesture():
26     gesture = sensor.gesture()
27     if gesture & sensor.UP:
28         return "Up"
29     elif gesture & sensor.DOWN:
30         return "Down"
31     elif gesture & sensor.LEFT:
32         return "Left"
33     elif gesture & sensor.RIGHT:
34         return "Right"
35     return None
36
37 def flush_gesture():
38     sensor.gesture()
39     time.sleep(0.1)
```

Line 17 - 23: Initializes and displays a "Ready" message on the OLED display followed by a "Swipe!" message after a 1 second delay.

Line 25 - 35: The **get_gesture()** function reads gesture from the sensor and checks which direction was detected. It returns "Up", "Down", "Left", "Right", or None depending on the signal picked up by the PAJ7620 sensor.

Line 37 - 39: The **flush_gesture()** function is used to quickly read and discard any leftover gesture signals. This is used to prevent false triggers before the player swipes. (Usually occur when starting a new game.)

Enter a Continuous Loop

Main Loop

```
41 while True:
42     oled.fill(0)
43     oled.text("Button A: Test Tone", 10, 25, 1)
44     oled.text("Button B: Play Game", 10, 40, 1)
45     oled.show()
```

Line 42: **oled.fill(0)** clears the OLED display by filling it with black (**0**).
Line 43 - 44: **oled.text**(" ", x, y, colour) displays text on the OLED display at coordinates (x, y). The text colour can be either filled with white (**1**) or black (**0**).
Line 45: **oled.show()** updates the OLED display to make the text visible.

button_tone (Button A)

```
47     if not button_tone.value:
48         oled.fill(0)
49         oled.text("Playing Test Tone", 15, 35, 1)
50         oled.show()
```

Line 47 - 50: Checks if the button connected to **button_tone** is pressed. If the button is pressed, this line clears the OLED display and prints "Playing Test Tone".

button_tone (Button A)

```
51         for key in tone_keys:
52             simpleio.tone(buzzer, tone_map[key], 0.3)
```

Line 51 - 52: Starts a **for loop** that iterates through a list of musical tone frequencies **DO RE MI FA** (4 cycles) with a **0.3** second delay after each tone.

What is for..loop?

A **for loop** is used to iterate or repeat over a sequence, in this case, the loop iterate 4 cycles from Do, Re, Mi to Fa.

During the first iteration, **tone_freq** will take on the value **FREQ_DO** = **261 Hz**.
The frequency values in the code should look like this in the first iteration:

simpleio.tone(buzzer, **261**, **0.3**)

Second iteration, with **FREQ_RE** = **293 Hz**
Third iteration with **FREQ_MI** = **329 Hz**
Fourth iteration with **FREQ_FA** = **349 Hz**

Check if Start Button (Button B) is Pressed

```
54      if not button_start.value:
55          gamestart()
56          tone_gesture = random.choice(list(tone_map))
57          selected_tone = tone_map[tone_gesture]
58          simpleio.tone(buzzer, selected_tone, 1)
59          flush_gesture()
```

Line 54 - 55: Checks if Button B is pressed. If it's pressed, it calls the **gamestart()** function (defined at line 17 - 23) to display "Ready" and "Go!" messages.

Line 56 - 58: Randomly selects one gesture (Up, Down, Left, Right), then search for its frequency from **tone_map** (Line 4) and plays it on the buzzer. Example: If "Left" is chosen, then play **329 Hz**.

Line 59: Calls **flush_gesture** (Line 37 - 39) to clear old sensor readings before waiting for a real swipe.

Continuous Game Loop

```
61          while True:
62              gesture_value = get_gesture()
63              if not gesture_value:
64                  continue
```

Line 62: **gesture_value** continuously read the gesture sensor.
Line 63 - 64: If no gesture is detected, the code will continue to loop. If a gesture is detected, stores "Up", "Down", "Left", or "Right" into the variable **gesture_value**.

Continuous Game Loop - Correct Gesture

```
65              if gesture_value == tone_gesture:
66                  oled.fill(0)
67                  oled.text("Good Job!", 35, 30, 1)
68                  oled.show()
69                  for i in range(3):
70                      simpleio.tone(buzzer, 1100, 0.1)
71                      time.sleep(0.1)
72                  break
```

Line 65 - 68: If gesture matches the chosen tone direction, the OLED will display "Good Job!", indicating the player has given the correct gesture.
Line 69 - 71: The buzzer will then proceed to play a victory beep 3 times at **1100** Hz.
Line 72: The **break** exits the code out of the game loop and returns to main menu.

Continuous Game Loop - Wrong Gesture

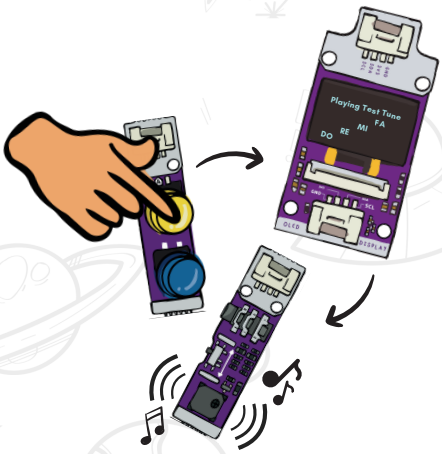
```
73              else:
74                  oled.fill(0)
75                  oled.text("Try Again!", 35, 30, 1)
76                  oled.show()
77                  for i in range(3):
78                      simpleio.tone(buzzer, 100, 0.1)
79                      time.sleep(0.1)
```

Line 73 - 76: If swipe gesture doesn't match with **tone_gesture**, then the OLED will display "Try Again!". Indicating that the player has provided a false gesture.
Line 77 - 79: Then buzzer will then play a low error beep 3 times at **100** Hz with a **0.1** seconds interval.

CHALLENGES

"DO RE MI FA" TONE TESTER

Since we are building a reaction arcade game with an OLED display, why not challenge ourselves to make it even better? Let's provide the player with more information about the test tones when they press button A.



QUICK TIPS

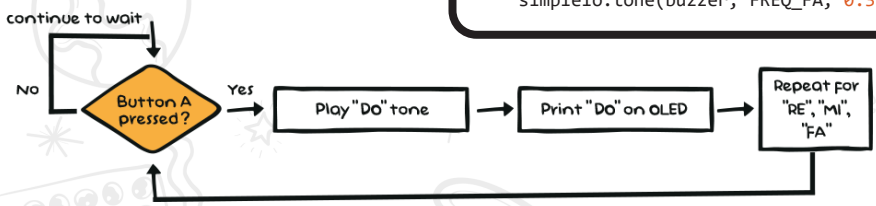
```
if not button_tone.value:
    oled.fill(0)
    oled.text("Playing Test Tune", 10, 10, 1)
    oled.show()

    oled.text("DO", 20, 55, 1)
    oled.show()
    simpleio.tone(buzzer, FREQ_DO, 0.3)

    oled.text("RE", 40, 45, 1)
    oled.show()
    simpleio.tone(buzzer, FREQ_RE, 0.3)

    oled.text("MI", 60, 35, 1)
    oled.show()
    simpleio.tone(buzzer, FREQ_MI, 0.3)

    oled.text("FA", 80, 25, 1)
    oled.show()
    simpleio.tone(buzzer, FREQ_FA, 0.3)
```



CHAPTER 4

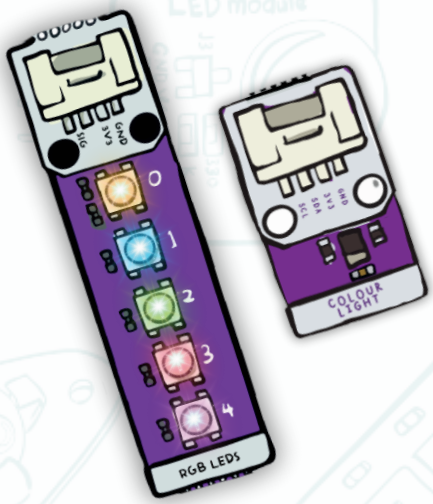
Colour Detection Game

RGB LEDs & Colour Sensor

Introduction to RGB LEDs

Introduction to Colour Sensor

Project: Colour Detection Game



In this chapter, we will explore two incredible components on the EDU PICO: RGB LEDs and the OPT4048 colour sensor module.

Imagine creating a game that can detect colours and respond with dazzling light displays. By the end of this chapter, you'll have the knowledge and skills to design and build your very own colour detection game.

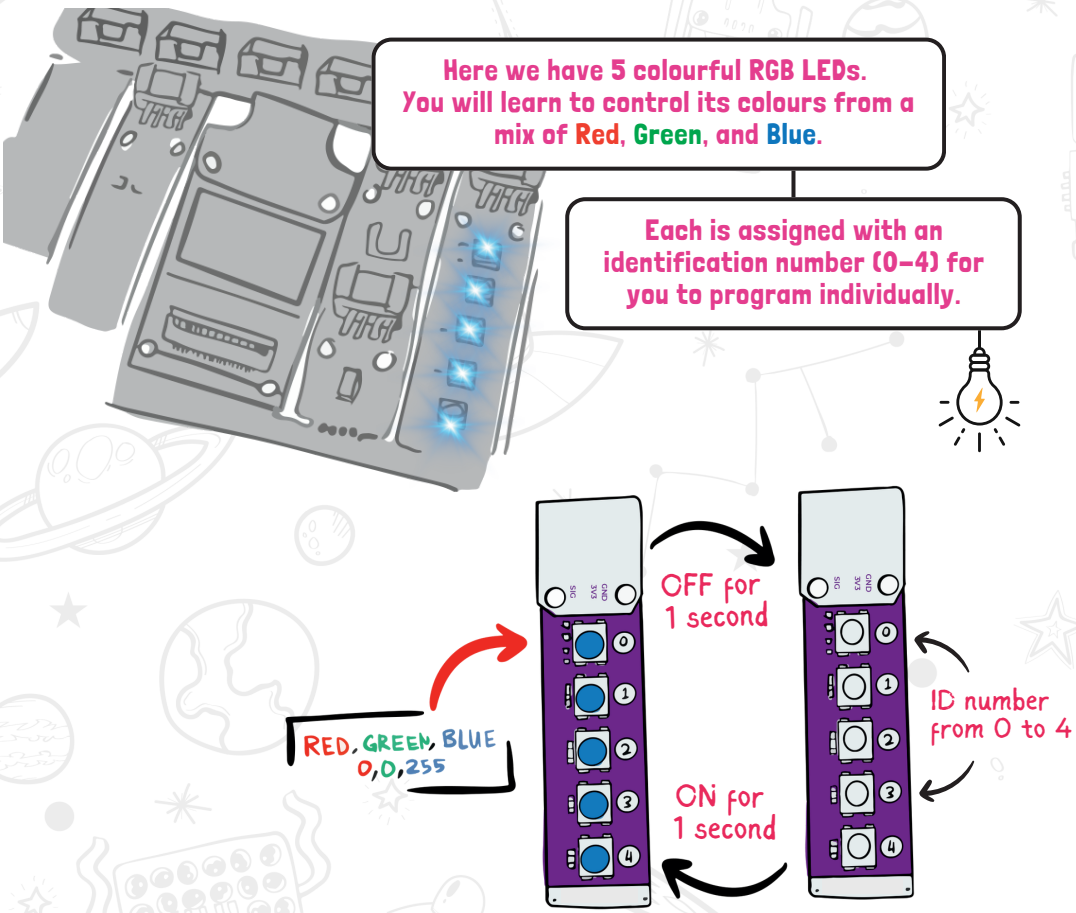
We'll begin by learning what RGB LEDs are and how they can be programmed to produce a mesmerizing rainbow of colours. Then, we'll introduce the OPT4048 colour sensor, a powerful tool that can 'see' colours just like you do.

Introduction to RGB LEDs

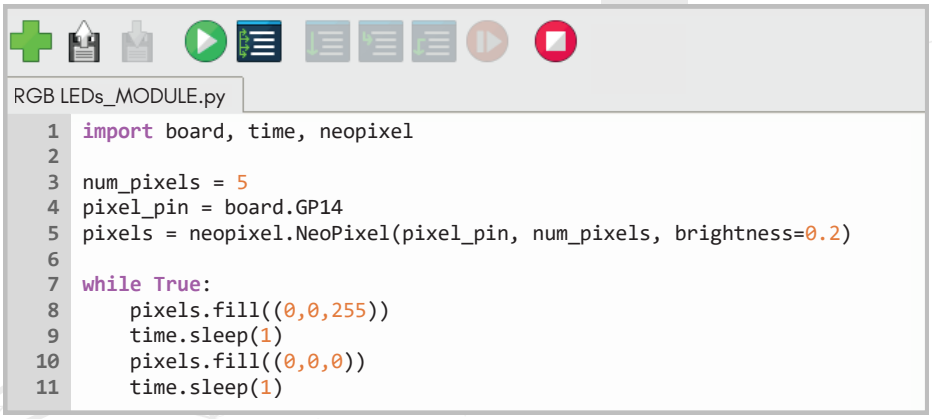
In this introduction, you will learn how to control the RGB LEDs module to create dazzling displays of colour and patterns! Once you're done with the sample code, make sure to check out the last section where you will learn how to mix various colours on the RGB LEDs.

How Does This Activity Work?

- **Libraries:** board, time, neopixel.
- **RGB LEDs Configuration:** num_pixels = **5**, pixel_pin = ***GP14***.
- **Output:**
  - The RGB LEDs will repeatedly turn all LEDs on and off in blue colour, with a **1** second interval.



Code



RGB LEDs_MODULE.py

```
1  import board, time, neopixel
2
3  num_pixels = 5
4  pixel_pin = board.GP14
5  pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.2)
6
7  while True:
8      pixels.fill((0,0,255))
9      time.sleep(1)
10     pixels.fill((0,0,0))
11     time.sleep(1)
```

Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Import Necessary Libraries

Libraries

```
1  import board, time, neopixel
```

neopixel library provides functionality to control RGB LEDs or WS2812 RGB LED modules.

Initialize Hardware Components

RGB LEDs Pin Configuration

```
3  num_pixels = 5
4  pixel_pin = board.GP14
5  pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.2)
```

Line 5: Initializes the RGB LEDs object to **pixel_pin** to (***GP14***), **num_pixels** to **5**, and the brightness parameter to **0.2** (20% brightness).

Enter a Continuous Loop

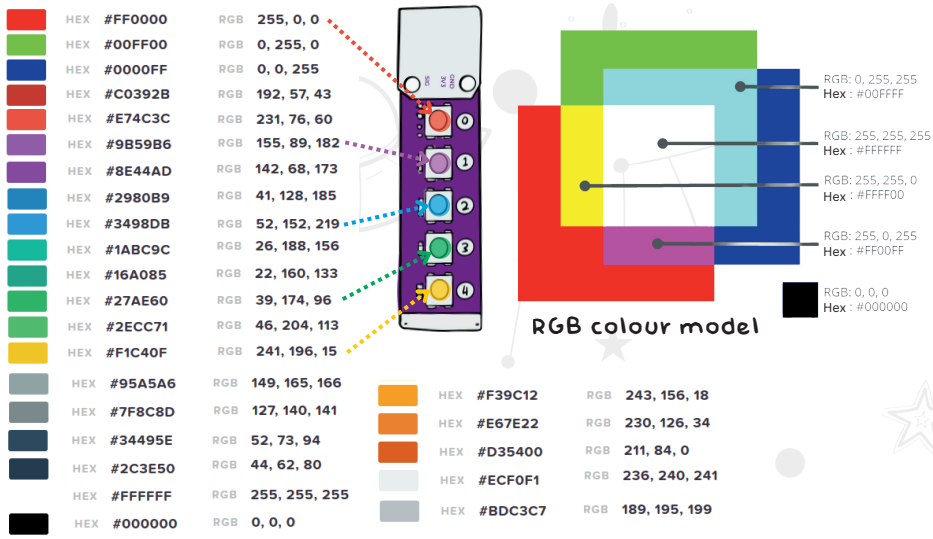
Light Up Pixel Continuously

```
while True:
    pixels.fill((0,0,255))
    time.sleep(1)
    pixels.fill((0,0,0))
    time.sleep(1)
```

The **pixels.fill**((0, 0, 255)) is used to fill all the LEDs with a specific colour. In this case, (0, 0, 255) represents full blue and (0, 0, 0) represents black or no colour.

THE MORE YOU KNOW

The RGB LEDs, or in technical terms, the WS2812 integrated light source, is a full red, green, and blue LED that is embedded with a driver chip within each LED. This allows the RGB LEDs to be individually addressable, allowing us to program each LED according to the identification number for different colours.



You can mix Red, Green and Blue to get the colour you want according to the colour code. Combining all 3 will create White light! Cool right?

Activity with AI

Prompt Framework: RGB LED

Task: Make the RGB LEDs show a running rainbow colour effect.

Step 1: Teach the AI

Copy and paste your working code from **Page 70** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Here is my current code:
<PASTE YOUR CODE FROM PAGE 70>"

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

Prompt #2:
"Make the 5 RGB LEDs display a running rainbow effect that smoothly moves across all pixels in a loop."

Step 3: Check the Answer

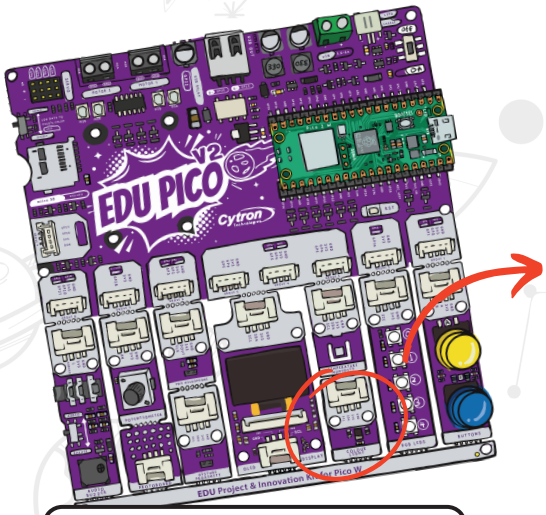
Copy and paste the code in Thonny, then run the code. Check if the RGB LEDs lights up multiple colours at once, to form a rainbow pattern continuously.

Introduction to Colour Sensor

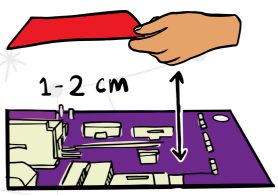
In this chapter, we will explore a sensor that can measure light. Light can be described in different ways, from how bright it is to the colours we see when it reflects off objects. By learning how to detect brightness and colour, we can begin to understand how machines perceive the world around them too.

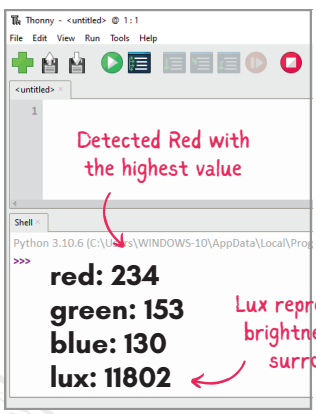
How Does This Activity Work?

- **Libraries:** board, time, busio, neopixel, adafruit_opt4048.
- **RGB LEDs Configuration:** num_pixels = **5**, pixel_pin = ***GP14***.
- **OPT4048 I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Input:**
  - Hold the colour cards above the colour sensor.
- **Output:**
  - RGB LEDs light up in white colour at 20% brightness.
  - The colour sensor reads and prints the amount of red (r), green (g), blue (b), and clear (c) light values with a **1** second interval.



Best Colour Detection Distance





Keep in mind that the brightness of your surrounding can influence the outcome.





Code



COLOUR_SENSOR_MODULE.py

```
1  import time, board, busio, neopixel
2  from adafruit_opt4048 import OPT4048, Mode
3
4  num_pixels = 5
5  pixel_pin = board.GP14
6  pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.2)
7
8  i2c = busio.I2C(board.GP5, board.GP4)
9  sensor = OPT4048(i2c)
10 sensor.mode = Mode.CONTINUOUS
11
12 while True:
13     pixels.fill((255, 255, 255))
14     R, G, B, LUX = sensor.rgb
15     print(f"red: {R}, green: {G}, blue: {B}, lux: {LUX:.2f}")
16     time.sleep(1)
```

```
red: 183, green: 147, blue: 106, lux: 80.56
red: 183, green: 147, blue: 106, lux: 80.65
red: 182, green: 147, blue: 106, lux: 80.77
red: 183, green: 146, blue: 106, lux: 80.66
red: 183, green: 146, blue: 107, lux: 80.66
red: 183, green: 147, blue: 106, lux: 80.73
red: 182, green: 147, blue: 106, lux: 80.91
```

Click the Green Button  to run the code and Red Button  to stop.

What the Code Does </>

Libraries

```
1 import time, board, busio, neopixel
2 from adafruit_opt4048 import OPT4048, Mode
```

Line 2: The **adafruit_opt4048** provides functionality to work with the OPT4048 light and colour sensor.

RGB LEDs and OPT4048 Initialization

```
4  num_pixels = 5
5  pixel_pin = board.GP14
6  pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.2)
7
8  i2c = busio.I2C(board.GP5, board.GP4)
9  sensor = OPT4048(i2c)
10 sensor.mode = Mode.CONTINUOUS
```

Line 4 - 6: Sets up the RGB LEDs strip with **5** LEDs connected to GPIO pin ***GP14***. The brightness of the RGB LEDs is set to 20% (**0.2**).

Line 8: The script initializes the I2C communication using pins ***GP5*** and ***GP4***. Connecting the OPT4048 sensor over that I2C bus.

Line 9: The **sensor** is now ready to read RGB and light intensity data.
Line 10: This sets the sensor into a continuous mode.

Main Loop

```
12 while True:
13     pixels.fill((255, 255, 255))
14     R, G, B, LUX = sensor.rgb
15     print(f"red: {R}, green: {G}, blue: {B}, lux: {LUX:.2f}")
16     time.sleep(1)
```

Line 13: Light up the RGB LEDs in white. By doing so, it creates a bright surrounding next to the OPT4048 sensor which will allow the sensor to detect the object's colour clearly and more accurately.

Line 14: Reads the raw RGB values from the OPT4048 sensor and the lux (brightness), providing separate variables for red **(R)**, green **(G)**, blue **(B)**, and lux **(LUX)** channels.

Line 15: Prints the RGB values in a formatted string (f-string) making it more human-readable. In this case, the {R}, {G}, {B}, and {LUX:.2f} are the placeholders within the f-string. They are used to indicate where the values of the variables r, g, b and, lux should be inserted within the text.

Activity with AI

Prompt Framework: Colour Sensor

Task: Use the colour sensor to detect red, green, or blue and show that colour on the RGB LED module.

Step 1: Teach the AI

Copy and paste your working code from **Page 74** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Here is my colour sensor code:
<PASTE YOUR CODE FROM PAGE 74>

Here is the code for RGB LED:
<PASTE YOUR CODE FROM PAGE 70>"

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

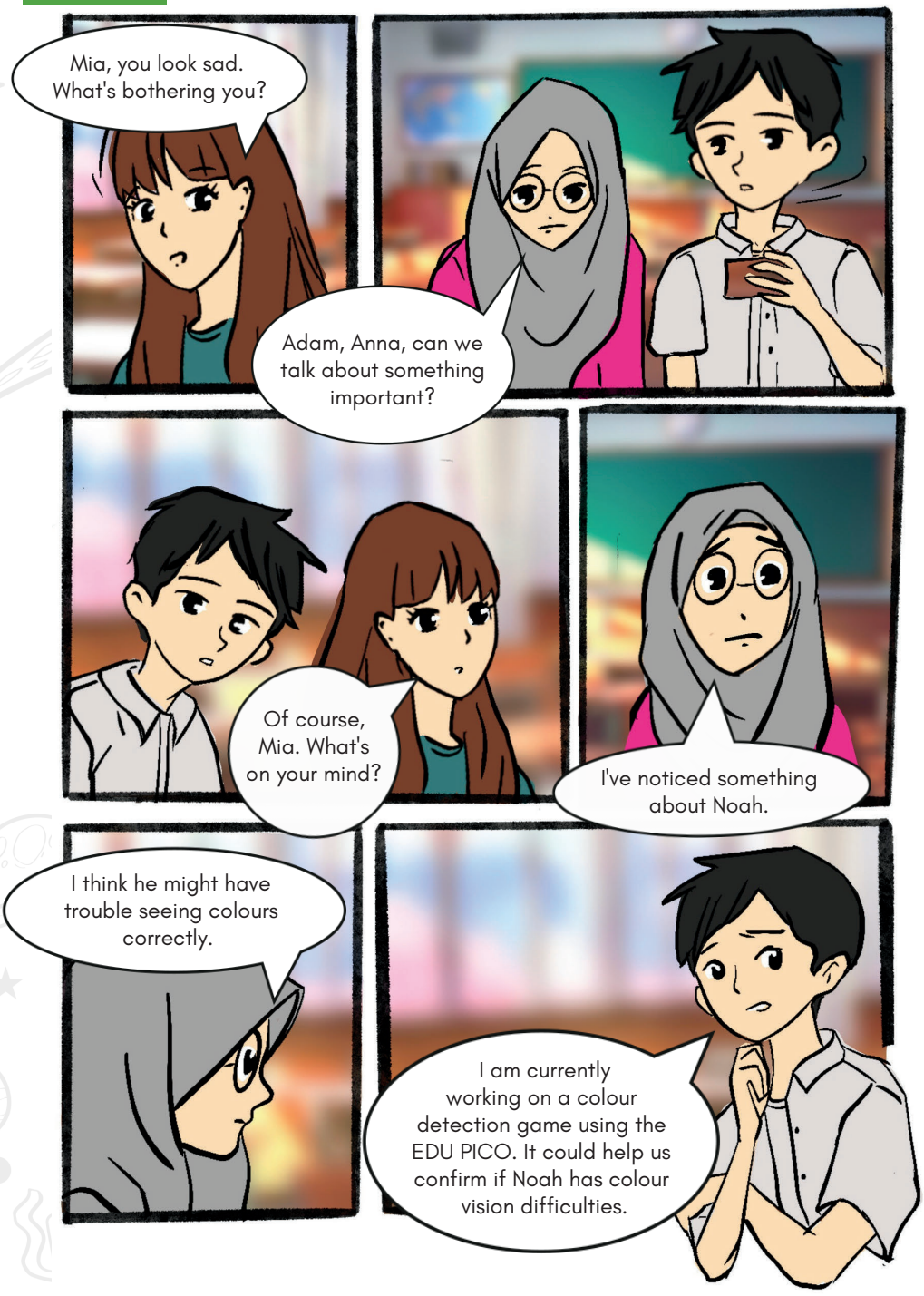
Prompt #2:
"Use the colour sensor to detect the amount of red, green, and blue light. Make the RGB LEDs show the same colour detected by the sensor."

Step 3: Check the Answer

Copy and paste the code in Thonny, then run the code. Check if the RGB LEDs colour changed according to the coloured objects that is placed in front of the colour sensor.

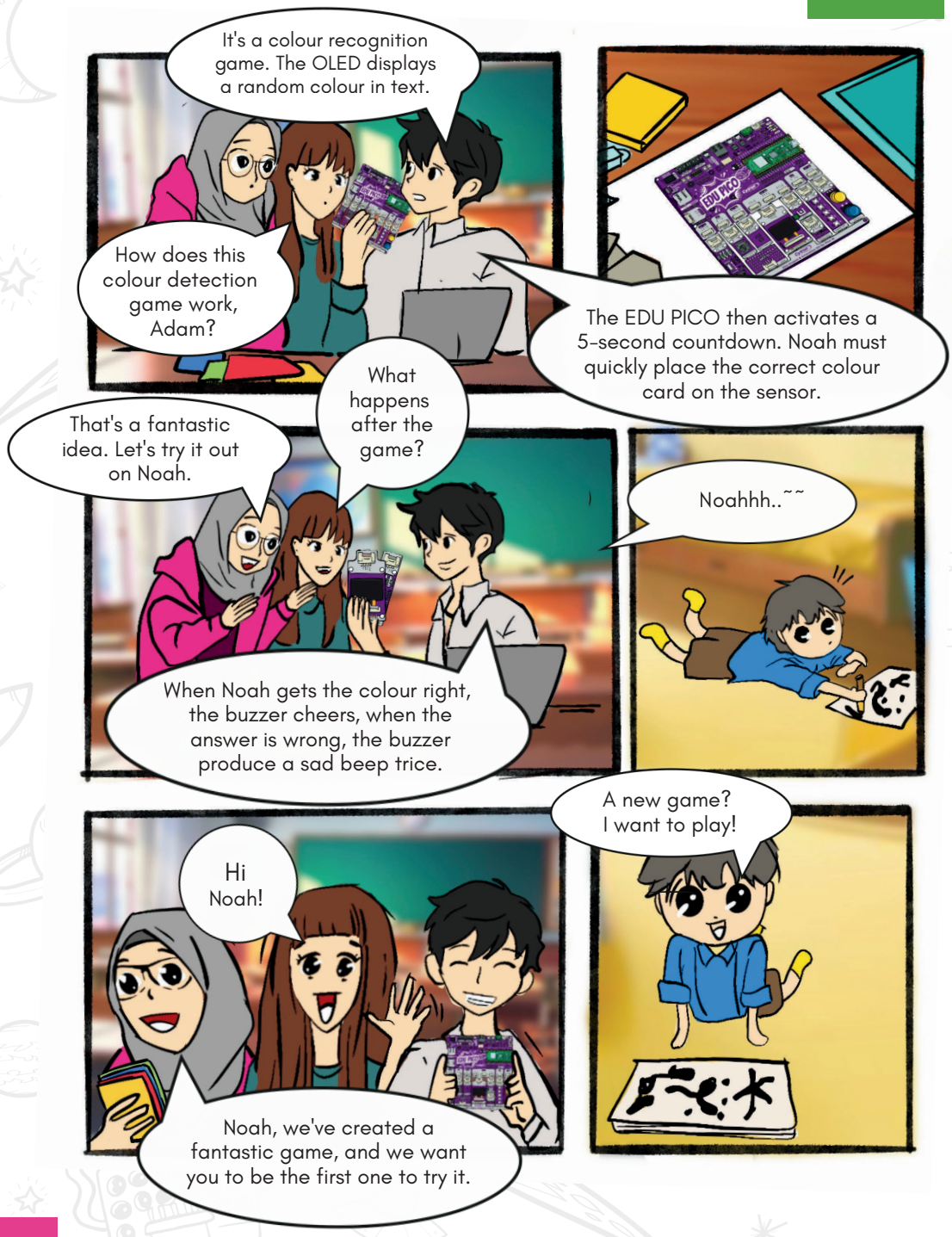
3 GOOD HEALTH AND WELL-BEING

PROLOGUE: COLOUR DETECTION GAME



PROLOGUE: COLOUR DETECTION GAME

3 GOOD HEALTH AND WELL-BEING



3 GOOD HEALTH AND WELL-BEING

PROLOGUE: COLOUR DETECTION GAME



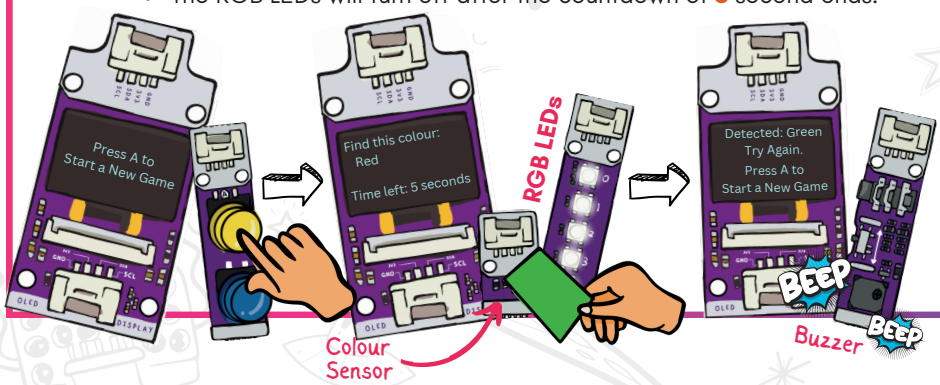
Colour Detection Game

Colour Blindness Tester

Did you know that approximately 300 million people worldwide have colour vision deficiency, also known as colour blindness? This condition can make it difficult to distinguish between colours, affecting a person's education, academic performance, and even career choices. In this project, we will learn how to build a basic colour detection game by integrating colour sensor, RGB LEDs, buzzer, button and OLED display.

How Does This Activity Work?

- **Libraries:** board, digitalio, time, simpleio, neopixel, busio, random, adafruit_ssd1306, adafruit_opt4048.
- **OLED & OPT4048 I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Audio / Buzzer Configuration:** Buzzer to ***GP21***.
- **RGB LEDs Configuration:** num_pixels = ***5***, pixel_pin = ***GP14***.
- **Buttons Configuration:** button_start (Yellow) to ***GP0*** as digital input.
- **Input:**
  - Press Button A to start the game with a ***5*** second countdown.
  - The player must quickly place the correct colour card above the colour sensor once the RGB LEDs white light turns on.
- **Output:**
  - The OLED will print a random colour in text to the player.
  - If the player gets the colour correct, the buzzer will play an exciting tone with a "Well Done!" message printed on the OLED display.
  - If the player got it wrong, the buzzer would beep 3 times, and the OLED would proceed to print "Try Again".
  - The RGB LEDs will turn off after the countdown of ***5*** second ends.



Code



PROJECT_COLOUR_DETECTION_GAME.py

```
import board, digitalio, time, simpleio, neopixel, busio, random
from adafruit_opt4048 import OPT4048, Mode
import adafruit_ssd1306

i2c = busio.I2C(board.GP5, board.GP4)
sensor = OPT4048(i2c)
sensor.mode = Mode.CONTINUOUS

buzzer = board.GP21
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)

button_start = digitalio.DigitalInOut(board.GP0)
button_start.direction = digitalio.Direction.INPUT

num_pixels = 5
pixel_pin = board.GP14
pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.2)

MELODY_NOTE = [523, 659, 784, 0, 659, 784]
MELODY_DURATION = [0.12, 0.12, 0.12, 0.1, 0.12, 0.2]

colours = ["Red", "Green", "Blue", "Yellow", "Purple"]

oled.fill(0)
oled.text("Press A to", 35, 25, 1)
oled.text("Start a New Game", 15, 35, 1)
oled.show()

while True:
    pixels.fill((0, 0, 0))
    while button_start.value:
        pass
    oled.fill(0)
    oled.show()
    random_colour = random.choice(colours)
    oled.text("Find this colour:", 10, 10, 1)
    oled.text("{}".format(random_colour), 20, 25, 1)
    oled.show()
    time.sleep(2)
    pixels.fill((255, 255, 255))
    oled.text("Time left:   seconds", 5, 50, 1)
```

```
    for countdown in range(5, -1, -1):
        oled.fill_rect(70, 50, 10, 7, 0)
        oled.text("{}".format(countdown), 70, 50, 1)
        oled.show()
        time.sleep(1)

    R, G, B, LUX = sensor.rgb
    if (R > G) and (R > B):
        if G > B:
            detected_colour = "Yellow"
        else:
            detected_colour = "Red"
    elif (B > R) and (B > G):
        if R > G:
            detected_colour = "Purple"
        else:
            detected_colour = "Blue"
    elif (G > R) and (G > B):
        detected_colour = "Green"

    oled.fill(0)
    oled.text("Detected: {}".format(detected_colour), 10, 5, 1)
    oled.show()

    if detected_colour == random_colour:
        oled.text("Well Done!", 35, 25, 1)
        oled.text("Press A to", 35, 45, 1)
        oled.text("Start a New Game", 15, 55, 1)
        oled.show()
        for i in range(len(MELODY_NOTE)):
            simpleio.tone(buzzer, MELODY_NOTE[i], MELODY_DURATION[i])
    else:
        oled.text("Try Again.", 35, 25, 1)
        oled.text("Press A to", 35, 45, 1)
        oled.text("Start a New Game", 15, 55, 1)
        oled.show()
        for i in range(3):
            simpleio.tone(buzzer, 100, 0.1)
            simpleio.tone(buzzer, 0, 0.1)
```

Click the Green Button  to run the code and Red Button  to stop.

What the Code Does

Import Necessary Libraries

Libraries

```
1 import board, digitalio, time, simpleio, neopixel, busio, random
2 from adafruit_opt4048 import OPT4048, Mode
3 import adafruit_ssd1306
```

Import a total of 9 libraries for the following project to work. Libraries include **adafruit_opt4048** for OPT 4048 colour sensor, **adafruit_ssd1306** for OLED display, **neopixel**, and **random** for generating a random number.

Initialize Hardware Components

Initialization

```
5
.
.
18
19 MELODY_NOTE = [523, 659, 784, 0, 659, 784]
20 MELODY_DURATION = [0.12, 0.12, 0.12, 0.1, 0.12, 0.2]
21
22 colours = ["Red", "Green", "Blue", "Yellow", "Purple"]
23
24 oled.fill(0)
25 oled.text("Press A to", 35, 25, 1)
26 oled.text("Start a New Game", 15, 35, 1)
27 oled.show()
```

Line 5 - 17: Initializes OPT 4048 colour sensor, OLED display, buzzer (***GP21***), button A (***GP0***) and RGB LEDs module (***GP14***).

Line 19: Defines a sequence of musical notes in frequency (Hz) for a melody. The melody will be played with the player guessing the correct colour.

Line 20: The array defines the duration of each note in the corresponding position in the **MELODY_NOTE** array. For example: **0.12** means that a note is played for **0.12** second.

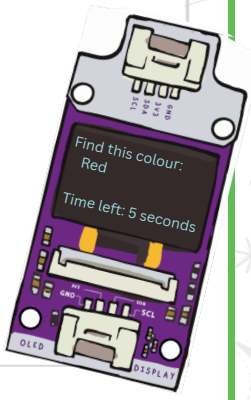
Line 22: The **colours** array contains the names of different colours used in the game.

Line 24 - 27: Print an initial message on the OLED display to instruct the player to press button A to start a new game.

Enter a Continuous Loop

Prepare the Game

```
29 while True:
30     pixels.fill((0,0,0))
31     while button_start.value:
32         pass
33     oled.fill(0)
34     oled.show()
35     random_colour = random.choice(colours)
36     oled.text("Find this colour:", 10, 10, 1)
37     oled.text("{}".format(random_colour), 20, 25, 1)
38     oled.show()
39     time.sleep(2)
40     pixels.fill((255,255,255))
41     oled.text("Time left:   seconds", 5, 50, 1)
```



Line 30: Clear the RGB LEDs strip (turn off all pixels).

Line 31: Wait for **button_start** (Button A) to be pressed to start the game.

Line 33: Clear the OLED display to prepare for the game.

Line 35: Randomly select a colour from the **colours** array defined at **Line 22**.

Line 37: Prints the randomly selected colour (stored in the **random_colour** variable) on the OLED display.

Line 40: Light up all RGB LEDs in white to indicate the start of the game.

Line 41: Prints "Time left: seconds", where the blank space will be replaced with the seconds value in the later code.

Starting the Game - The Countdown

```
43     for countdown in range(5, -1, -1):
44         oled.fill_rect(70, 50, 10, 7, 0)
45         oled.text("{}".format(countdown), 70, 50, 1)
46         oled.show()
47         time.sleep(1)
```

Line 43: The for loop iterates through a range of numbers from **5** to **0**.

Line 44: This line clears a rectangular region on the OLED display at the coordinates of (**70**, **50**), and with a size of (**10** x **7**). The cleared space will be used to update the duration value in seconds located in between "Time left: { } seconds".

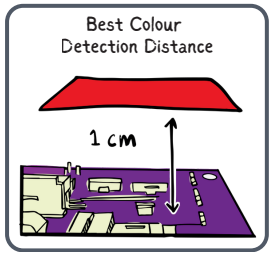
Line 45: This line displays the current value of the countdown as text on the OLED.

Line 47: This line introduces a **1** second interval to synchronize with the countdown.

Check the Colour Data Value

Detect Colour

```
49      R, G, B, LUX = sensor.rgb
50      if (R > G) and (R > B):
51          if G > B:
52              detected_colour = "Yellow"
53          else:
54              detected_colour = "Red"
55      elif (B > R) and (B > G):
56          if R > G:
57              detected_colour = "Purple"
58          else:
59              detected_colour = "Blue"
60      elif (G > R) and (G > B):
61          detected_colour = "Green"
62
63      oled.fill(0)
64      oled.text("Detected: {}".format(detected_colour), 10, 5, 1)
65      oled.show()
```



Line 49: Read colour data from the OPT4048 where R, G, B represents red, green, blue channel readings, and LUX for illuminance or brightness.

Line 50 - 61: The following lines determine the detected colour based on the RGB values by comparing them to one another:

- If R is largest:
  - If G > B → treat as Yellow (red+green mix).
  - Else → Red.
- If B is largest:
  - If R > G → Purple (red+blue mix).
  - Else → Blue.
- If G is largest → Green.

Line 63 - 65: Clear OLED and show the detected colour string near the top-left of the screen.

Compare Colour

```
67          if detected_colour == random_colour:
68              oled.text("Well Done!", 35, 25, 1)
69              oled.text("Press A to", 35, 45, 1)
70              oled.text("Start a New Game", 15, 55, 1)
71              oled.show()
72              for i in range(len(MELODY_NOTE)):
73                  simpleio.tone(buzzer, MELODY_NOTE[i], MELODY_DURATION[i])
74          else:
75              oled.text("Try Again.", 35, 25, 1)
76              oled.text("Press A to", 35, 45, 1)
77              oled.text("Start a New Game", 15, 55, 1)
78              oled.show()
79              for i in range(3):
80                  simpleio.tone(buzzer, 100, 0.1)
81                  simpleio.tone(buzzer, 0, 0.1)
```



Line 67: This condition checks if the detected colour is equal to the random colour chosen at the beginning of the game.

Line 68 - 73: If the detected colour matches the random colour, the OLED prints the text shown on the right, followed by a melody iterating through **MELODY_NOTE** and **MELODY_DURATION** arrays.

Line 74 - 81: If the detected colour doesn't match the random colour, the OLED will print "Try Again" and the buzzer will beep 3 times.

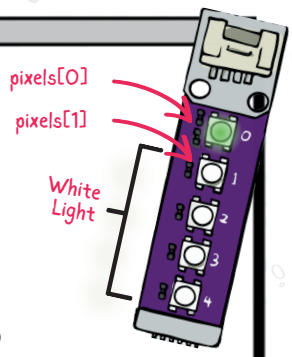
CHALLENGE: COLOUR HINT INDICATOR

In this challenge, you are required to add a hint function for the player while maintaining the flow of the game. Program only the first RGB LED (ID:0) to light up in the random colour similar to the colour text shown on the OLED. This helps players match colours more easily and sharpen their colour recognition skills. Give it a try!

QUICK TIPS

```
if random_colour == "Red":
    pixels[0] = (255, 0, 0)  # Red
elif random_colour == "Green":
    pixels[0] = (0, 255, 0)  # Green
elif random_colour == "Blue":
    pixels[0] = (0, 0, 255)  # Blue
elif random_colour == "Yellow":
    pixels[0] = (255, 255, 0)  # Yellow
elif random_colour == "Purple":
    pixels[0] = (128, 0, 128)  # Purple

for countdown in range(5, -1, -1):
    oled.fill_rect(70, 50, 10, 7, 0)
    oled.text(f"{countdown}", 70, 50, 1)
    oled.show()
    time.sleep(1)
```



CHAPTER 5

Automated Waste Bin

Servo Motor & Proximity Sensor

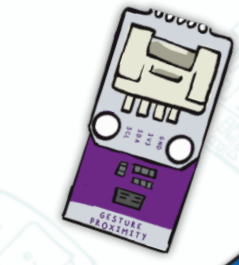
Introduction to Servo Motor

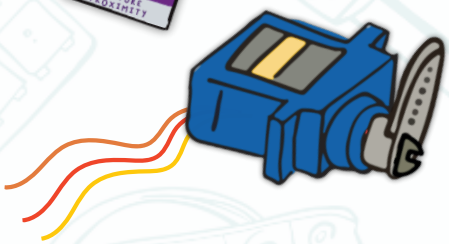
Introduction to Proximity Sensor

Project: Automated Waste Bin

In this chapter, we will learn to control the servo motor and read proximity values using the proximity sensor module. The servo motor is a crucial component in robotics, enabling precise positioning for applications such as solar panel alignment and automated doors.

The proximity sensor comes with various built-in functionalities, one of which is measuring proximity to detect nearby objects. It empowers machines to perceive their surroundings, much like how our senses work for us. With its ability to detect the presence of objects, it can enable innovative solutions such as touchless switches and gesture-controlled devices.





Introduction to Servo Motor

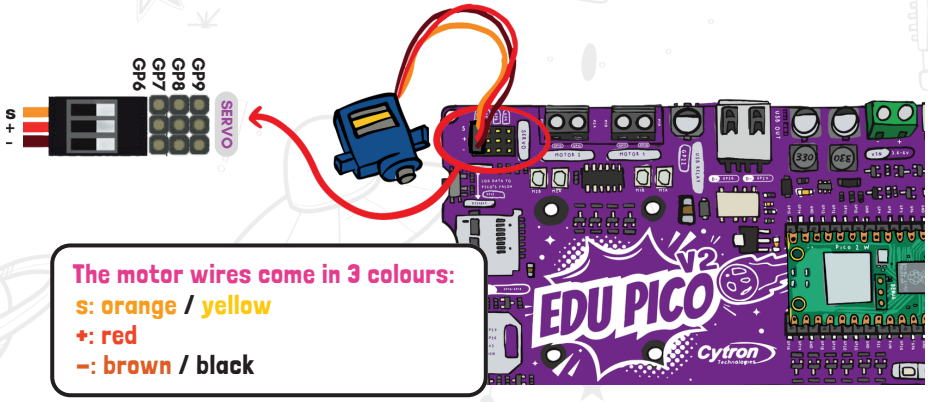
Servo motors are essential components in the world of robotics and automation. They allow precise control over the angular position of a shaft, making them ideal for applications like controlling robotic arms, steering mechanisms, or even opening and closing doors. In this activity, you will learn how to connect and control the servo motor to specific angles with the EDU PICO.

How Does This Activity Work?

- **Libraries:** board, time, pwmio, adafruit_motor.
- **Servo Motor Configuration:**
  - Attach and screw the horn to the servo motor.

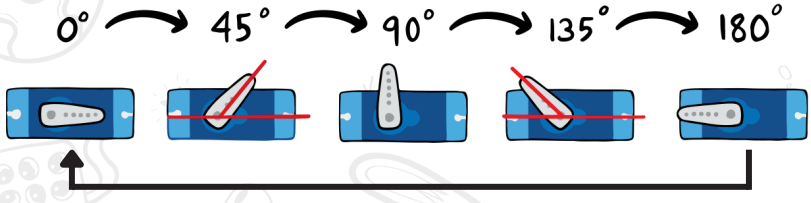


  - Connect the orange wire to S (**GP6**), red to (+) and brown to (-), as shown in the illustration below.

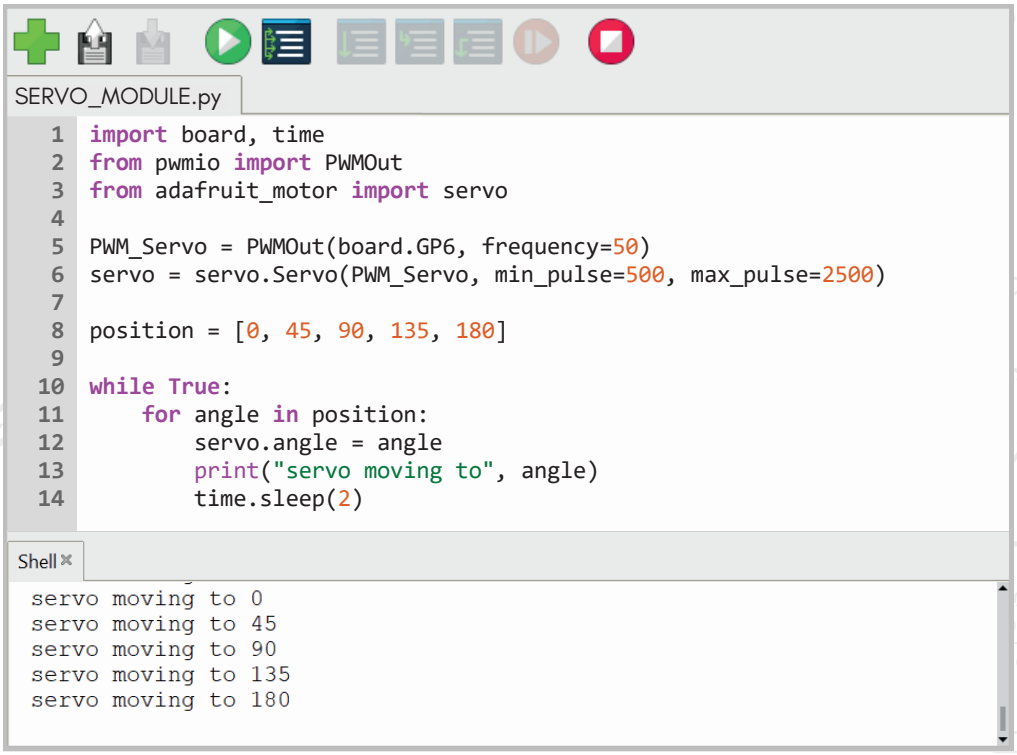


The motor wires come in 3 colours:
s: orange / yellow
+: red
–: brown / black

- **Output:** The servo rotates to each angle position from 0 to 45, 90, 135, and 180 degrees repetitively.



Code



SERVO_MODULE.py

```
1  import board, time
2  from pwmio import PWMOut
3  from adafruit_motor import servo
4
5  PWM_Servo = PWMOut(board.GP6, frequency=50)
6  servo = servo.Servo(PWM_Servo, min_pulse=500, max_pulse=2500)
7
8  position = [0, 45, 90, 135, 180]
9
10 while True:
11     for angle in position:
12         servo.angle = angle
13         print("servo moving to", angle)
14         time.sleep(2)
```

Shell

```
servo moving to 0
servo moving to 45
servo moving to 90
servo moving to 135
servo moving to 180
```

Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Import Necessary Libraries

Libraries

```
2 from pwmio import PWMOut
3 from adafruit_motor import servo
```

Line 2: **PWMOut** is a class from the **pwmio** library that allows you to create a PWM (Pulse Width Modulation) output on a specific pin.

Line 3: **adafruit_motor** module allows a high-level interface (allow programmer to write shorter code for controlling a servo motor.

Initialize Hardware Components

Configure Servo Pins & Rotation Sequence

```
5 PWM_Servo = PWMOut(board.GP6, frequency=50)
6 servo = servo.Servo(PWM_Servo, min_pulse=500, max_pulse=2500)
7
8 position = [0, 45, 90, 135, 180]
```

Line 5: Initializes ***GP6*** pin as output with the PWM frequency parameter set to **50 Hz**.

Line 6: Specify **PWM_Servo** as the PWM output for controlling the servo motor. The **min_pulse** is set to **500**, which represents the minimum pulse width for the servo motor's rotation, and **max_pulse** is set to **2500**, which represents the maximum pulse width for the servo motor's rotation. These values define the range of motion for the servo.

Line 8: Create a list with a variable named **position** that contains a sequence of angles [0, 45, 90, 135, 180]. These angles represent the positions to which the servo motor will be moved.

Enter a Continuous Loop

Main Loop

```
10 while True:
11     for angle in position:
12         servo.angle = angle
13         print("servo moving to", angle)
14         time.sleep(2)
```

Line 11: Initiates a for loop that iterates through each value in the **position** list: [0, 45, 90, 135, 180].

Line 12: Sets the angle of the servo motor to the value stored in the **angle** variable in every iteration; this command also controls the servo motor's position by specifying the angle it should move.

Line 13: Prints the angle the servo motor is currently located to the shell console.

Line 14: Remain at the same angle for **2** seconds before iterating to the next loop.

Activity with AI

Prompt Framework: Servo Motor

Task: Use Button A to trigger the servo motor to move from 0° to 180°.

Step 1: Teach the AI

Copy and paste your working code from **Page 89** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Button A connected to GP0.

Here is my current code:
<PASTE YOUR CODE FROM PAGE 89>

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

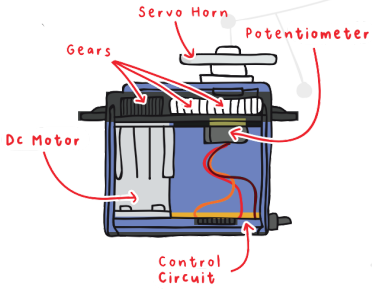
Prompt #2:
"Make the servo move from 0° to 180° only when Button A is pressed. When Button A is not pressed, return the servo to 0°."

Step 3: Check the Answer

Copy and paste the code in Thonny, then run the code. Check if by pressing Button A the servo rotates from 0° to 180°, and when Button A is released, the servo returns to 0°.

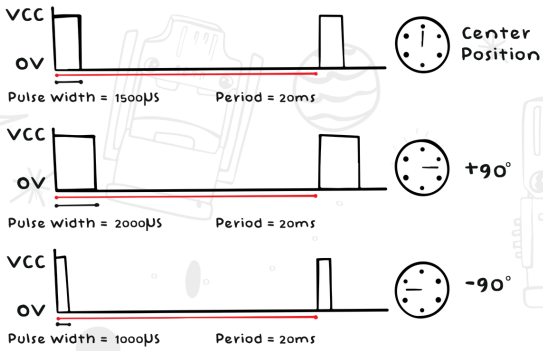
THE MORE YOU KNOW

A servo motor typically consists of a DC motor, gears, a potentiometer (position sensor), and a control circuit. The built-in controller translates commands in the form of pulses to rotate the servo motor in degrees. Unlike a DC motor that rotates continuously (which will be covered in Chapter 8), we can control a servo motor rotation to an assigned angle between the range of 0 to 180 degrees.



Pulse Width Modulation (PWM)

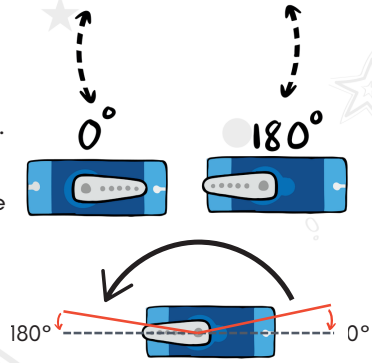
Modern RC servos rely on pulse-width modulation to determine the angle of mechanical rotation. Although a standard RC servo expects a pulse every 20ms (milliseconds), the duration of this pulse can vary significantly across different servos. In EDU PICO's case, we are using a pulse of 20ms where a PWM frequency of 50Hz is set (1/50 = 20ms).



Servos have traditionally been limited to a pulse width range of 1000 - 2000us, offering a **90°** range of motion. However, modern servos have a much wider range of motion, typically **170°** - **180°**, which requires pulse widths outside of the standard range.

```
6 servo = servo.Servo(PWM_Servo, min_pulse=_____, max_pulse=_____)
```

If your servo doesn't rotate from **0°** to **180°** correctly, you may need to adjust the **min_pulse** and **max_pulse** values to calibrate the range of movement. If you hear a buzzing noise coming from the servo, the value(s) you set may be too low or too high, causing the servo mechanism to hit the end stop. Re-adjust the values carefully to find a safe range of movement.

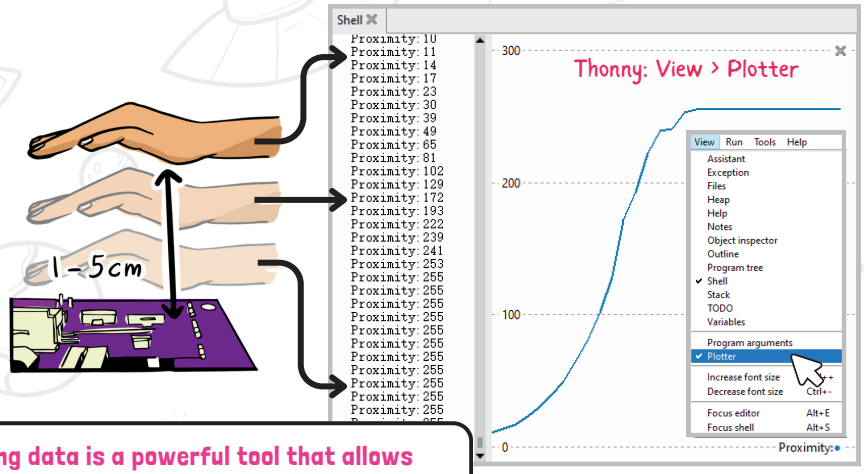


Introduction to Proximity Sensor

The proximity sensor detects the presence or absence of objects using the reflection of infrared light (IR) without physical contact. They are used in various applications, such as touchless switches in public restrooms, automatic faucets, and smartphones. Proximity sensors are also important in industrial automation, robotics, and automotive systems for tasks like object detection and machine safety features.

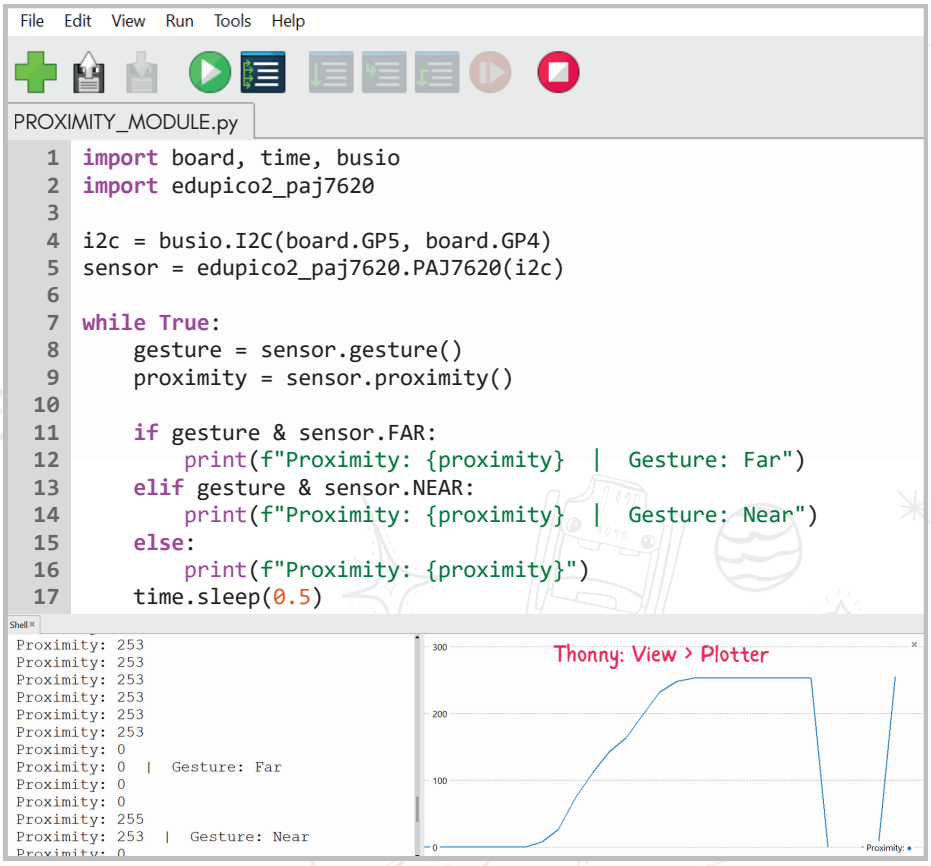
How Does This Activity Work?

- **Libraries:** board, time, busio, edupico2_PAJ7620.
- **Proximity Sensor I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Input:** Move your hand up and down above the proximity sensor.
- **Output:**
  - The proximity sensor will continuously read and print the proximity values to the shell console.
  - The closer the obstacle to the sensor, the greater the proximity value.
  - The code will continuously update and display the proximity value at an interval of **0.5** second.
- **Action Required:** Enable the plotter function in Thonny IDE to have a better visual on the proximity value.



Graphing data is a powerful tool that allows visualization on complex information. It also makes it easier to identify patterns and trends.

Code



Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

PAJ7620 Initialization

```
4 i2c = busio.I2C(board.GP5, board.GP4)
5 sensor = edupico2_paj7620.PAJ7620(i2c)
```

Line 4 - 5: Initializes the I2C communication bus for the PAJ7620 sensor. The configuration is similar to when using the gesture sensor.

Main Loop

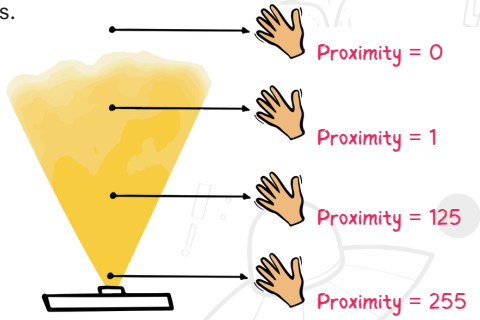
```
7  while True:
8      gesture = sensor.gesture()
9      proximity = sensor.proximity()
10
11     if gesture & sensor.FAR:
12         print(f"Proximity: {proximity}  |  Gesture: Far")
13     elif gesture & sensor.NEAR:
14         print(f"Proximity: {proximity}  |  Gesture: Near")
15     else:
16         print(f"Proximity: {proximity}")
17     time.sleep(0.5)
```

Line 8: Detects whether the hand motion is moving towards the sensor or away from the sensor, in this case "Near" or "Far".

Line 9: Proximity gives a numerical value of how close something is to the sensor. The nearer the object to the sensor, the higher the value it gets (255).

Line 11 - 14: gesture & sensor.FAR means the sensor detected a "Far" gesture whereas **gesture & sensor.NEAR** detects a "Near" gesture.

Line 16: If neither motion is detected, the sensor continues to print only the proximity values.



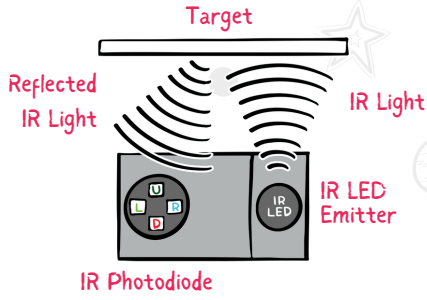
When you place your smartphone next to your ear and the display turn off automatically, that's actually triggered by using this similar sensor.

THE MORE YOU KNOW

Proximity Sensor

IR LED (Emitter): Active proximity sensor consist of an IR LED (Infrared Light Emitting Diode) that emits infrared light (around 950 nanometers) when powered on.

IR Photodiode (Receiver): The IR photodetector (usually known as a photodiode) is placed next to the IR LED. It is sensitive to the same infrared light that the IR LED emits. The intensity of the reflected infrared light is used to determine the distance of the object!



Activity with AI

Prompt Framework: Proximity Sensor

Task: Use the proximity sensor to trigger a buzzer. The closer an object gets, the faster the beeping (like a car reverse sensor).

Step 1: Teach the AI

Copy and paste your working code from **Page 94** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Buzzer connected to GP21

Here is my current code:
<PASTE YOUR CODE FROM PAGE 94>

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

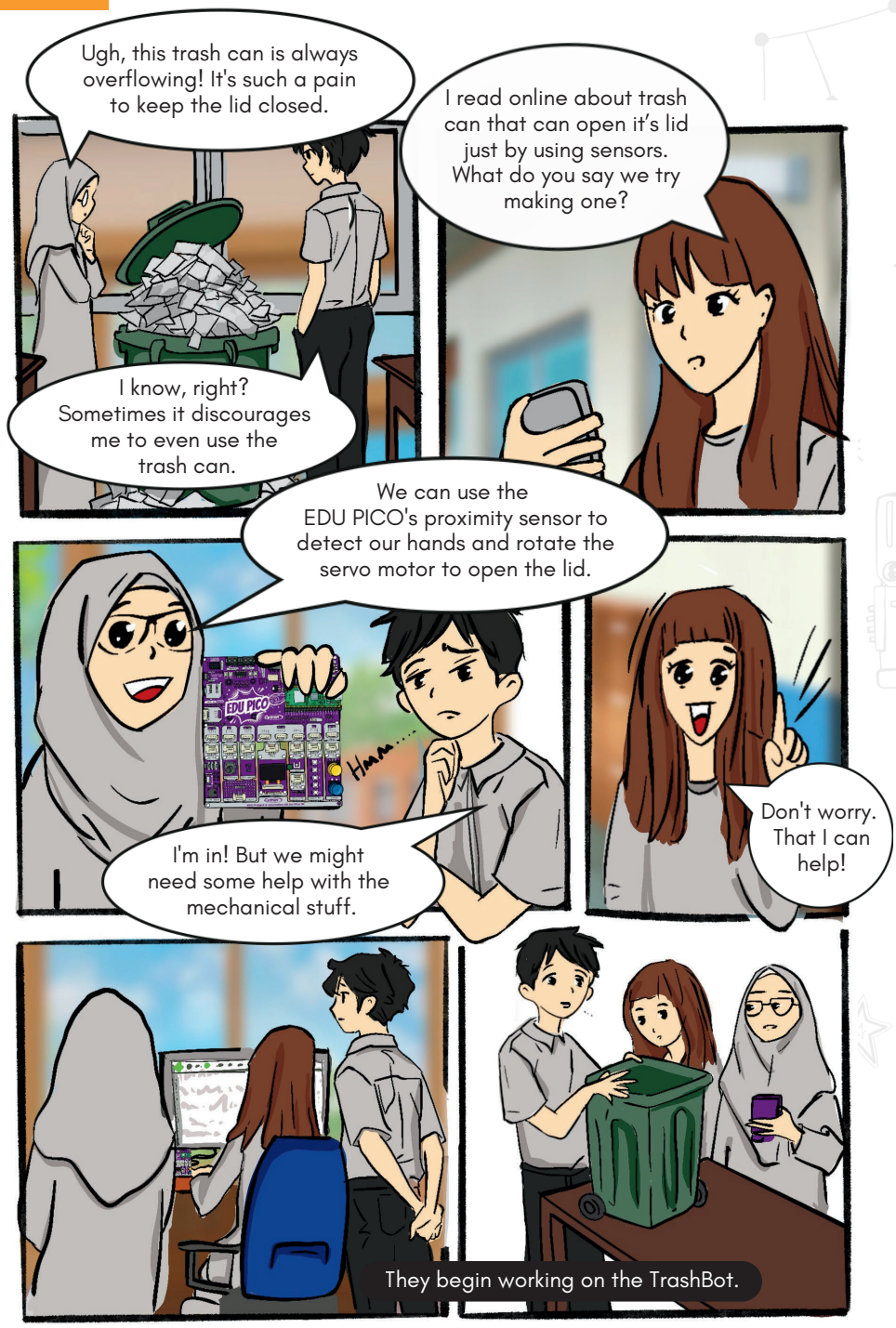
Prompt #2:
"Use the proximity sensor readings to control a buzzer. When an object gets closer, the buzzer beeps faster. When the object is far away, the buzzer beep slowly or stay silent."

Step 3: Check the Answer

Copy and paste the code in Thonny, then run the code. Check if the buzzer beeps faster as your hand moves closer to the sensor.

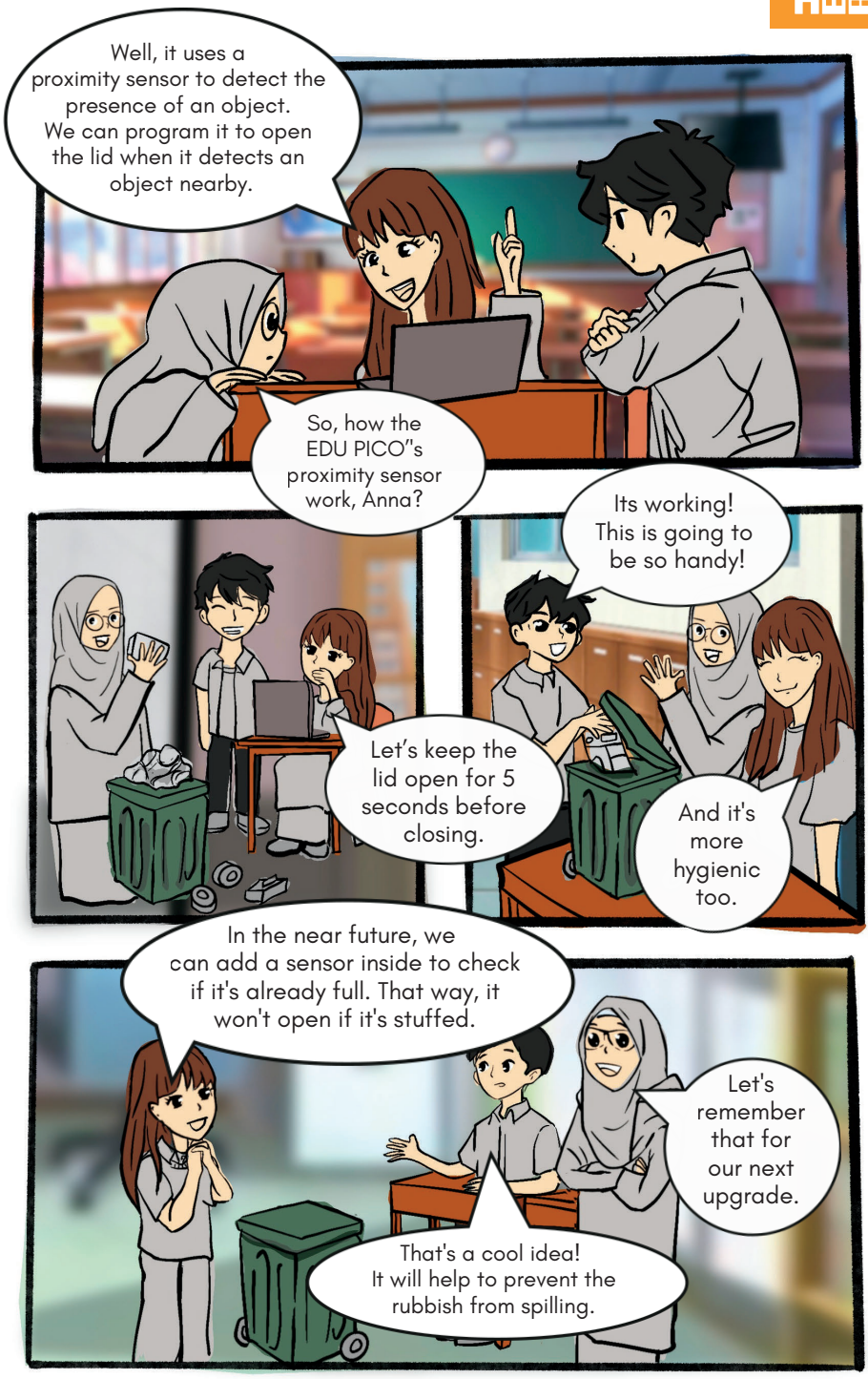
11 SUSTAINABLE CITIES AND COMMUNITIES

PROLOGUE: AUTOMATED WASTE BIN



PROLOGUE: AUTOMATED WASTE BIN

11 SUSTAINABLE CITIES AND COMMUNITIES



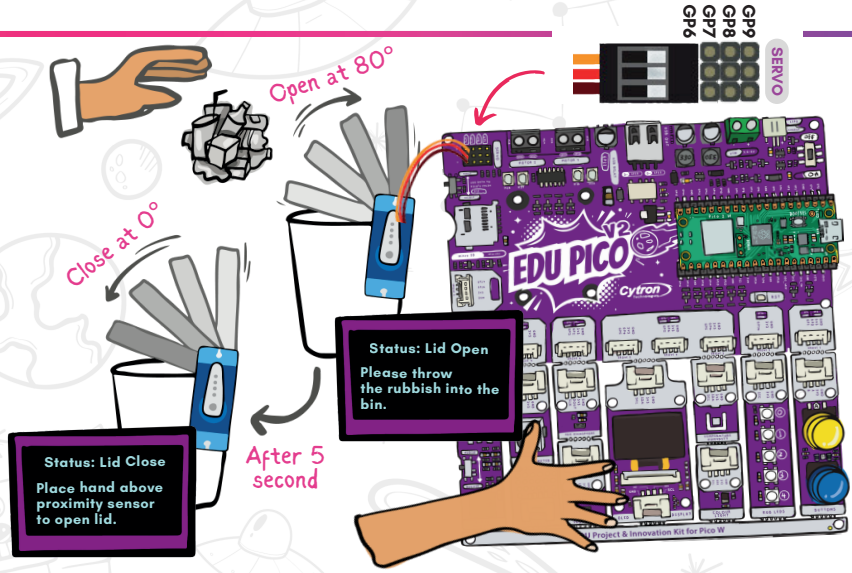
Automated Waste Bin

TrashBot Smart Bin

Time to build your very own TrashBot! Equipped with an proximity sensor and a servo motor, it detects the presence of your hand (or any object) and automatically opens its lid, eliminating the need for physical contact. This makes using it more convenient and hygienic, reducing the risk of germs spreading.

How Does This Activity Work?

- **Libraries:** board, time, busio, adafruit_ssd1306, adafruit_motor, pwmio, edupico2_paj7620, font5x8.bin.
- **Proximity Sensor and OLED Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Servo Motor Configuration:** ***GP6*** with orange wire connected to (S), red to **(+),** and brown to **(-)**, as shown in the illustration below.
- **Input:** Place your hand (or any object) roughly 1 cm above the proximity sensor to activate the servo motor.
- **Output:**
  - If an object is detected by the proximity sensor, the servo motor will rotate from **0** to **80** degrees, opening the lid of the trashbin.
  - The OLED will print the status of the lid from "Status: Lid Close" to "Status: Lid Open" once the sensor is triggered.
  - The lid remains open for **5** second before closing.



Code



PROJECT_AUTOMATED_WASTE_BIN.py

```
import board, time, busio, adafruit_ssd1306
import edupico2_paj7620
from adafruit_motor import servo
from pwmio import PWMOut

i2c = busio.I2C(board.GP5, board.GP4)
sensor = edupico2_paj7620.PAJ7620(i2c)

oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)

PWM_Servo = PWMOut(board.GP6, frequency=50)
servo = servo.Servo(PWM_Servo, min_pulse=500, max_pulse=2500)
servo.angle = 0

def show_lid_close():
    oled.fill(0)
    oled.text("Status: Lid Close", 10, 10, 1)
    oled.text("Place hand above", 10, 30, 1)
    oled.text("proximity sensor", 10, 40, 1)
    oled.text("to open lid.", 10, 50, 1)
    oled.show()

def show_lid_open():
    oled.fill(0)
    oled.text("Status: Lid Open", 10, 10, 1)
    oled.text("Please throw", 10, 30, 1)
    oled.text("the rubbish into", 10, 40, 1)
    oled.text("the bin.", 10, 50, 1)
    oled.show()

while True:
    proximity = sensor.proximity()

    if proximity > 20:
        servo.angle = 80
        show_lid_open()
        time.sleep(5)
        servo.angle = 0
        show_lid_close()

    time.sleep(0.05)
```

What the Code Does

Import Necessary Libraries

Libraries

```
1 import board, time, busio, adafruit_ssd1306
2 import edupico2_paj7620
3 from adafruit_motor import servo
4 from pwmio import PWMOut
```

Line 1: **adafruit_ssd1306** is used for controlling the OLED display.

Line 2: **edupico2_paj7620** is used for controlling the PAJ7620 proximity sensor.

Line 3 - 4: **adafruit_motor** and **pwmio** are used for controlling of the servo motor.

Initialize Hardware Components

Hardware Initialization

```
6  i2c = busio.I2C(board.GP5, board.GP4)
7  sensor = edupico2_paj7620.PAJ7620(i2c)
8
9  oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
10
11 PWM_Servo = PWMOut(board.GP6, frequency=50)
12 servo = servo.Servo(PWM_Servo, min_pulse=500, max_pulse=2500)
13 servo.angle = 0
```

Line 6: Initialize I2C communication on GPIO pins ***GP5*** and ***GP4***.

Line 7: Create an instance of the PAJ7620 proximity sensor and enable proximity sensing.

Line 9: Initialize SSD1306 OLED display with a resolution of **128**x**64** pixels.

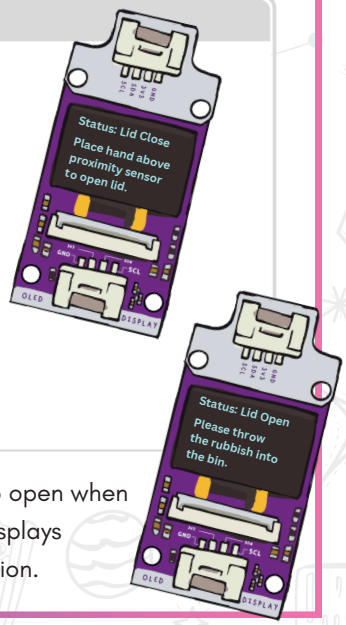
Line 11: Create a PWMOut instance for controlling a servo on GPIO pin ***GP6*** with a PWM frequency of **50 Hz**.

Line 12: Create an instance (**servo**) using the **servo.Servo** class, specifying the **PWMOut** instance and pulse width limits.

Line 13: Initialize the servo motor by rotating it to **0** degrees.

Define a Custom Function

```
15 def show_lid_close():
16     oled.fill(0)
17     oled.text("Status: Lid Close", 10, 10, 1)
18     oled.text("Place hand above", 10, 30, 1)
19     oled.text("proximity sensor", 10, 40, 1)
20     oled.text("to open lid.", 10, 50, 1)
21     oled.show()
22
23 def show_lid_open():
24     oled.fill(0)
25     oled.text("Status: Lid Open", 10, 10, 1)
26     oled.text("Please throw", 10, 30, 1)
27     oled.text("the rubbish into", 10, 40, 1)
28     oled.text("the bin.", 10, 50, 1)
29     oled.show()
```



show_lid_close() function is called to display instruction to open when the lid is in the closed position. **show_lid_open** function displays instruction to throw rubbish when the lid is in the open position.

Enter a Continuous Loop

Main Loop

```
31 while True:
32     proximity = sensor.proximity()
33
34     if proximity > 20:
35         servo.angle = 80
36         show_lid_open()
37         time.sleep(5)
38         servo.angle = 0
39         show_lid_close()
40
41     time.sleep(0.05)
```

Line 32: Read proximity data from the PAJ7620 sensor using **sensor.proximity** and store it in the **proximity** variable.

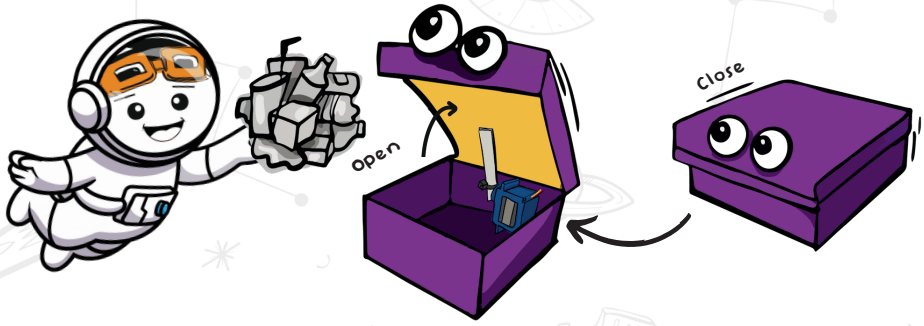
Line 34: A proximity value greater than **20** indicates that an object or hand is close to the proximity sensor.

Line 35 - 39: When the proximity sensor detects an object, the servo motor will rotate and remain at **80** degrees for **5** seconds. After **5** seconds, the servo motor returns to its original state at **0** degree, closing the lid of the trash bin.

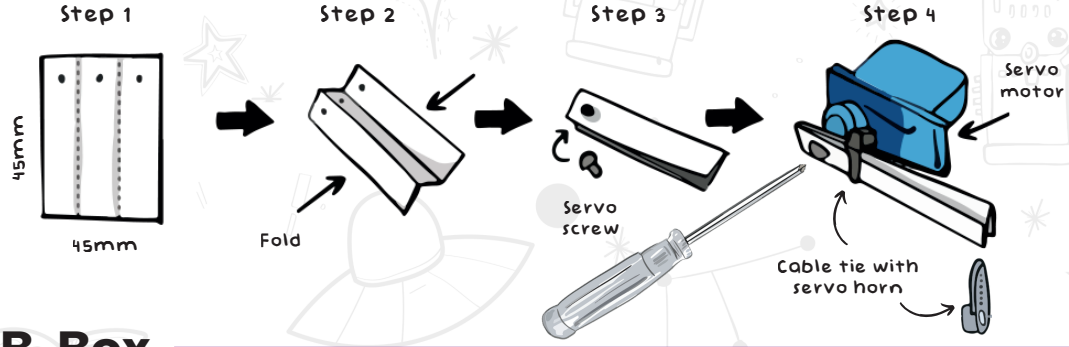
WHAT'S NEXT?

TRASHBOT ACCESSORY

Let's build our very own Trashbot! You will need these materials to accomplish this project. Worry not, all the materials are already included in the EDU PICO 2 set.



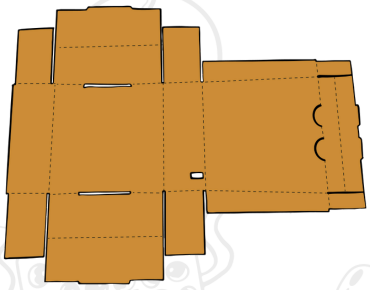
A. Servo Stick



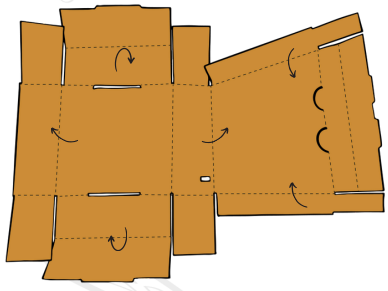
B. Box

Fold the Trashbot box accessory as shown below.

Step 1

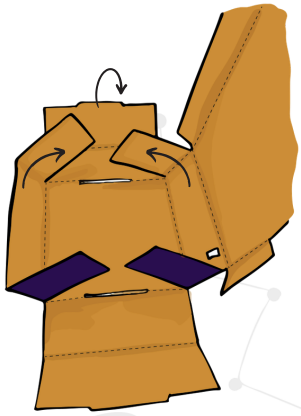


Step 2

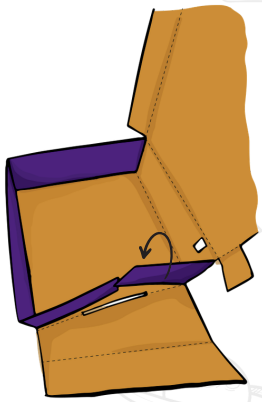


Fold the Trashbot's body.

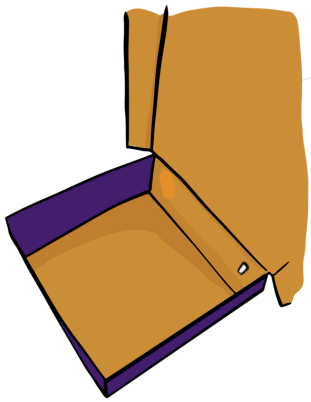
Step 3:



Step 4:

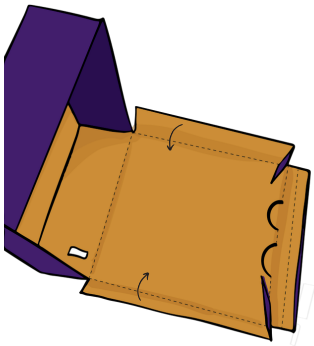


Step 5:

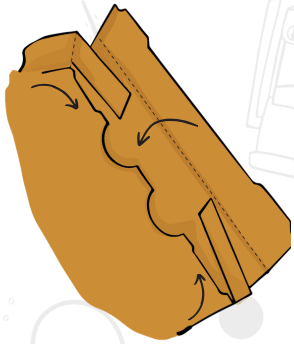


Fold the Trashbot's head.

Step 6:



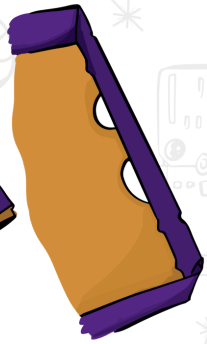
Step 7:



Step 8:

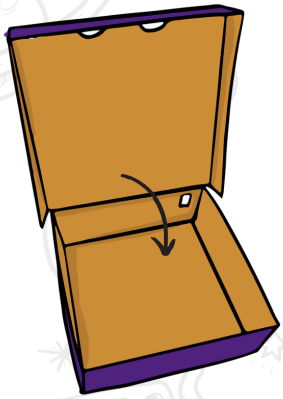


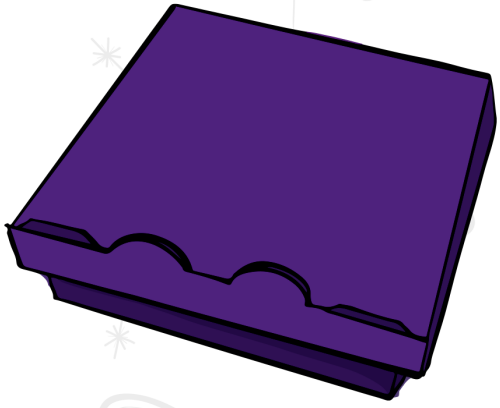
Step 9:



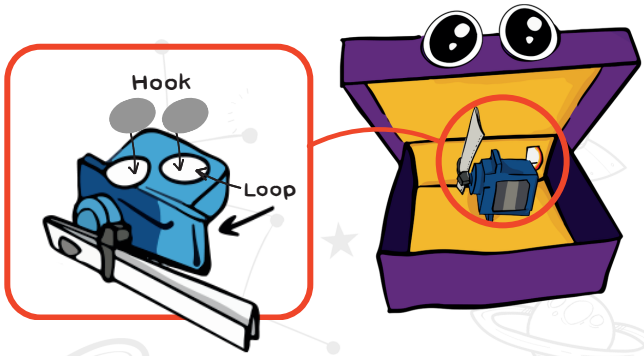
Step 10:

Fold the top of the Trashbot downwards, and then it's complete!

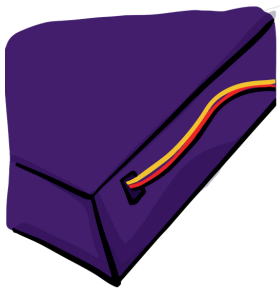




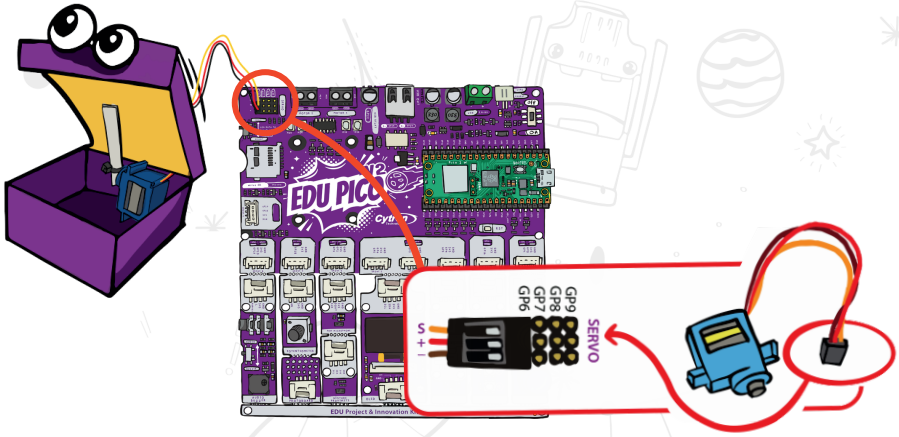
Step 11:
Attach the velcro's hook above the loop and attach the servo motor to the inner back of the Trashbot.



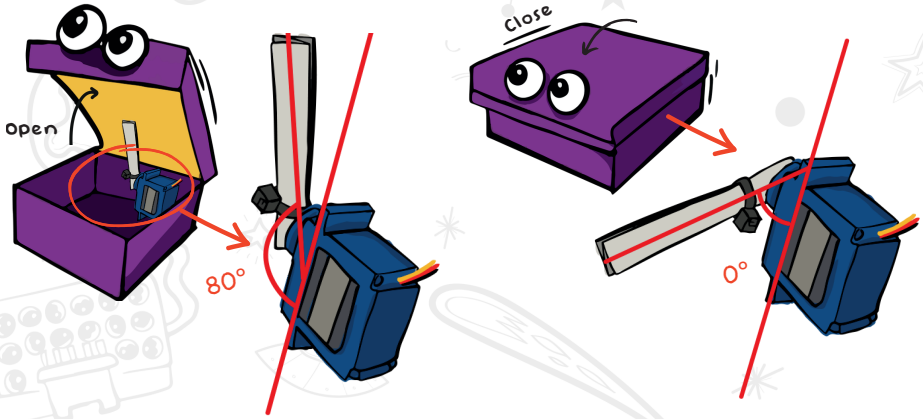
Step 12:
Slide the cable through the back of the box.



Step 13:
Connect Servo Motor Cable to EDU PICO board at GP6.



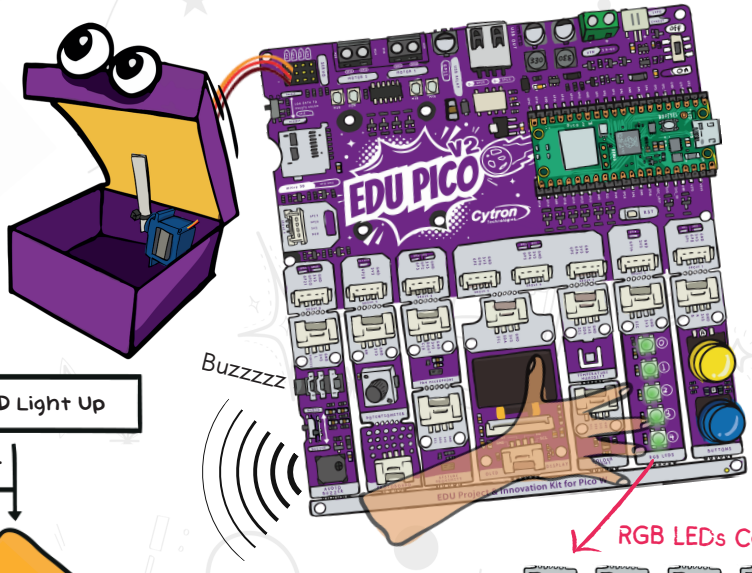
Step 14:
Test the Trashbot's servo motor's angle.

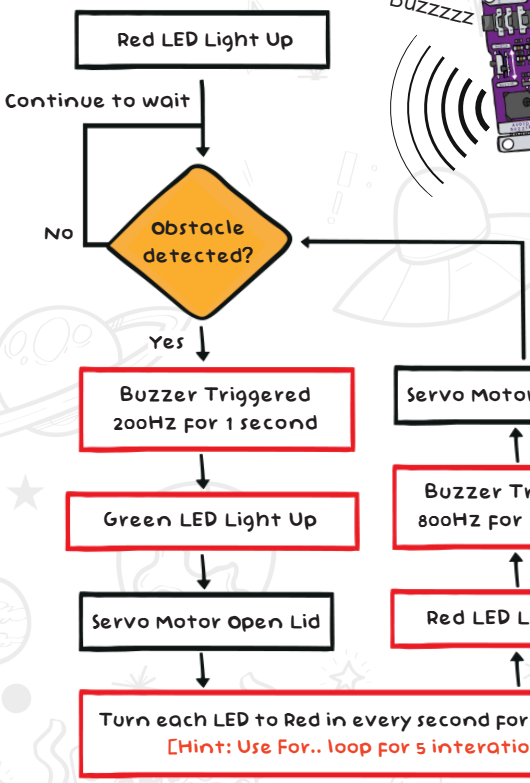


CHALLENGE

Now that we've got the Trashbot running, why don't we introduce two more elements into the project? The first element, let's light up the RGB LEDs in green when an obstacle is detected and remain red when there are no obstacles. After that, the RGB LEDs starts a countdown to indicate how much time is left before the lid of the bin closes.

The next element we will introduce is sound; let's add a soothing buzz tone when the lid returns to its closed state. The tone will indicate to the user that the Trashbot's work is done. Ready? Let's give it a try!







You will need to include the codes highlighted in red boxes.

CHAPTER 6

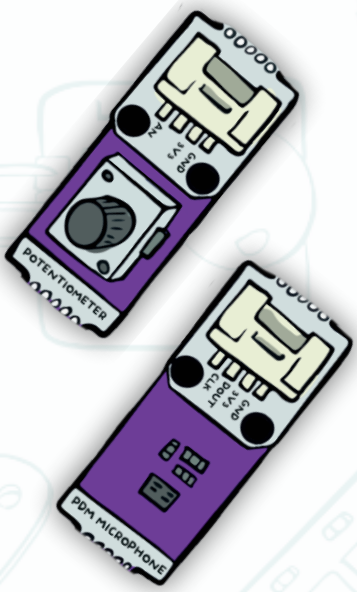
Noise Pollution Monitoring System

Potentiometer & PDM Sound Sensor

Introduction to Potentiometer

Introduction to PDM Sound Sensor

Project: Room Noise Monitoring System



Hello Makers!

In this chapter, we will learn to measure and analyze analog and digital signals using the Potentiometer and the pulse density modulation (PDM) Sound Sensor.

Imagine being able to control the brightness of an LED or the speed of a motor with a simple twist on the potentiometer, or to trigger light switches or fan with a clap.

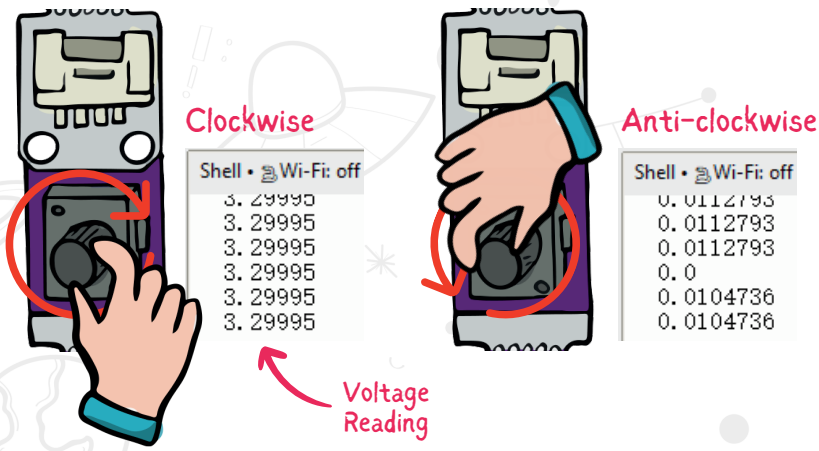
That's exactly what you can achieve using the components covered in this chapter. Let's go!

Introduction to Potentiometer

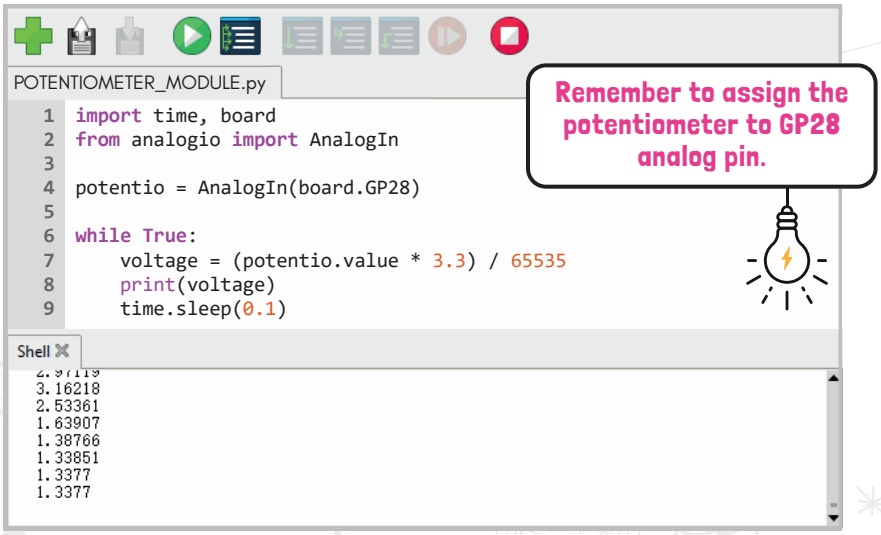
In this lesson, you will learn to interpret analog signal voltage while manipulating a variable resistor or a potentiometer (pot). This is also one of the more popular projects created for beginners to learn how to control electrical output by using a simple analog input device.

How Does This Activity Work?

- **Libraries:** board, time, AnalogIn.
- **Potentiometer Configuration:** ***GP28*** with analog input.
- **Input:**
  - Turning the potentiometer knob will alter the output flow of electricity.
  - Turning the knob clockwise will increase the voltage, and anticlockwise will reduce the voltage.
- **Output:**
  - The code continuously read the voltage from the analog pin ***GP28*** (connected to the EDU PICO potentiometer) and prints the voltage value to the shell console at an interval of **0.1** second.



Code



Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Import Necessary Libraries

```
Libraries
1 import time, board
2 from analogio import AnalogIn
```

Line 2: **analogio** library enables interaction with analog pins, typically used for reading analog signals through components like the potentiometer.

Initialize Hardware Components

```
4 potentio = AnalogIn(board.GP28)
```

Line 4: Create an AnalogIn instance named **potentio** and assign it to pin ***GP28***. This prepares the pin to read analog voltage.

Enter a Continuous Loop

```
6 while True:
7     voltage = (potentio.value * 3.3) / 65535
8     print(voltage)
9     time.sleep(0.1)
```

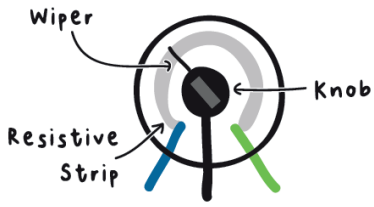
Line 7: Read the analog voltage from the potentiometer using potentio.value. This value is an integer ranging from **0** to **65535 (2^16 - 1)**, which represents the voltage level across the potentiometer. The raw analog value is then converted to voltage by scaling it from the range [0, 65535] to [0, 3.3] volts.

Line 8: Print the calculated voltage to the shell console.

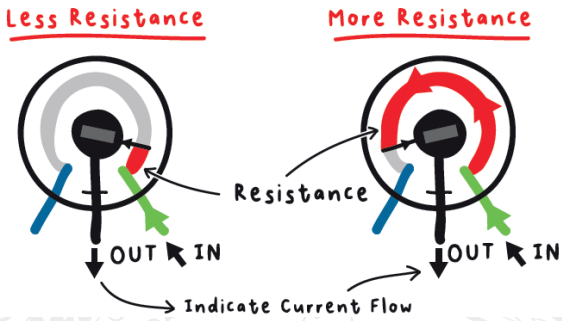
Line 9: Add a small delay of **0.1** second using **time.sleep(0.1)** to limit the rate of voltage readings.

THE MORE YOU KNOW

Potentiometer



A potentiometer, often referred to as a "pot," is a type of variable resistor used to control electrical signals. It consists of a resistive element with a movable contact called a wiper. If you have a 10kΩ potentiometer, turning the knob allows you to adjust the resistance value between 0Ω to 10,000Ω.



Potentiometers are commonly used in audio devices for volume & balance control too!

Activity with AI

Prompt Framework: Potentiometer

Task: Use the potentiometer to control the brightness of the RGB LED strip.

Step 1: Teach the AI

Copy and paste your working code from **Page 109** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

RGB LED is connected to GP14 with 5 LEDs

Here is my current code:
<PASTE YOUR CODE FROM PAGE 109>

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

Prompt #2:
"Use the potentiometer to control the brightness of all RGB LEDs together. The LED will light up in pink by default."

Step 3: Check the Answer

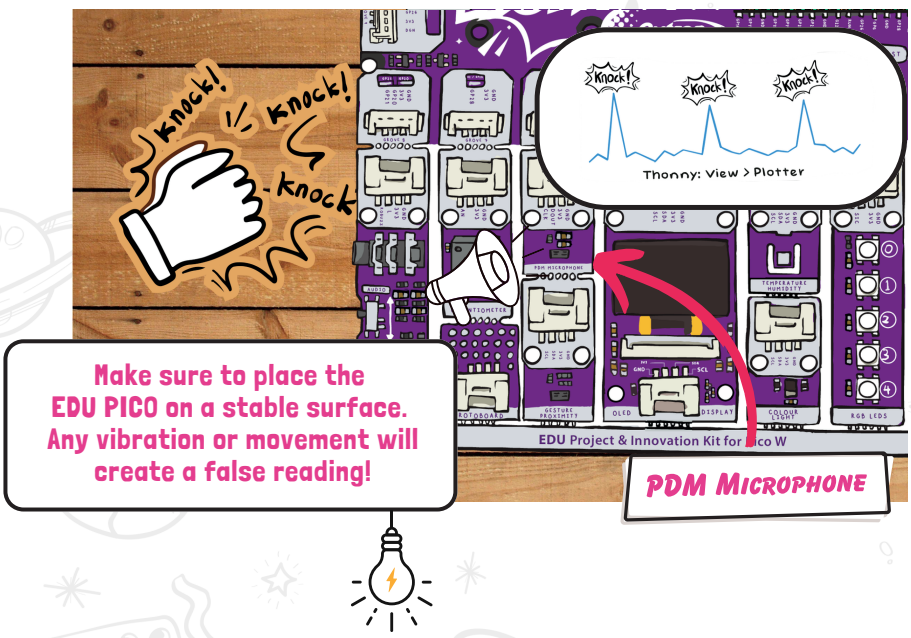
Copy and paste the code in Thonny, then run the code. Check if turning the potentiometer clockwise should increase the LED brightness and anti-clockwise to reduce the LED brightness. All the LEDs should like up pink in colour.

Introduction to Sound Sensor

A PDM microphone, or Pulse-Density Modulation microphone, is a type of microphone that converts sound waves into a digital signal. Unlike our traditional microphones that use analog signals, the digital signals produced by PDM microphones are more resistant to noisy environments, making them ideal for applications such as smart home voice commands, noise level monitoring, or even sound analysis.

How Does This Activity Work?

- **Libraries:** board, time, array, math, audiobusio.
- **PDM microphone configuration:** Data (DAT) = ***GP2***, Clock (CLK) = ***GP3***.
- **Output:**
  - The sensor will measure sound level in voltage (magnitude) and convert it to sound level in decibels (dB).
  - If the magnitude is greater than **0**, the code prints the sound level in decibels (dB) at the shell console with an interval of **0.1** second.
  - If the magnitude is less than **0**, the code prints "Magnitude is too small to calculate dB." at the shell console.



Code

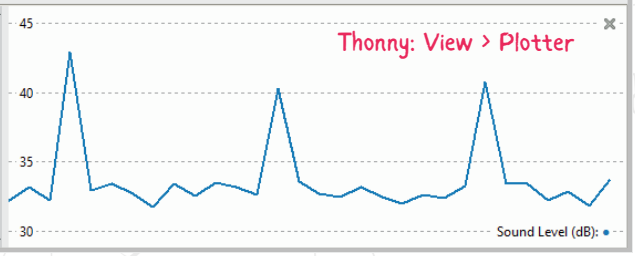
File Edit View Run Tools Help

PDM_SOUND_MODULE.py

```
1  import board, time, array, math, audiobusio
2
3  mic = audiobusio.PDMIn(board.GP3, board.GP2, sample_rate=32000, bit_depth=16)
4  samples = array.array('H', [0] * 1024)
5
6  def log10(x):
7      return math.log(x) / math.log(10)
8
9  def normalized_rms(values):
10     minbuf = sum(values) / len(values)
11     samples_sum = sum(float(sample - minbuf) * (sample - minbuf)for sample in values)
12     return math.sqrt(samples_sum / len(values))
13
14 while True:
15     mic.record(samples, len(samples))
16     magnitude = normalized_rms(samples)
17     if magnitude > 0:
18         sound_level_dB = 20 * log10(magnitude)
19         print(f"Sound Level (dB): {sound_level_dB:.2f}")
20     else:
21         print("Magnitude is too small to calculate dB.")
22     time.sleep(0.1)
```

Shell

```
Sound Level (dB): 32.75
Sound Level (dB): 32.48
Sound Level (dB): 33.21
Sound Level (dB): 32.51
Sound Level (dB): 32.03
Sound Level (dB): 32.66
Sound Level (dB): 32.41
Sound Level (dB): 33.27
Sound Level (dB): 40.76
Sound Level (dB): 33.44
Sound Level (dB): 33.45
Sound Level (dB): 32.23
Sound Level (dB): 32.88
Sound Level (dB): 31.83
Sound Level (dB): 33.78
```



Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Import Necessary Libraries

Libraries

```
1 import board, time, array, math, audiobusio
```

math: Provides mathematical functions.

audiobusio: Provides audio input and output functionality.

Initialize Hardware Components

Initialize Microphone

```
3 mic = audiobusio.PDMIn(board.GP3, board.GP2, sample_rate=32000, bit_depth=16)
4 samples = array.array('H', [0] * 1024)
```

Line 3: Configures the microphone to use GPIO pins ***GP3*** and ***GP2*** for audio input and sets the sample rate to **32,000** samples per second with bit depth set to **16** bits.

Line 4: An array named **samples** is created to collect and store **1,024** audio samples per cycle.

Defining Custom Functions

Functions

```
6  def log10(x):
7      return math.log(x) / math.log(10)
8
9  def normalized_rms(values):
10     minbuf = sum(values) / len(values)
11     samples_sum = sum(float(sample - minbuf) * (sample - minbuf)for sample in values)
12     return math.sqrt(samples_sum / len(values))
```

Line 6 - 7: Defines a function **log10(x)** to calculate the base-10 logarithm of value **x**. This is used to convert the sound magnitude to decibels.

Line 9 - 12: This function calculates the normalized root mean square (RMS) of the audio samples. The variable **values** is an array of audio samples measured from the PDM microphone. It calculates the RMS by subtracting the mean value (**minbuf**) from each sample, squaring the result, and then taking the square root of the average of the squared values.

What is sample rate? Think of it as how often you take pictures. With a higher sample rate, you take more pictures per second, like a fast camera.



Enter a Continuous Loop

Main Loop

```
14 while True:
15     mic.record(samples, len(samples))
16     magnitude = normalized_rms(samples)
17     if magnitude > 0:
18         sound_level_dB = 20 * log10(magnitude)
19         print(f"Sound Level (dB): {sound_level_dB:.2f}")
20     else:
21         print("Magnitude is too small to calculate dB.")
22     time.sleep(0.1)
```

Line 15: Record audio samples from the microphone into the samples array using **mic.record**.

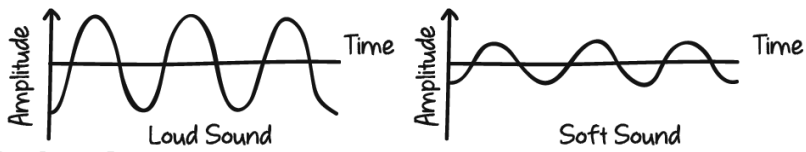
Line 16: Calculate the magnitude of the audio signal using the **normalized_rms** function.

Line 17 - 19: If the magnitude is greater than **0** (indicating that there is some sound), calculate the sound level in decibels (dB) using the **log10** function and print it to the shell console.

Line 20 - 21: Else if the magnitude is less than or equal to **0**, a message is printed indicating that the magnitude is too small to be calculated.

THE MORE YOU KNOW — Sound Sensor

Sound is generated through the vibrations of objects, such as when a drum is struck. These vibrations set the surrounding air molecules (the medium) into motion, resulting in the formation of sound waves.



Sound sensors are now integrated into almost all of our daily devices. For example, smartphones utilize voice recognition technology, enabling users to interact with virtual assistants simply by speaking commands or queries.

Activity with AI

Prompt Framework: Sound Sensor

Task: Use the sound sensor to trigger an LED On and Off when the sound level exceeds a set threshold (like clapping or loud noise).

Step 1: Teach the AI

Copy and paste your working code from **Page 113** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

RGB LED is connected to GP14 with 5 LEDs.

Here is my current code:
<PASTE YOUR CODE FROM PAGE 113>

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

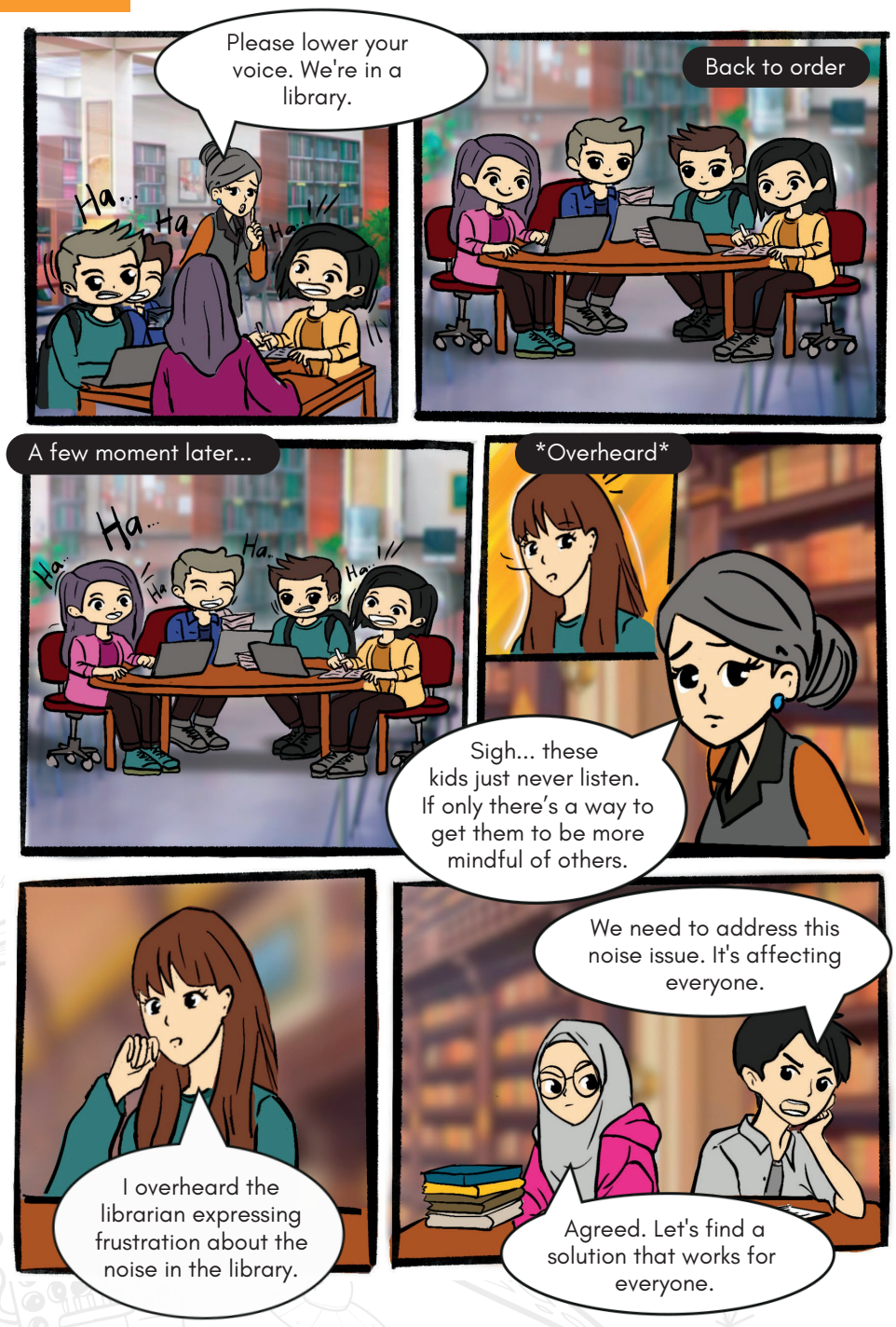
Prompt #2:
"Use the sound sensor to detect loud sounds. When the sound level exceeds 80dB, turn the LED ON. When the sound level is below 80dB, turn the LED OFF."

Step 3: Check the Answer

Copy and paste the code in Thonny, then run the code. Try clapping or make a loud sound near the microphone, the LED should turn ON. When quiet, the light should turn off.

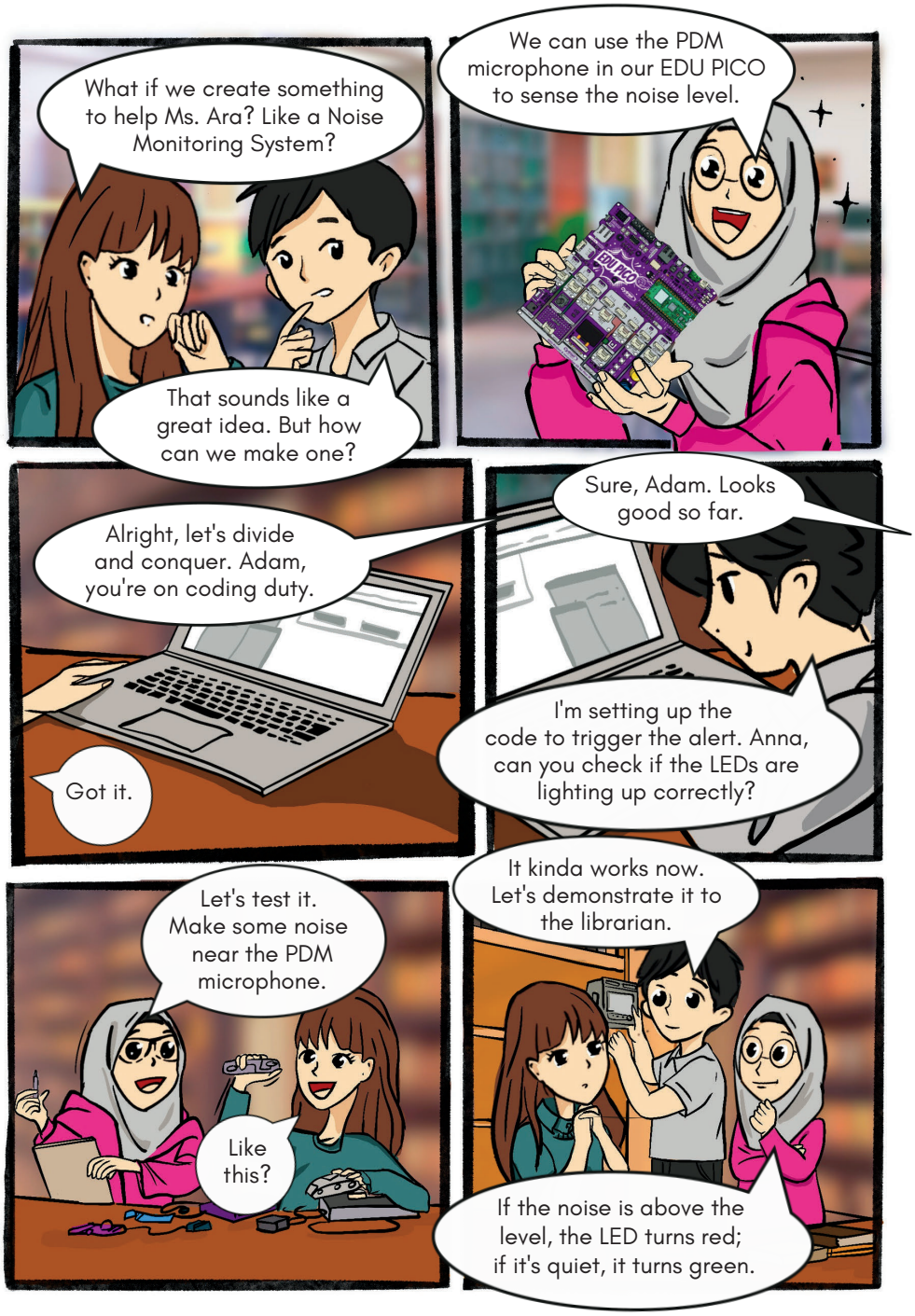
11 SUSTAINABLE CITIES AND COMMUNITIES

PROLOGUE: NOISE POLLUTION MONITORING



PROLOGUE: NOISE POLLUTION MONITORING

11 SUSTAINABLE CITIES AND COMMUNITIES



11 SUSTAINABLE CITIES AND COMMUNITIES

PROLOGUE: NOISE POLLUTION MONITORING

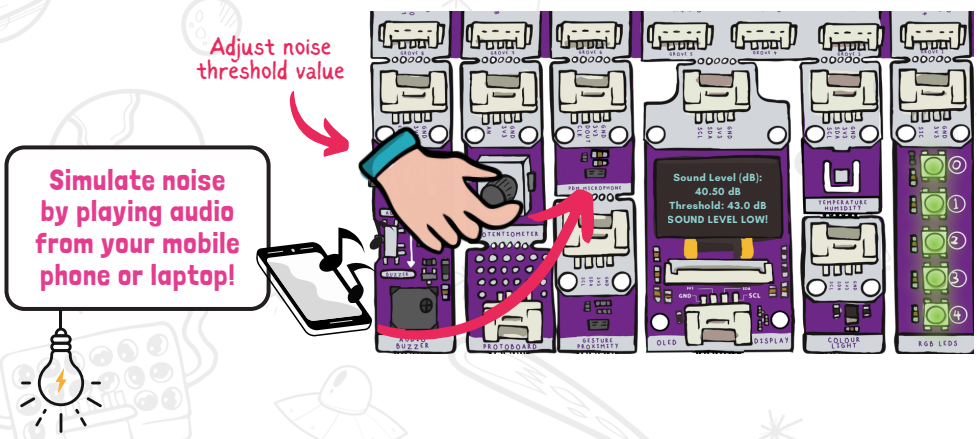


Noise Pollution Monitoring System

Noise pollution is a critical issue that often goes unnoticed in our ever-expanding urban society, ultimately affecting our overall well-being and quality of life. Fortunately, we can address this problem with the help of PDM microphone and EDU PICO. By learning how to use these tools, we can effectively measure and analyze the noise levels in our environment. Through this section, we will program a fully functional noise monitoring system that allows users to input an acceptable noise limit. In no time, you'll be able to deploy your EDU PICO for accurate noise measurement in your classroom or local library!

How Does This Activity Work?

- **Libraries:** board, time, analogIn, neopixel, busio, array, audiobusio, math, adafruit_ssd1306.
- **PDM Microphone Configuration:** ***GP2*** and ***GP3***.
- **OLED I2C Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Potentiometer Configuration:** ***GP28*** with analog input.
- **Input:**
  - Ambient noise (The louder the noise, the higher the decibel (dB)).
  - Adjust potentiometer value to adjust noise threshold (level).
- **Output:**
  - If noise received from the PDM microphone exceeds the threshold value set by the potentiometer, the RGB LEDs will light up in red, indicating the space is too noisy.
  - If noise is below the threshold, RGB LEDs will light up in green, indicating a safe noise level.



Code



PROJECT_ROOM_NOISE_MONITORING.py

```
import board, time, neopixel, busio, array, math, audiobusio, adafruit_ssd1306
from analogio import AnalogIn

i2c = busio.I2C(board.GP5, board.GP4)
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
potentiometer = AnalogIn(board.GP28)

pixels = neopixel.NeoPixel(board.GP14, 5, brightness=0.2)
pixels.fill(0)

mic = audiobusio.PDMIn(board.GP3, board.GP2, sample_rate=32000, bit_depth=16)
samples = array.array('H', [0] * 1024)

sound_min = 30
sound_max = 80

def log10(x):
    return math.log(x) / math.log(10)

def normalized_rms(values):
    minbuf = sum(values) / len(values)
    samples_sum = sum(float(sample - minbuf) * (sample - minbuf)for sample in values)
    return math.sqrt(samples_sum / len(values))

while True:
    oled.fill(0)
    mic.record(samples, len(samples))
    magnitude = normalized_rms(samples)

    if magnitude > 0:
        pot_value = potentiometer.value / 65535 * (sound_max - sound_min) + sound_min
        oled.text(f"Threshold: {pot_value:.1f} dB", 10, 35, 1)

        sound_level_dB = 20 * log10(magnitude)
        oled.text("Sound Level (dB):", 15, 5, 1)
        oled.text(f"{sound_level_dB:.2f} dB", 40, 20, 1)

        if sound_level_dB > pot_value:
            pixels.fill((255, 0, 0))
            oled.text("SOUND LEVEL HIGH!", 15, 50, 1)
        else:
            pixels.fill((0, 255, 0))
            oled.text("SOUND LEVEL LOW!", 20, 50, 1)

    oled.show()
    time.sleep(0.1)
```

Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Import Necessary Libraries

Libraries

```
1 import board, time, neopixel, busio, array, math, audiobusio, adafruit_ssd1306
2 from analogio import AnalogIn
```

Line 1: Imports various libraries and modules, such as **board** for pin definitions, **time** for delays, **neopixel** for controlling RGB LEDs, **busio** for I2C communication, **array** for creating arrays, **math** for mathematical operations, **audiobusio** for audio input, and **adafruit_ssd1306** for the SSD1306 OLED display.

Line 2: Import **AnalogIn** from **analogio** library for reading analog input.

Initialize Hardware Components

Hardware Initialization

```
11 mic = audiobusio.PDMIn(board.GP3, board.GP2, sample_rate=32000, bit_depth=16)
12 samples = array.array('H', [0] * 1024)
13
14 sound_min = 30
15 sound_max = 80
```

Line 11 - 12: Initialize audio input using the PDM (Pulse-Density Modulation) method from the microphone connected to pins ***GP3*** and ***GP2***. It sets the sample rate to **32,000** samples per second and a bit depth of **16**. It also initializes an array variable to store the audio samples.

Line 14 - 15: **sound_min** and **sound_max** determine the minimum and maximum sound level thresholds (in decibels) that will be adjusted using the potentiometer.

Define Custom Functions

Calculate Sound in Decibel (dB)

```
17 def log10(x):
18     return math.log(x) / math.log(10)
19
20 def normalized_rms(values):
21     minbuf = sum(values) / len(values)
22     samples_sum = sum((sample - minbuf) ** 2 for sample in values)
23     return math.sqrt(samples_sum / len(values))
```

Line 17 - 18: This function calculates the base 10 logarithm of a given number **x**. In many scientific and engineering applications, sound levels are expressed in decibels (dB), and the logarithm base 10 is commonly used to calculate these levels.

Line 20 - 23: This function calculates the Root Mean Square (RMS) value of a set of audio samples. The RMS value is used to quantify the magnitude of an audio signal.

Enter a Continuous Loop

Record & Process Audio Sample

```
25 while True:
26     oled.fill(0)
27     mic.record(samples, len(samples))
28     magnitude = normalized_rms(samples)
```

Line 27: Records audio samples from the microphone using the **mic.record** method. It captures a number of samples specified by **len(samples)** and stores them in the **samples** array.

Line 28: Calls the **normalized_rms()** function to calculate the Root Mean Square (RMS) of the audio samples. The RMS represents the loudness of the captured sound.

Reads Potentiometer Value & Setup OLED

```
30    if magnitude > 0:
31        pot_value = potentiometer.value / 65535 * (sound_max - sound_min) + sound_min
32        oled.text(f"Threshold: {pot_value:.1f} dB", 10, 35, 1)
33
34        sound_level_dB = 20 * log10(magnitude)
35        oled.text("Sound Level (dB):", 15, 5, 1)
36        oled.text(f"{sound_level_dB:.2f} dB", 40, 20, 1)
```

Line 30: Proceed with the main code only if there's audio detected (not silence).

Line 31: Reads the value of the potentiometer, map it to a range between **sound_min** and **sound_max**, and store the results in the **pot_value** variable. This value represents the threshold for determining whether the sound level is high or low.

Line 32: Print the current threshold value (**pot_value**) on the OLED display, controlled by the potentiometer.

Lines 35 - 36: Update the OLED display to show the "Sound Level (dB)" label, the actual sound level in decibels (**sound_level_dB**), and the threshold value set by the potentiometer (**pot_value**).

Sound Level Conditions

Check Sound Level

```
38        if sound_level_dB > pot_value:
39            pixels.fill((255, 0, 0))
40            oled.text("SOUND LEVEL HIGH!", 15, 50, 1)
41        else:
42            pixels.fill((0, 255, 0))
43            oled.text("SOUND LEVEL LOW!", 20, 50, 1)
44
45    oled.show()
46    time.sleep(0.1)
```

Line 38 - 40: Here, the code checks whether the **sound_level_dB** is greater than the **pot_value**. If it is, it fills the RGB LEDs with a red RGB value of (255, 0, 0) to indicate a high sound level. It also displays "SOUND LEVEL HIGH!" on the OLED display.

Line 41 - 43: If the sound level is not greater than the threshold, it fills the RGB LEDs with green RGB value of (0, 255, 0) to indicate a low sound level. It also displays "SOUND LEVEL LOW!" on the OLED display.

CHALLENGE - SERVO SOUND METER

Having an OLED to display noise decibel is great, however, in practical usage, the OLED may be too small for everyone in the surroundings to notice. In this challenge, we will solve this problem by integrating a servo motor as a noise indicator, reflecting the values displayed on the OLED, but in a physical form with a larger view.

Use the code below to get started and fill in the missing parts based on the description.

A: Include necessary libraries here to run a servo motor. (Refer to Page 89)

B: Include log10 and normalized_rms() functions. Initialize the servo motor pin and perform a servo test by sweeping the motor through its range from 0 to 180 degrees. (Refer to Page 121)

C: Calculate the angle for the servo motor based on sound level in decibels (dB).
Check the angle to make sure it's within 0 to 180 degrees before rotating the servo motor.

```
import board, time, neopixel, busio, array, math, audiobusio, adafruit_ssd1306
from analogio import AnalogIn
                                                                          A
i2c = busio.I2C(board.GP5, board.GP4)
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
potentiometer = AnalogIn(board.GP28)

pixels = neopixel.NeoPixel(board.GP14, 5, brightness=0.2)
pixels.fill(0)

mic = audiobusio.PDMIn(board.GP3, board.GP2, sample_rate=32000, bit_depth=16)
samples = array.array('H', [0] * 1024)
                                        Adjust sample size for better
                                        accuracy & remove sudden spike.
sound_min = 30
sound_max = 80
                                                                          B
while True:
    oled.fill(0)
    mic.record(samples, len(samples))
    magnitude = normalized_rms(samples)

    if magnitude > 0:
        pot_value = potentiometer.value / 65535 * (sound_max - sound_min) + sound_min
        oled.text(f"Threshold: {pot_value:.1f} dB", 10, 35, 1)

        sound_level_dB = 20 * log10(magnitude)
        oled.text("Sound Level (dB):", 15, 5, 1)
        oled.text(f"{sound_level_dB:.2f} dB", 40, 20, 1)
                                                                          C

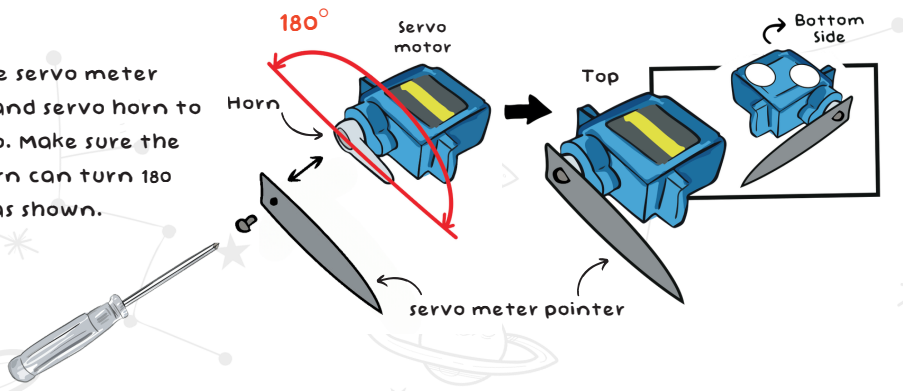
        if sound_level_dB > pot_value:
            pixels.fill((255, 0, 0))
            oled.text("SOUND LEVEL HIGH!", 15, 50, 1)
        else:
            pixels.fill((0, 255, 0))
            oled.text("SOUND LEVEL LOW!", 20, 50, 1)
    time.sleep(0.1)
    oled.show()
```

CHALLENGES

SERVO SOUND METER ACCESSORY

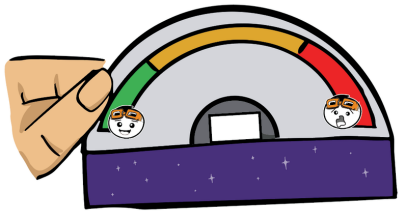
Step 1:

Screw the servo meter pointer and servo horn to the servo. Make sure the servo horn can turn 180 degrees as shown.



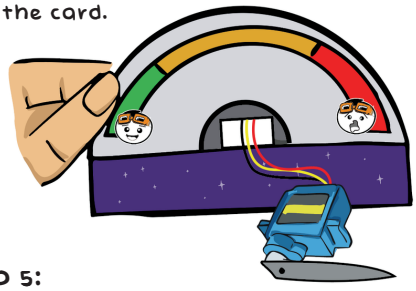
Step 2:

Push through the middle to make a hole on the card accessory.



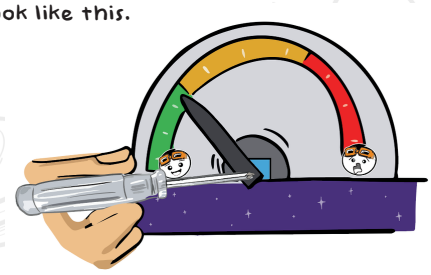
Step 3:

Plug in the servo motor from the front of the card.



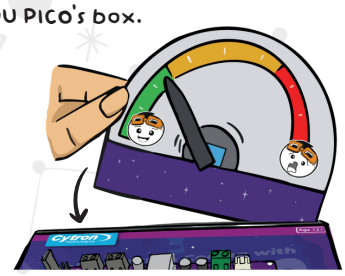
Step 4:

The assembled piece will look like this.



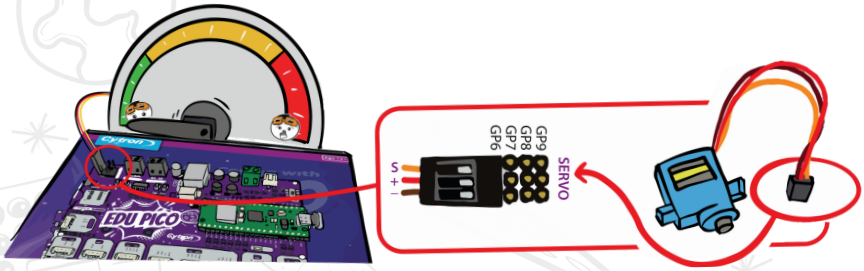
Step 5:

Slot the card into the slit on the EDU PICO's box.



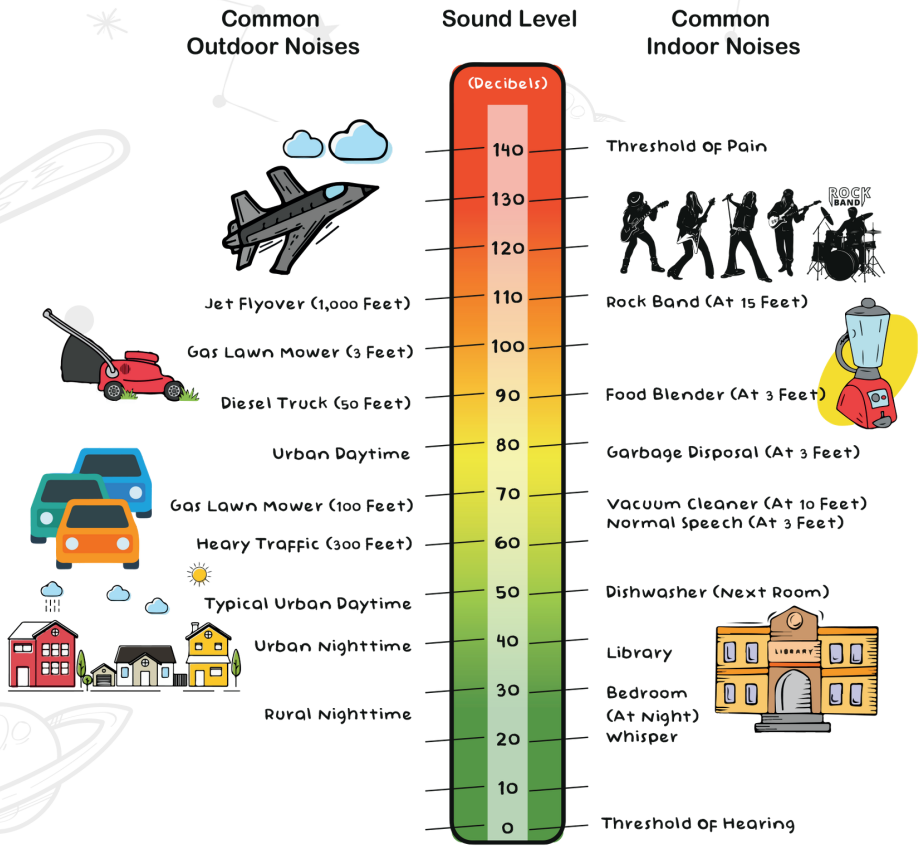
Step 6:

Connect Servo Motor Cable to EDU PICO board at GP6 and now you're ready!



WHAT'S NEXT?

Once you're done with the program, here's what you can do to make this project more exciting. Take a walk with your EDU PICO with the noise monitoring program still running. Try to identify the noise level in each location indicated below. Is the measured noise level within the safe range as shown in the graph? If not, what action can be taken to reduce the noise level in the area?



Sound is perceived differently by every individual. The PDM microphone helps to narrow that gap.

CHAPTER 7

Smart Classroom

DC Motor & Relay

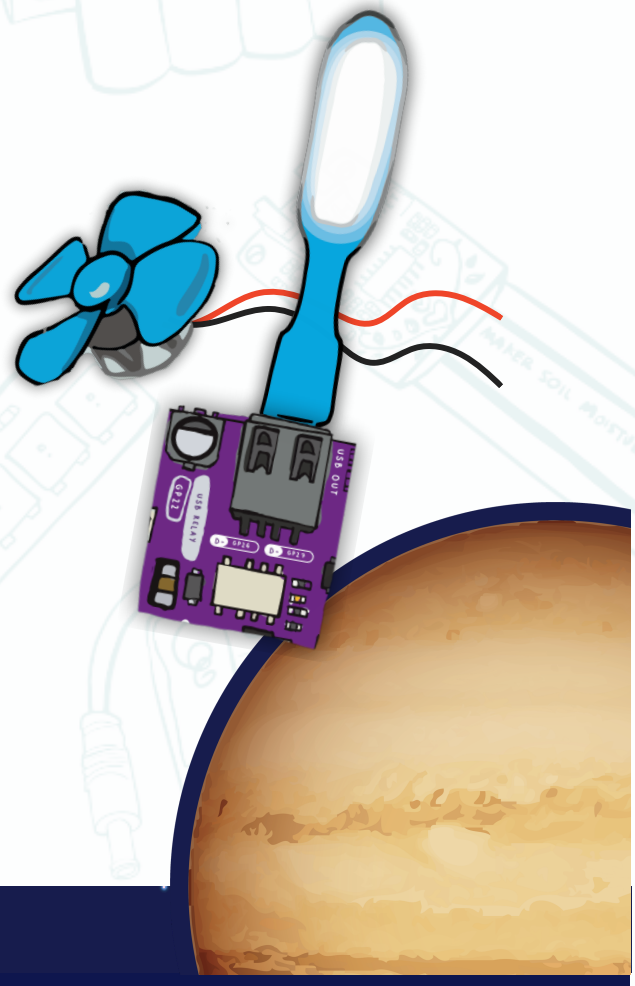
Introduction to DC Motor

Introduction to Relay

Project: Smart Classroom

In this chapter, we will learn to control a DC motor. The DC motor is the main component that act like muscles behind automated systems. They're responsible for making things spin, rotate, and move!

Relay switches, on the other hand, act as the traffic directors, allowing you to control high-power electrical devices using a low-power signal from your EDU PICO.

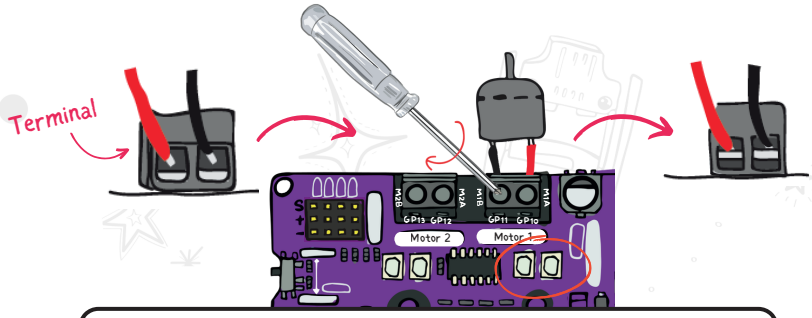


Introduction to DC Motor

In this lesson, we will finally get our hands on a DC motor. This particular invention serves as the cornerstone of automation, converting electric energy into motion. Its a versatile component that can power an array of mechanisms, from spinning wheels to robotic arms. Consider it as a tool that can transform your ideas into reality. Let's get this thing moving!

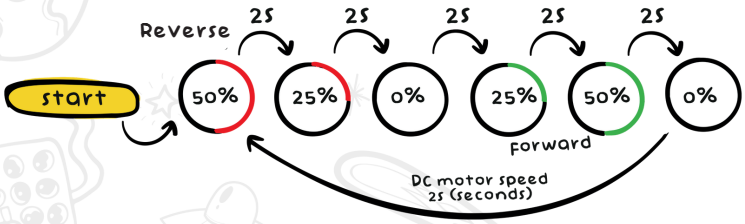
How Does This Activity Work?

- **Libraries:** board, time, pwmio, adafruit_motor.
- **DC Motor Configuration:**
  - PWM_M1A to ***GP10***, PWM_M1B to ***GP11***.
  - Connect DC Motor to M1A & M1B terminals as shown below:



Press the M1A & M1B buttons to test the motor. If the motor is not spinning, make sure the wire connection at the terminal is secure and EDU PICO is powered on.

- **Output:**
  - The DC motor speed will be printed on the shell console.
  - It then continuously rotates at a different speed and direction starting from reverse direction at 50% (**-0.5**) to 25% (**-0.25**), and to a halt at 0%.
  - Then it proceeds to forward direction of 25% (**0.25**) and 50% (**0.5**) speed.
  - Each speed will last for **2** second, then the cycle repeats.



Code

DC_MOTOR_MODULE.py

```
1  import board, time
2  from pwmio import PWMOut
3  from adafruit_motor import motor
4
5  PWM_M1A = PWMOut(board.GP10,frequency=10000)
6  PWM_M1B = PWMOut(board.GP11,frequency=10000)
7  motor = motor.DCMotor(PWM_M1A, PWM_M1B)
8  speed_mode = [-0.5, -0.25, 0, 0.25, 0.5, 0]
9
10 while True:
11     for speed in speed_mode:
12         print("Speed:", speed * 100)
13         motor.throttle = speed
14         time.sleep(2)
```

Configure DC MOTOR with gp10 and gp11



Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Import Necessary Libraries

Libraries

```
1 import board, time
2 from pwmio import PWMOut
3 from adafruit_motor import motor
```

Line 2: **pwmio** module allows the control of pulse-width-modulation (PWM) output.

Line 3: **adafruit_motor** module provides motor control functionality.

Initialize Hardware Components

Configure the DC Motor

```
5 PWM_M1A = PWMOut(board.GP10,frequency=10000)
6 PWM_M1B = PWMOut(board.GP11,frequency=10000)
7 motor = motor.DCMotor(PWM_M1A, PWM_M1B)
8 speed_mode = [-0.5, -0.25, 0, 0.25, 0.5, 0]
```

Line 5 - 6: Initializes two PWM outputs to control the DC motor. **PWM_M1A** and **PWM_M1B**, on ***GP10*** and ***GP11*** pins respectively. The frequency parameter is set to **10,000 Hz** (10 kHz), which defines the frequency of the PWM signal.

Line 7: An instance of a DC motor is created using the **motor.DCMotor** class. This instance is named **motor** and is configured to use the **PWM_M1A** and **PWM_M1B** PWM outputs to control the motor.

Line 8: A list called **speed_mode** is created, which contains a set of speed values. The values in this list range from **-0.5** to **0.5** with a step size of **0.25**. These values represent different speed settings for the motor, ranging from 50% (**-0.5**) reverse speed to 50% (**0.5**) forward speed. (Note that reversing speed will rotate the motor in the opposite direction)

Enter a Continuous Loop

Main Loop

```
10 while True:
11     for speed in speed_mode:
12         print("Speed:", speed * 100)
13         motor.throttle = speed
14         time.sleep(2)
```

Line 11: A for loop that iterates through the elements of the **speed_mode** array.

Line 12 - 14: Print the current speed percentile to the shell console and sets the throttle (**speed**) of the DC motor to the current speed level. For an example:

- 1st iteration: **speed** = **-0.5** (Reverse 50% speed)
- 2nd iteration: **speed** = **-0.25** (Reverse 25% speed)
- 3rd iteration: **speed** = **0** (Stop)

...and so on. After arriving at the final element in the array (**speed** = 0%), the for loop will reiterate to speed = **-0.5**.

Activity with AI

Prompt Framework: DC Motor

Task: Use the potentiometer to control the speed and direction of the DC motor.

Step 1: Teach the AI

Copy and paste your working code from **Page 130** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Here is my code for potentiometer:
<PASTE YOUR CODE FROM PAGE 109>

Here is my current code:
<PASTE YOUR CODE FROM PAGE 130>

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

Prompt #2:
"Use the potentiometer to control the DC motor speed and direction. Turning the knob to the left makes the motor spin backward, center stops it, and turning right makes it spin forward. The further I turn, the faster it spins."

Step 3: Check the Answer

Copy and paste the code in Thonny, then run the code. Check if turning the potentiometer knob to the left spins the motor backward, turning it to the center stops the motor and turning it to the right spins the motor forward.

MINI ACTIVITY

From the previous chapter, we learnt how to operate the potentiometer module on the EDU PICO, now, it is time we make use of that knowledge in this mini activity. Let's program our EDU PICO to control the DC motor speed using the potentiometer. Sounds simple right? Let's give it a try!

Fill In The Blank

```
import board, time
                                          A
from pwmio import PWMOut
from adafruit_motor import motor

PWM_M1A = PWMOut(board.GP10,frequency=10000)
PWM_M1B = PWMOut(board.GP11,frequency=10000)
motor = motor.DCMotor(PWM_M1A, PWM_M1B)
                                          B

while True:          C
    speed =                  / 65535
    motor.throttle = speed
    print("Speed:", speed)
    time.sleep(0.1)
```

A: Include necessary libraries required to function the potentiometer. (Refer to Page 109)

B: Initialize the pin used for the potentiometer. Make sure to assign a suitable variable upon reading the signal from the pin.

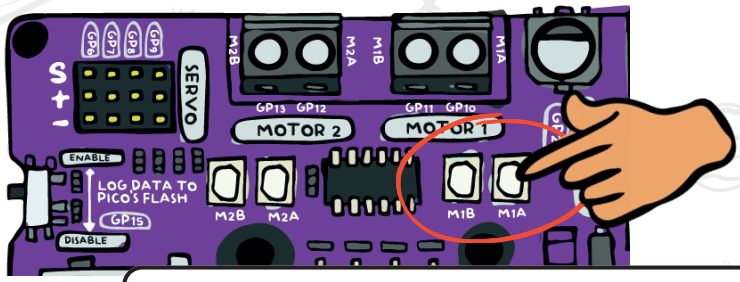
C: Call the variable value assigned from **B** into the following formula. Dividing the potentiometer raw values by **65535** will provide you a value range of **0** to **1** which will then be directly translated to the DC motor speed.

THE MORE YOU KNOW

Electric vehicles (EVs) are becoming increasingly popular nowadays, but did you know that the concept of an electric car with a DC motor isn't a recent innovation? The first functional electric car was built in the early 19th century, and it was powered by a DC motor.

DC Motor

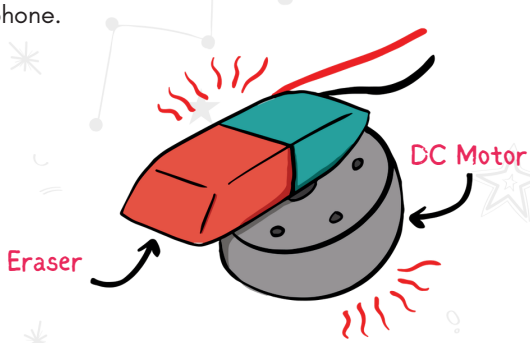
There are two DC motor terminals on the EDU PICO board, which means you can connect two DC motors and control both at the same time!



If you haven't noticed, the EDU PICO 2 has a built-in motor test circuit that allows you to check the connected DC motor in forward or reverse direction. Give it a try!

Converting a DC motor into a vibrating haptic feedback device is one of the many popular applications of a DC motor. Haptic feedback is a technology that enriches user experiences by providing tactile sensations in response to digital interactions, for example, the micro-vibration you get when typing on a smartphone.

To create a vibration from your DC motor, you must first attach the motor with an eccentric weight (which is required for vibration). This is typically a small disc or something as simple as an eraser, easy enough that you can secure it to the motor's shaft as shown on the right.

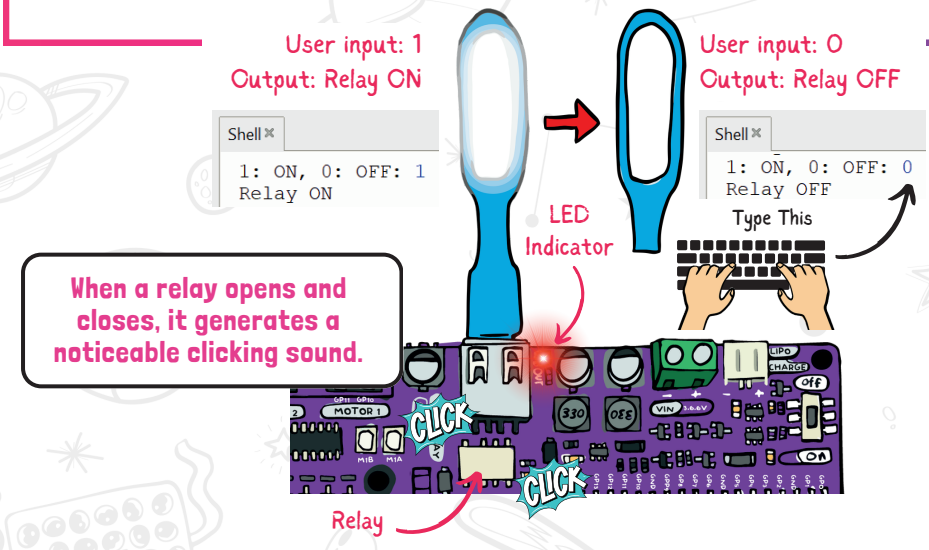


Introduction to Relay

In this lesson, we will learn how to operate the USB relay on the EDU PICO, but more importantly, to understand how a relay works. A relay is an electromechanical device that acts as a switch controlled by an electrical signal. It allows a low-power circuit to control a high-power circuit, making it an essential component in various electrical and electronic applications. In this case, we will use the EDU PICO to control the ON / OFF of the relay, which will control the ON / OFF of a USB LED light stick connected to the USB port!

How Does This Activity Work?

- **Libraries:** board, digitalio.
- **USB Relay Configuration:**
  - Initialize relay to pin ***GP22*** and set the pin direction to output.
- **Output:**
  - The program will begin by prompting the user to input either **1** for ON or **0** for OFF at the shell console to control the on and off of the USB relay.
  - If the input is **1**, the USB relay will turn ON and you will notice the relay LED indicator next to the USB port will light up, indicating the relay is switched ON.
  - If the input is **0**, the USB relay will turn OFF.
  - Connect the USB LED light stick to the USB relay to test out the USB relay functionality.



Code

RELAY_MODULE.py

```
1  import board, digitalio
2
3  relay = digitalio.DigitalInOut(board.GP22)
4  relay.direction = digitalio.Direction.OUTPUT
5
6  while True:
7      user_input = input("1: ON, 0: OFF \nYour choice: ")
8      state = int(user_input)
9
10     if state == 0:
11         print("Relay OFF")
12         relay.value = False
13     elif state == 1:
14         print("Relay ON")
15         relay.value = True
16     else:
17         print("Invalid input. Please enter 0 for OFF or 1 for ON.")
```

Shell

```
1: ON, 0: OFF
Your choice: 1
Relay ON
1: ON, 0: OFF
Your choice: 0
```

Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Enter a Continuous Loop

Main Loop

```
6  while True:
7      user_input = input("1: ON, 0: OFF \nYour choice: ")
8      state = int(user_input)
```

Line 7: Prompts the user to enter either **1** to turn ON or **0** to turn OFF the USB relay using the input function at the shell console. "\nYour choice: " will wait for the user's input in the next line. The input is then stored in the **user_input** variable as a string.

Line 8: Converts the user's input into an integer using int(**user_input**) and stores the result in a **state** variable.

Process User Input - On / Off Relay

```
10      if state == 0:
11          print("Relay OFF")
12          relay.value = False
13      elif state == 1:
14          print("Relay ON")
15          relay.value = True
16      else:
17          print("Invalid input. Please enter 0 for OFF or 1 for ON.")
```

Line 10 - 12: If the state is **0**, the "Relay OFF" text will be printed on the shell console and the **relay.value** will be set to false, turning the relay off.

Line 13 - 15: If the state is **1**, "Relay ON" will be printed on the shell console and the **relay.value** will be set to true, turning the relay on.

Line 16 - 17: If the state is neither **0** nor **1**, the script prints "Invalid input. Please enter 0 for OFF or 1 for ON."

THE MORE YOU KNOW

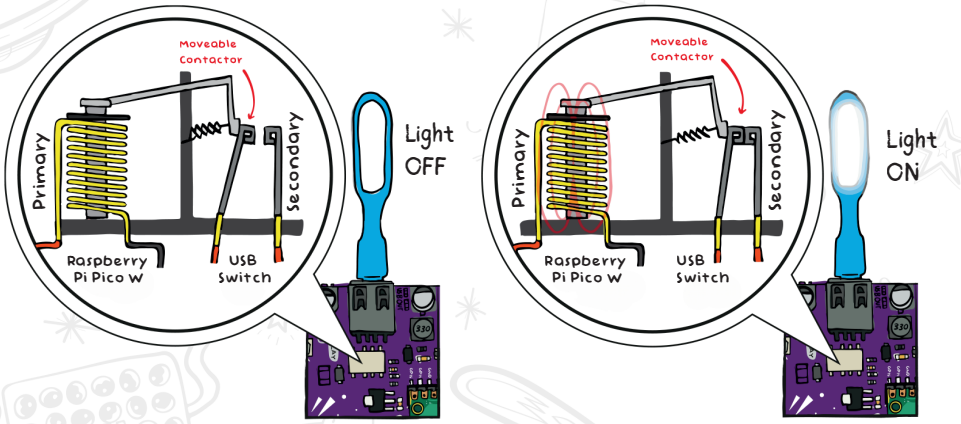
Relay

A relay has two circuits in its body: the primary circuit and the secondary circuit.

The **primary circuit** mainly receives an external signal that controls the ON / OFF operation of the relay; in this case, it's connected to EDU PICO ***GP22***.

The **secondary circuit** is connected to the load or the output component; in this case, the secondary side is connected to the USB port which is also connected to the LED stick.

When current flows through a relay's coil, it creates an electromagnetic field (like a magnet) that attracts the movable contactor which will connect and complete the circuit on the secondary side.



Activity with AI

Prompt Framework: Relay

Task: Use Button A to toggle the relay ON and OFF (like a light switch).

Step 1: Teach the AI

Copy and paste your working code from **Page 136** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Button A connected to GP0. (Provide button code for better outcome)

Here is my current code:
<PASTE YOUR CODE FROM PAGE 136>"

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

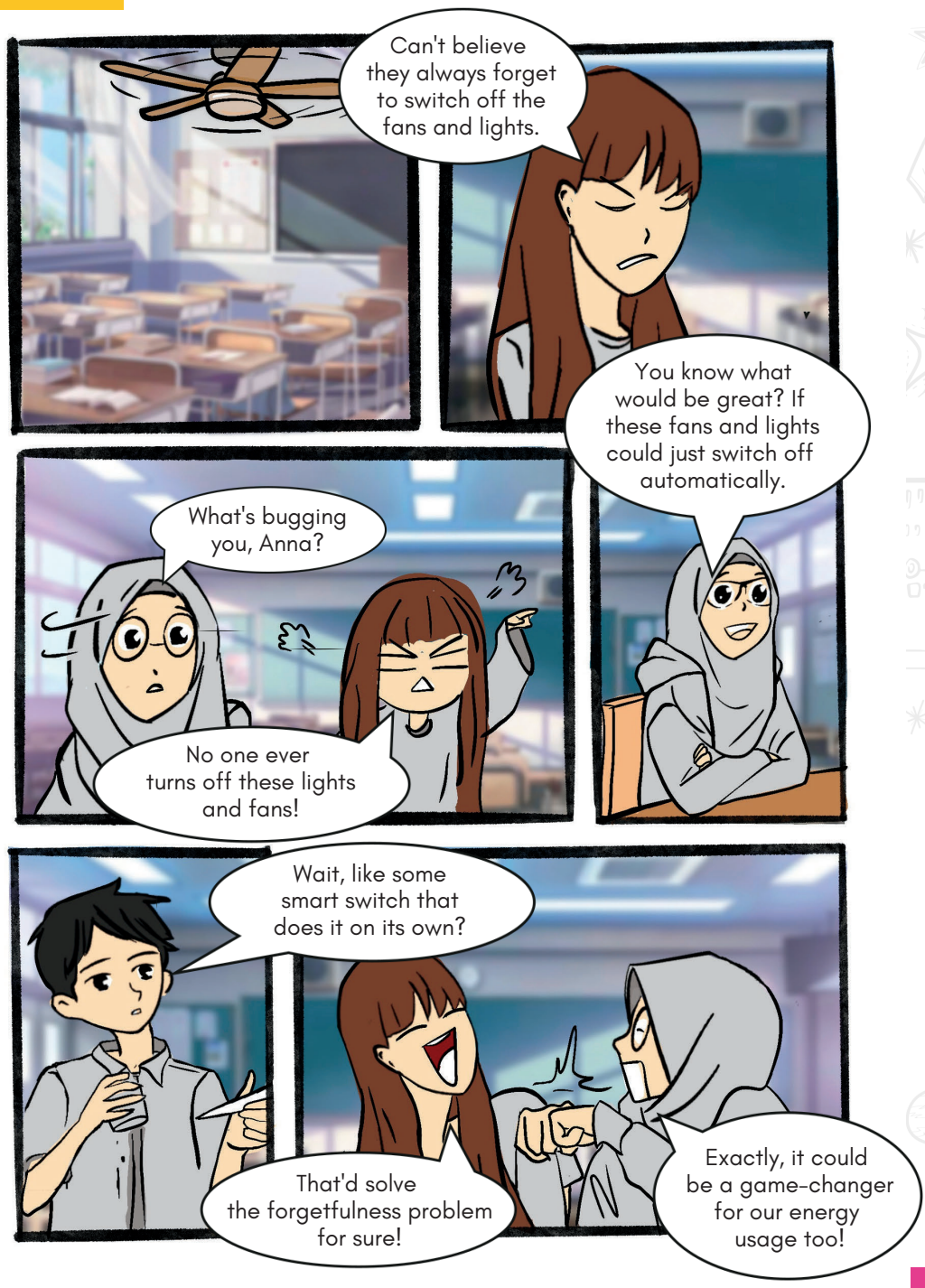
Prompt #2:
"Use Button A to control the relay. When I press the button once, turn the relay ON. When I press the button again, turn the relay OFF."

Step 3: Check the Answer

Copy and paste the code in Thonny, then run the code. Check that when you press Button A, the relay turns ON and the LED stick lights up. When you press Button A again, the relay turns OFF and the LED stick turns off as well.

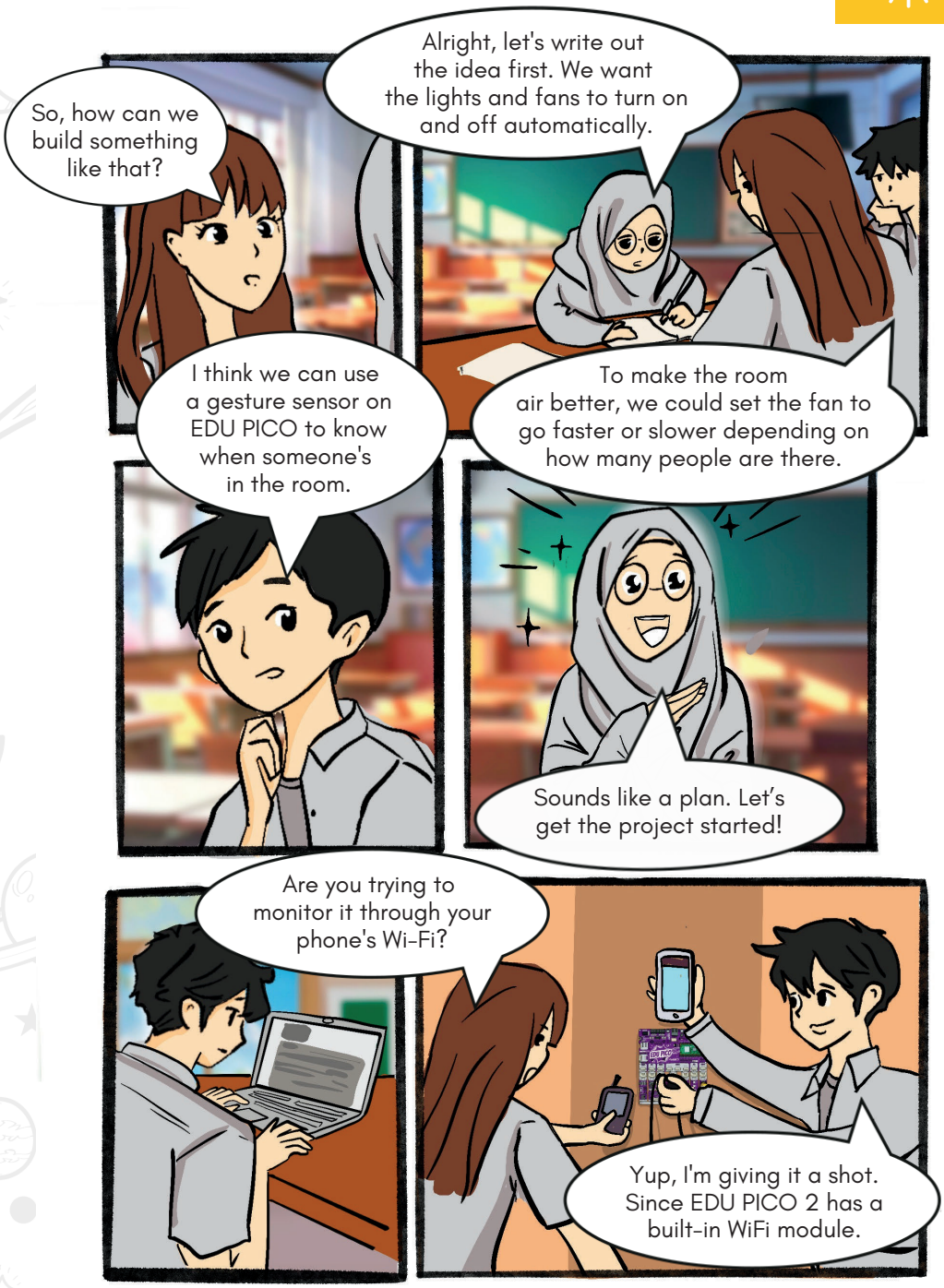
7 AFFORDABLE AND CLEAN ENERGY

PROLOGUE: SMART CLASSROOM



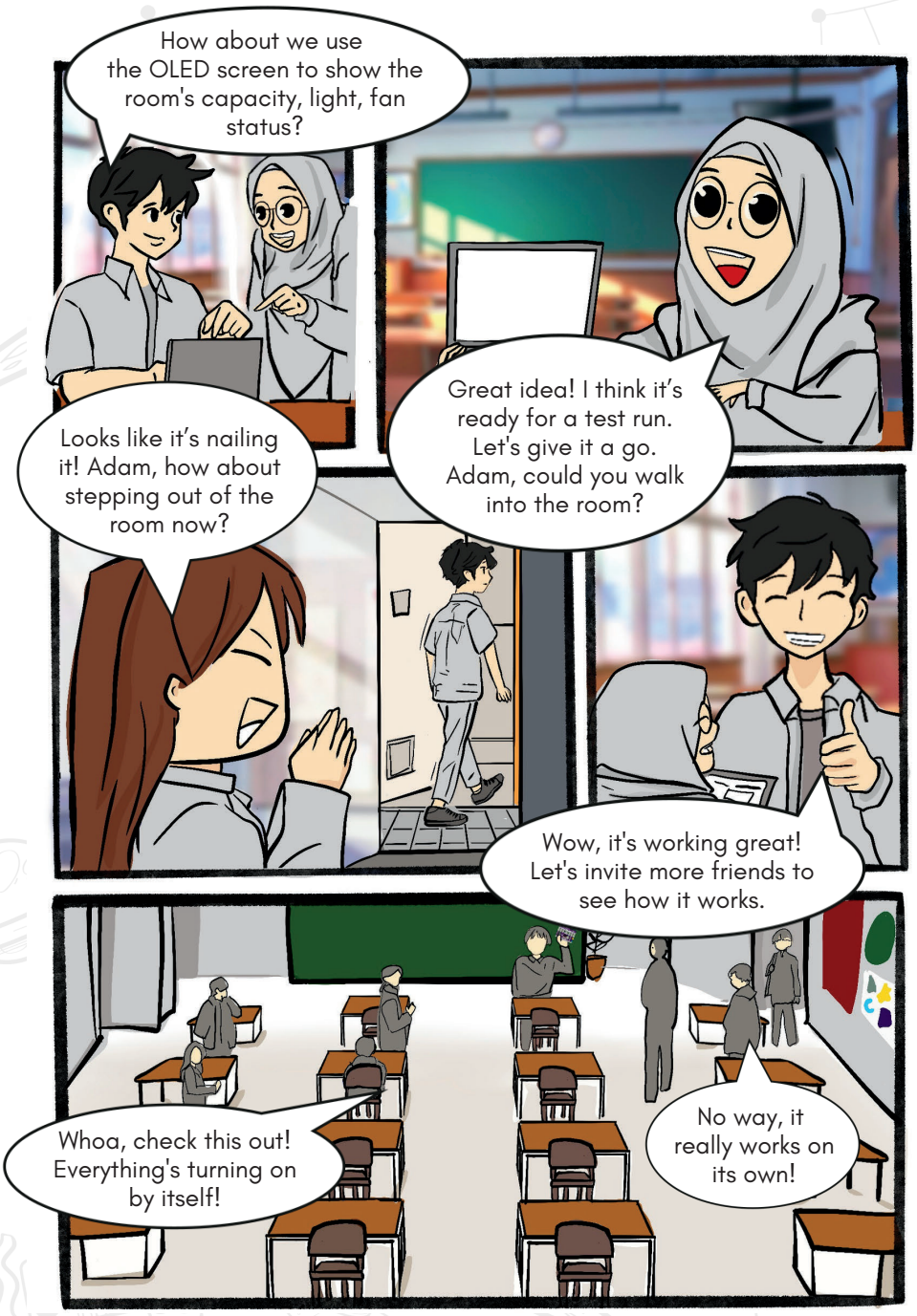
PROLOGUE: SMART CLASSROOM

7 AFFORDABLE AND CLEAN ENERGY



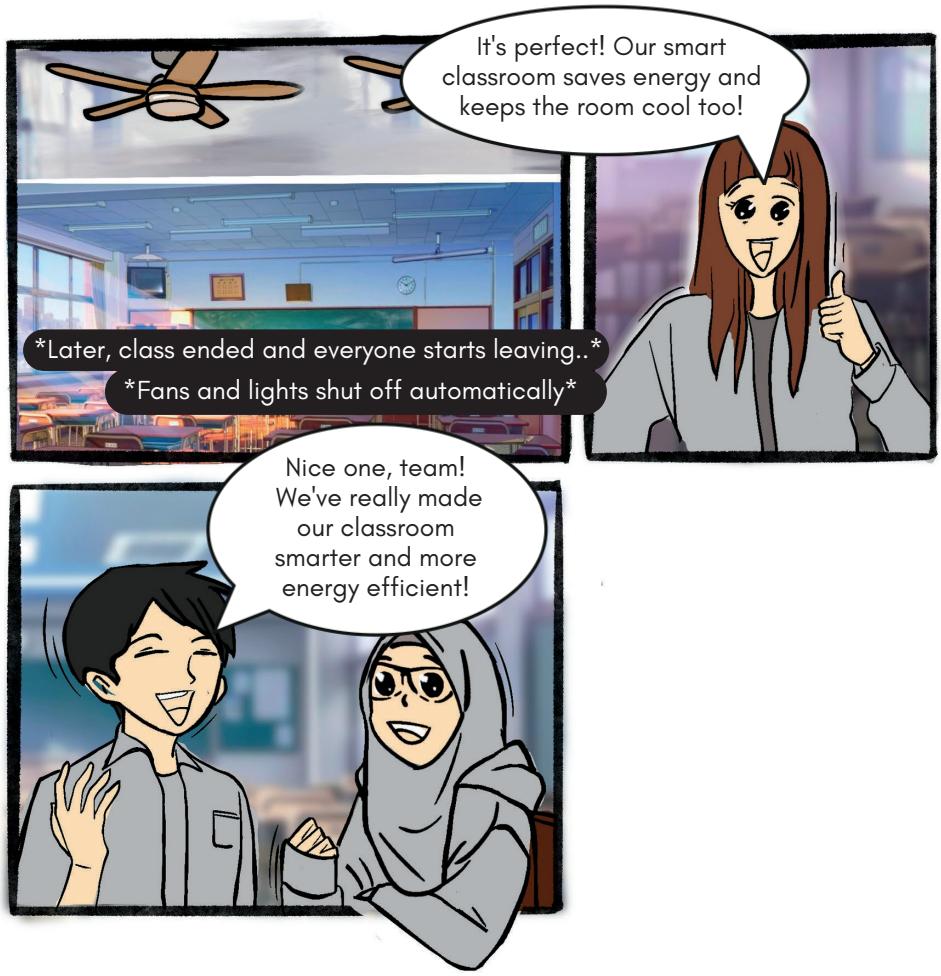
7 AFFORDABLE AND CLEAN ENERGY

PROLOGUE: SMART CLASSROOM



PROLOGUE: SMART CLASSROOM

7 AFFORDABLE AND CLEAN ENERGY



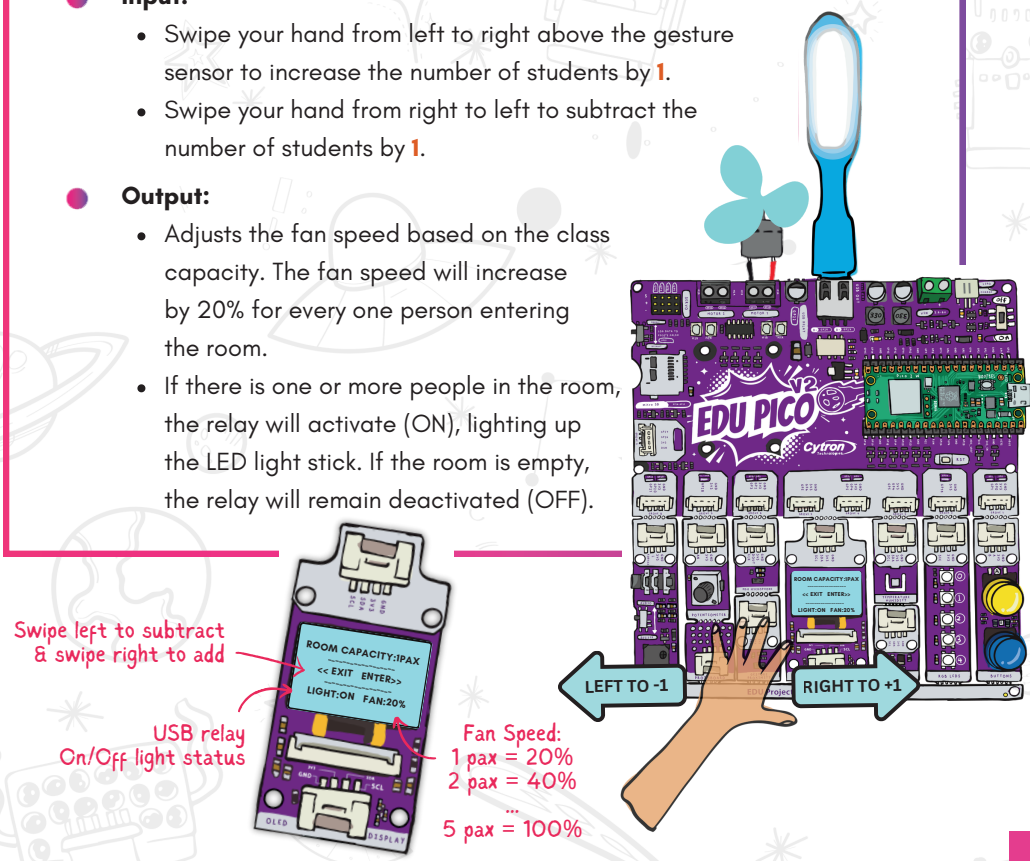
[Caution: Never attempt to handle or modify AC power appliances on your own. Always ask an adult or qualified professional for help to avoid the risk of electric shock or injury.]

Smart Classroom

In this project, we will learn to integrate PAJ7620 gesture sensor to track students' movement when entering and leaving the room. The goal is to create a smart classroom that optimizes energy usage based on the number of occupants. To achieve that, we will program the relay to turn on and off a USB LED light stick based on the presence of students, as well as to program the DC motor fan to regulate the room's temperature according to the number of occupants.

How Does This Activity Work?

- **Libraries:** board, digitalio, busio, edupico2_paj7620, pwmio, adafruit_motor, adafruit_ssd1306.
- **DC Motor Configuration:** PWM_M1A to ***GP10***, PWM_MIB to ***GP11***.
- **USB Relay Configuration:** ***GP22***.
- **OLED and PAJ7620 I2C Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Input:**
  - Swipe your hand from left to right above the gesture sensor to increase the number of students by **1**.
  - Swipe your hand from right to left to subtract the number of students by **1**.
- **Output:**
  - Adjusts the fan speed based on the class capacity. The fan speed will increase by 20% for every one person entering the room.
  - If there is one or more people in the room, the relay will activate (ON), lighting up the LED light stick. If the room is empty, the relay will remain deactivated (OFF).



Code

PROJECT_SMART_CLASSROOM.py

```
import board, digitalio, busio, adafruit_ssd1306
from pwmio import PWMOut
from adafruit_motor import motor
import edupico2_paj7620

i2c = busio.I2C(board.GP5, board.GP4)
sensor = edupico2_paj7620.PAJ7620(i2c)

PWM_M1A = PWMOut(board.GP10, frequency=10000)
PWM_M1B = PWMOut(board.GP11, frequency=10000)
motor_instance = motor.DCMotor(PWM_M1A, PWM_M1B)
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
relay = digitalio.DigitalInOut(board.GP22)
relay.direction = digitalio.Direction.OUTPUT

class_num = 0
current_speed = 0.0

oled.invert(True)
oled.text("--------------------", 5, 15, 1)
oled.text("--------------------", 5, 40, 1)
oled.text("ROOM CAPACITY: ", 5, 7, 1)
oled.text("Light:", 5, 50, 1)
oled.text("Fan:", 65, 50, 1)
oled.text("<< EXIT    ENTER >>", 7, 27, 1)
oled.show()

def handle_gesture():
    global class_num, current_speed

    oled.fill_rect(90, 7, 35, 7, 0)
    oled.fill_rect(40, 50, 20, 7, 0)
    oled.fill_rect(90, 50, 35, 7, 0)
    gesture = sensor.gesture()

    if gesture & sensor.LEFT:
        class_num = max(0, class_num - 1)
    elif gesture & sensor.RIGHT:
        class_num = min(5, class_num + 1)
```

```
    current_speed = class_num * 0.2
    motor_instance.throttle = current_speed
    oled.text("{}%".format(current_speed * 100), 90, 50, 1)

    relay.value = class_num > 0
    oled.text("ON" if class_num > 0 else "OFF", 40, 50, 1)

    class_status = f"{class_num} PAX" if class_num < 5 else "FULL"
    oled.text(class_status, 90, 7, 1)
    oled.show()

try:
    while True:
        handle_gesture()

finally:
    oled.fill(1)
    oled.show()
    print("deinit I2C")
    i2c.deinit()
```

Click the Green Button to run the code and Red Button to stop.

What the Code Does

Import Necessary Libraries

Libraries

```
import board, digitalio, busio, adafruit_ssd1306
from pwmio import PWMOut
from adafruit_motor import motor
from edupico2_paj7620
```

The code imports the required libraries and modules to enable the board, digital input/output, I2C communication, OLED display, PWM control, DC motor, and the PAJ7620 gesture sensor.

Initialize Hardware Components

Initialize I2C, PAJ7620 Sensor, PWM for Motor, OLED Display, and Relay

```
6  i2c = busio.I2C(board.GP5, board.GP4)
7  sensor = edupico2_paj7620.PAJ7620(i2c)
8
9  PWM_M1A = PWMOut(board.GP10, frequency=10000)
10 PWM_M1B = PWMOut(board.GP11, frequency=10000)
11 motor_instance = motor.DCMotor(PWM_M1A, PWM_M1B)
12 oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
13 relay = digitalio.DigitalInOut(board.GP22)
14 relay.direction = digitalio.Direction.OUTPUT
15
16 class_num = 0
17 current_speed = 0.0
```

These lines initialize the I2C interface, the PAJ7620 gesture sensor, PWM control for the DC motor, OLED display, and the relay.

Line 6: Configure I2C communication to ***GP5*** and ***GP4***.

Line 7: Create an instance of the PAJ7620 gesture sensor.

Line 9 - 11: PWM outputs for controlling the DC motor.

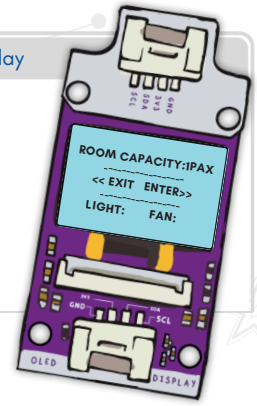
Line 12: Configure the OLED display with the assigned I2C pins.

Line 13 - 14: Configure ***GP22*** as digital output for controlling the relay.

Line 16 - 17: Initialize variables for class capacity and current fan speed to zero.

Set Up the OLED Display

```
19 oled.invert(True)
20 oled.text("--------------------", 5, 15, 1)
21 oled.text("--------------------", 5, 40, 1)
22 oled.text("ROOM CAPACITY: ", 5, 7, 1)
23 oled.text("Light:", 5, 50, 1)
24 oled.text("Fan:", 65, 50, 1)
25 oled.text("<< EXIT    ENTER >>", 7, 27, 1)
26 oled.show()
```



The OLED display is set up with an initial text, providing a baseline for the user to navigate when reading information relating to room capacity, light, and fan.

Define a Custom Function

handle_gesture Function

```
28 def handle_gesture():
29     global class_num, current_speed
30
31     oled.fill_rect(90, 7, 35, 7, 0)
32     oled.fill_rect(40, 50, 20, 7, 0)
33     oled.fill_rect(90, 50, 35, 7, 0)
34     gesture = sensor.gesture()
35
36     if gesture & sensor.LEFT:
37         class_num = max(0, class_num - 1)
38     elif gesture & sensor.RIGHT:
39         class_num = min(5, class_num + 1)
```

Line 29: Initializes **class_num** and **current_speed** as global variables which enable their values to persist across different class to the **handle_gesture** function. If local variables were used, their values would be re-initialized every time the function is called.

Line 31 - 33: Clear specific areas on the OLED display where information will be updated. It uses **oled.fill_rect** to clear rectangles at a specified area.

Line 34: Reads the gesture data from the PAJ7620 sensor. The function returns a bitmask value, ie: 0b0100 or 0b1000.

Line 36 - 39: The **&** symbol is a bitwise **AND** operator. The line checks whether the LEFT gesture is detected by reading the bitmask value. If detected, it will reduce the current class count by 1. With max(0, ...), the class_num will never go below 0. This applies to the RIGHT gesture too but instead of reducing the class count, it increases by 1 when the gesture is detected.

Adjust Fan Speed

```
41     current_speed = class_num * 0.2
42     motor_instance.throttle = current_speed
43     oled.text("{}%".format(current_speed * 100), 90, 50, 1)
```

Line 41 - 43: Adjust the fan speed **current_speed** based on the class capacity. For each person entered, the fan speed will increase by 20% (**class_num** * **0.2**). The **motor_instance.throttle** property is used to set the fan speed.

The greater the number of people in the room, the faster the fan spins to ventilate the room.

Display Information on OLED

```
45      relay.value = class_num > 0
46      oled.text("ON" if class_num > 0 else "OFF", 40, 50, 1)
47
48      class_status = f"{class_num} PAX" if class_num < 5 else "FULL"
49      oled.text(class_status, 90, 7, 1)
50      oled.show()
```

Line 45: Controls the state of the relay based on the value of **class_num**. If **class_num** is greater than **0**, **relay.value** will be set to true, turning the relay ON. If **class_num** is **0** or less, **relay.value** will be set to false, turning the relay OFF.

Line 46: Updates the OLED display to show whether the relay is ON or OFF. It uses a conditional expression where if **class_num** is greater than **0**, the text is set to "ON"; otherwise, it is set to "OFF".

Line 48: Creates a string **class_status** that displays the current room capacity status. If **class_num** is less than **5**, the string shows the number of people in the room. If **class_num** is **5** or more, the string displays "FULL".

Line 49 - 50: Updates the OLED display to show the room capacity status.

Enter a Continuous Loop

Main Loop, try...finally

```
52  try:
53      while True:
54          handle_gesture()
55
56  finally:
57      oled.fill(1)
58      oled.show()
59      print("deinit I2C")
60      i2c.deinit()
```

Line 52 - 54: The **try** block contains the main loop of the program. The **while True** loop ensures that the **handle_gesture** function is continuously called.

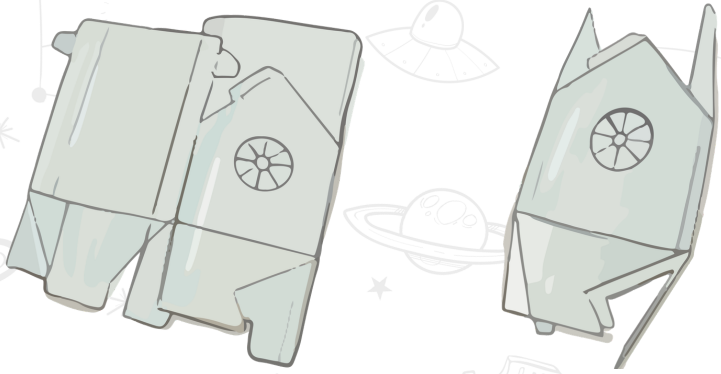
Line 56 - 58: The **finally** block is used for cleanup operations which include resetting the OLED display to its original state.

Line 60: Deinitializes the I2C interface. *[Note: This is important for releasing the resources used by I2C before the program terminates.]*

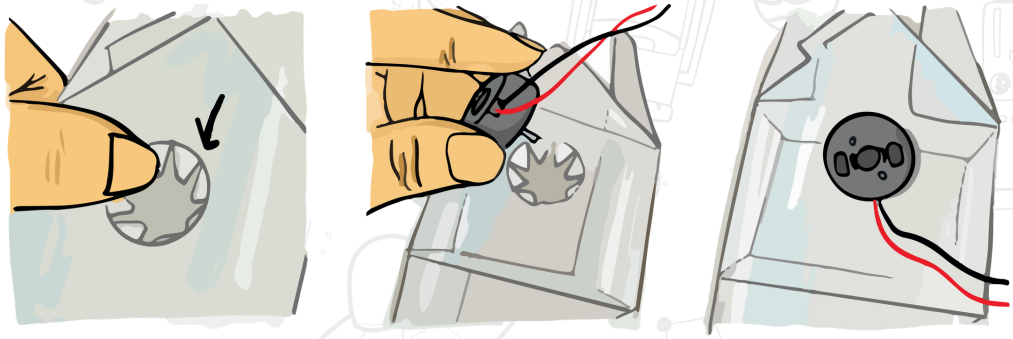
WHAT'S NEXT?

HOUSE ACCESSORY - DC MOTOR FAN

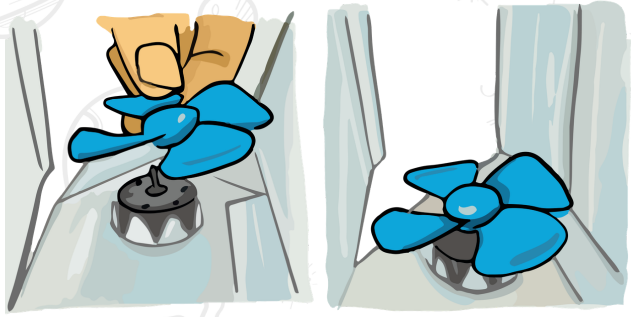
Step 1:
Prepare the house accessory by opening it into a shape of a box.



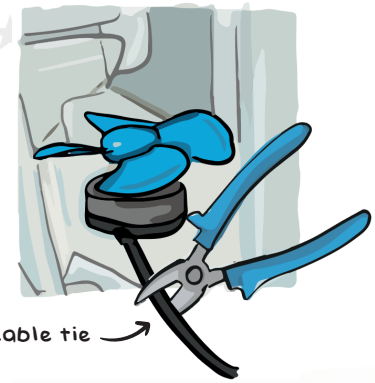
Step 2:
Push the DC motor hole inwards and push the DC motor through the hole carefully.



Step 3:
Attach the DC motor fan blade from the inside of the house.

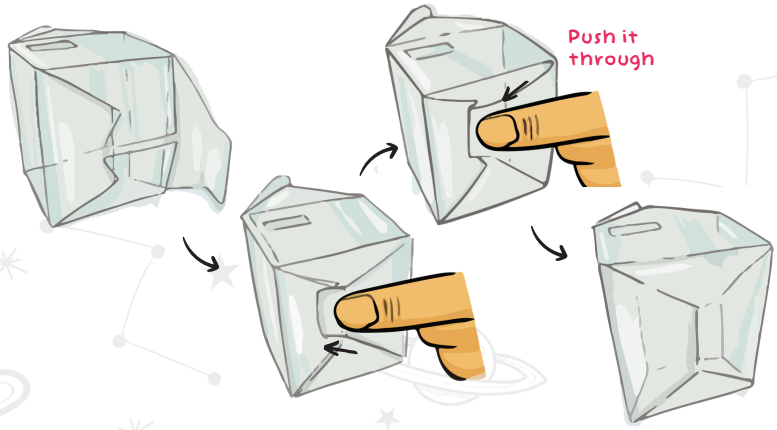


Step 4:
Secure a cable tie around the DC motor.



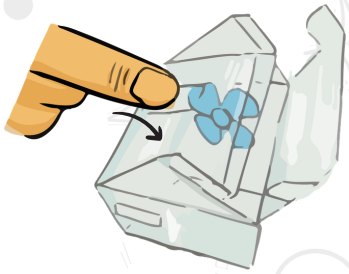
Step 5:

Fold the bottom of the house with the following sequence.
This will hold the house in place.



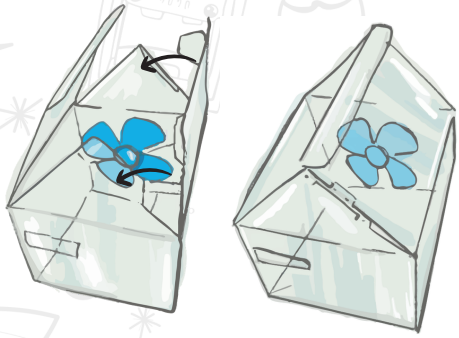
Step 6:

Push the bigger roof downwards so that it keeps the roof closed by default.



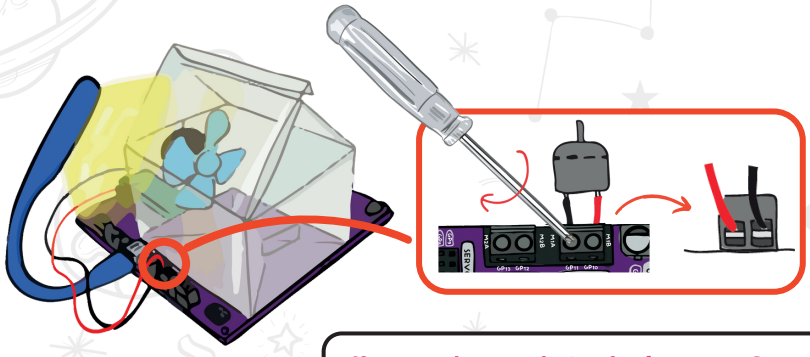
Step 7:

Fold the other side of the roof and slot the sides down as shown.



Step 8:

Complete the project by connecting the DC Motor wires to the EDU PICO terminal and position the house next to the LED light stick as shown.



Now you're ready to deploy your Smart Classroom! Remember to run your code and test run the project following the instructions in Page 143.

CHAPTER 8

Climate Control Greenhouse

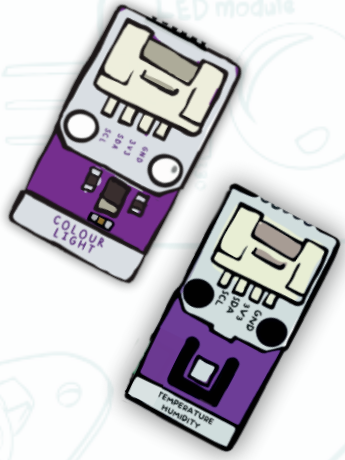
Light Sensor & Temperature Humidity Sensor

Introduction to Light Sensor

Introduction to Temperature & Humidity Sensor

Project: Climate Control Greenhouse

Bonus: Wireless Network (AP-Mode)



In this chapter, we will learn to read the data values from the OPT4048 light sensor module and the AHT20 temperature and humidity sensor.

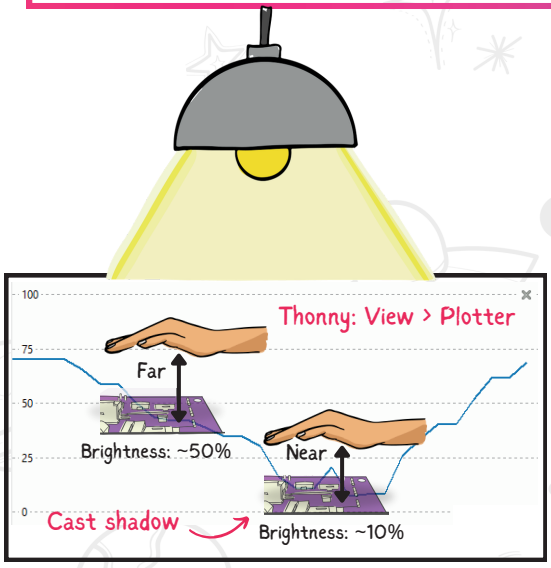
Imagine having the ability to constantly monitor and adjust the conditions within a greenhouse to create an ideal environment for plants to thrive. With a simple light sensor and the AHT20 sensor, we will be able to create exactly that. These sensors enable us to sense and react to two critical factors that influence plant growth - light and climate. We will control external factors such as light, and airflow using RGB LEDs and DC motor fan.

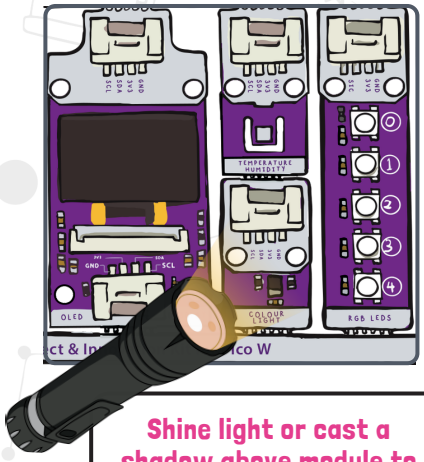
Introduction to Light Sensor

Aside from using the OPT4048 sensor for colour sensing, there's still one feature left that we have yet to explore, and that is none other than the commonly measured, brightness! In this activity, we will learn how to measure different levels of ambient light, allowing you to create projects that respond to changes in lighting conditions.

How Does This Activity Work?

- **Libraries:** board, time, digitalio, adafruit_opt4048, busio.
- **Light Sensor I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Input:**
  - Shine light to the light sensor to increase the brightness level.
  - Block light from entering the light sensor to decrease the brightness level.
- **Output:** Print brightness percentile in shell console.





Shine light or cast a shadow above module to test the light sensor.

The simplest test you can perform is by simply casting various shadow intensities above the light sensor. Use the serial plotter in Thonny to help better visualize the brightness data.

Code

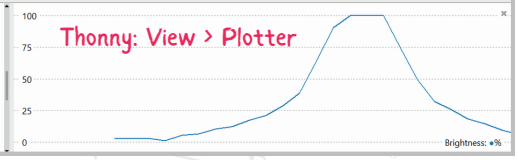
File Edit View Run Tools Help

LIGHT_SENSOR_MODULE.py

```
1  import time, board, busio
2  from adafruit_opt4048 import OPT4048, Mode
3
4  i2c = busio.I2C(board.GP5, board.GP4)
5  sensor = OPT4048(i2c)
6  sensor.mode = Mode.CONTINUOUS
7
8  MAX_LUX = 1000
9
10 while True:
11     R, G, B, LUX = sensor.rgb
12     brightness_percent = min((LUX / MAX_LUX) * 100, 100)
13     print(f"Brightness: {brightness_percent:.1f}%")
14     time.sleep(1)
```

Shell

```
Brightness: 100.0%
Brightness: 74.4%
Brightness: 49.5%
Brightness: 32.2%
Brightness: 25.7%
Brightness: 18.3%
Brightness: 14.5%
Brightness: 9.4%
Brightness: 6.3%
```



Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Initialize Hardware Components

```
4  i2c = busio.I2C(board.GP5, board.GP4)
5  sensor = OPT4048(i2c)
6  sensor.mode = Mode.CONTINUOUS
```

Line 4 - 5: Creates the I2C bus using pins GP5 (SCL) and GP4 (SDA) and initializes the OPT4048 sensor connected to that bus.

Line 6: Sets the sensor to continuous mode, meaning it will keep measuring light automatically without needing to trigger each measurement manually.

Enter a Continuous Loop

```
8  MAX_LUX = 1000
9
10 while True:
11     R, G, B, LUX = sensor.rgb
12     brightness_percent = min((LUX / MAX_LUX) * 100, 100)
13     print(f"Brightness: {brightness_percent:.1f}%")
14     time.sleep(1)
```

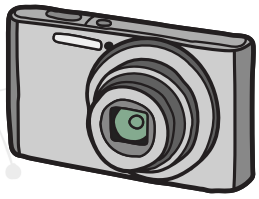
Line 8: MAX_LUX of 1000 is used as the maximum expected brightness from the sensor in order to scale the reading into percentage.
Line 11: sensor.rgb reads the current Red, Green, Blue, and Lux values from the sensor. In this case, only LUX is used to measure brightness.
Line 12: (LUX / MAX_LUX) * 100 converts the brightness to a percentage scale.

THE MORE YOU KNOW

We have compiled a list of unique applications for light sensors that you might find intriguing!

Digital Camera:

- Digital cameras often have light sensors that automatically adjust the appropriate exposure for a photo. These sensors can quickly measure the ambient brightness and adjust the camera settings to capture the best possible image.



Bill Validator:

- Vending machines, ticketing machines, and bill validators often employ light sensors to detect the authenticity of banknotes by analyzing their optical properties and security features.

Museum Display:

- Museums use light sensors to control the illumination of valuable artifacts to ensure the light levels are carefully regulated to prevent damage to the exhibits.

Solar Panels:

- Solar panels use light sensors to track the sun's movement, ensuring that they are always pointed at the sun to maximize energy generation. This process is called solar tracking.

Activity with AI

Prompt Framework: Light Sensor

Task: Use the light sensor to automatically turn the RGB LED ON in the dark and OFF in bright environment.

Step 1: Teach the AI

Copy and paste your working code from **Page 153** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

RGB LED is connected to GP14 with 5 LEDs.

Here is my current code:
<PASTE YOUR CODE FROM PAGE 153>"

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

Prompt #2:
"If it's dark (below 50% brightness), turn all LEDs ON. If it's bright (above the 50% brightness), turn all LEDs OFF. Use white colour for the LEDs"

Step 3: Check the Answer

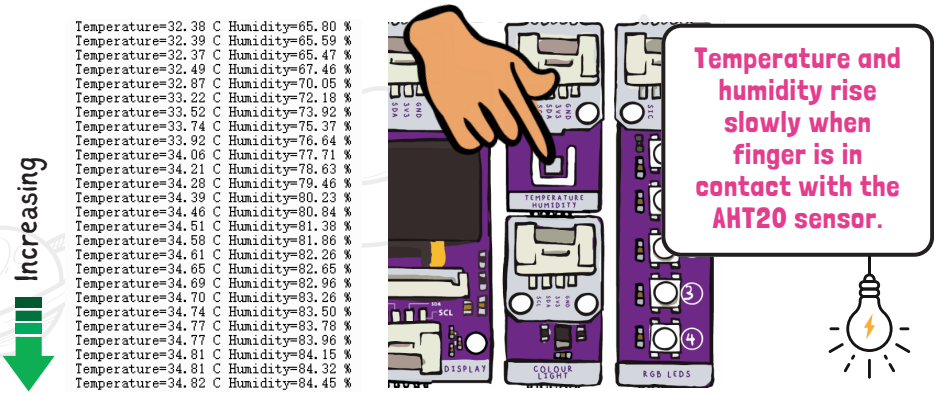
Copy and paste the code in Thonny, then run the code. Check that when cover the sensor, all the LEDs should turn ON. When shining a light on the sensor, all the LEDs should be turned OFF.

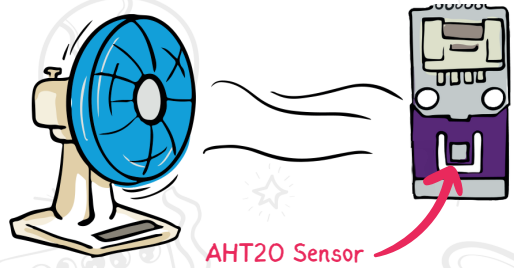
Introduction to Temperature & Humidity Sensor (AHT20)

The AHT20 sensor is a powerful component that provides accurate measurements for both humidity and temperature. Imagine having the ability to monitor and respond to changes in your project's environment, whether its a home automation system, a weather station, or a smart gardening project.

How Does This Activity Work?

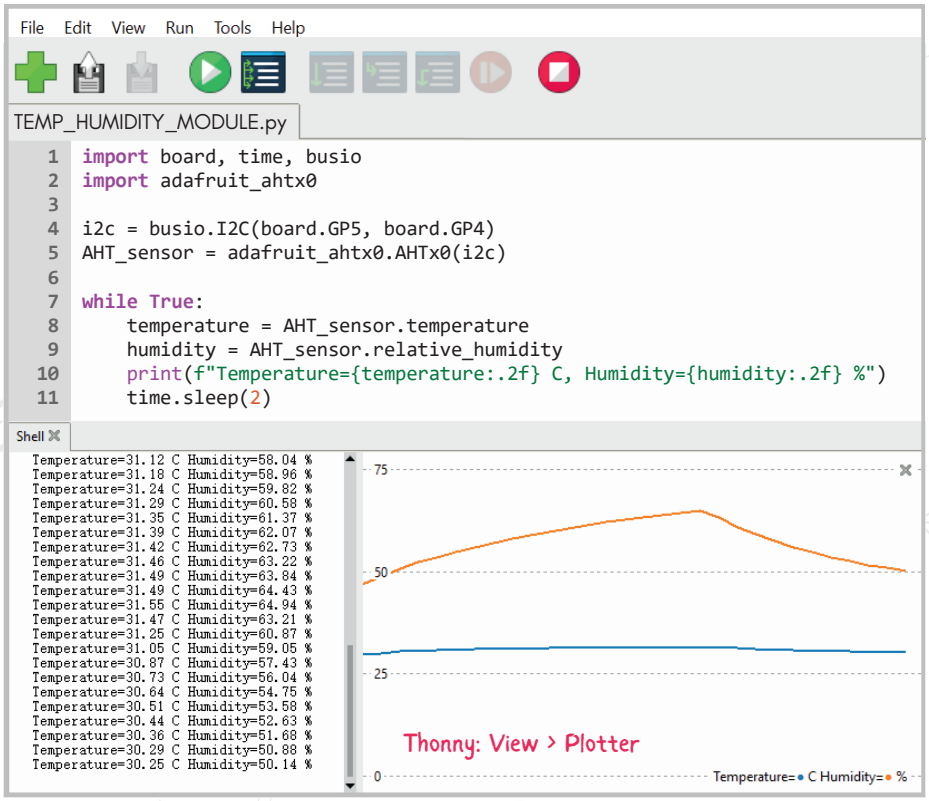
- **Libraries:** board, time, busio, adafruit_ahtx0.
- **AHT20 I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Output:**
  - Print temperature in Celsius and relative humidity percentile in the shell console.
  - Enabling **view > plotter** in Thonny IDE will allow you to visualize the temperature and humidity data in the same graph, making graph analysis easier.





The temperature drops when the sensor is exposed to cool air. The level of humidity is dependent on the amount of moisture present in the air. Blowing air from your mouth may result in a rise in humidity and an increase in temperature.

Code



Click the Green Button to run the code and Red Button to stop.

What the Code Does </>

Import Necessary Libraries

```
1 import board, time, busio
2 import adafruit_ahtx0
```

Line 2: Provides functionality to read temperature and humidity data from the AHT20 sensor.

Initialize Hardware Components

```
4 i2c = busio.I2C(board.GP5, board.GP4)
5 AHT_sensor = adafruit_ahtx0.AHTx0(i2c)
```

Line 5: Initializes I2C communication with the AHT20 sensor.

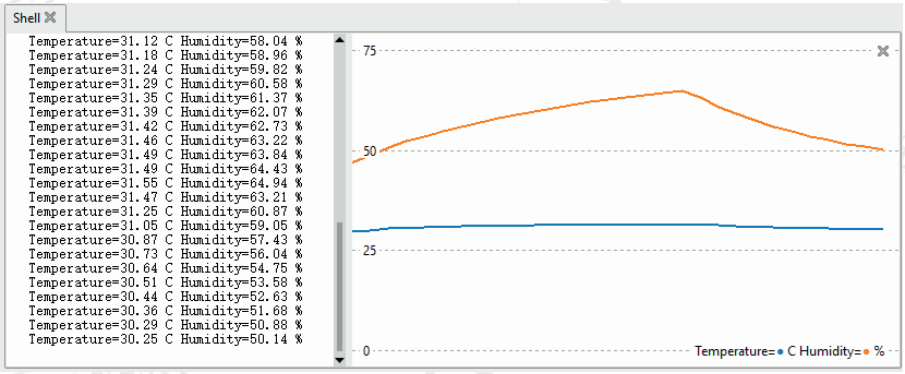
Enter a Continuous Loop

```
7  while True:
8      temperature = AHT_sensor.temperature
9      humidity = AHT_sensor.relative_humidity
10     print(f"Temperature={temperature:.2f} C, Humidity={humidity:.2f} %")
11     time.sleep(2)
```

Line 8: Read the temperature data from the AHT sensor and store it in the **temperature** variable.

Line 9: Read the relative humidity data from the AHT sensor and store it in the **humidity** variable.

Line 10: This line prints the temperature and humidity values in a formatted string. The **":.2f"** inside the f-string is used to format the floating-point numbers to two decimal places.



Printing both temperature and humidity in the same line of code will allow the plotter to plot both values in the same graph.

Activity with AI

Prompt Framework: Temp & Humid Sensor

Task: Use the temperature and humidity sensor to display live readings on the OLED display.

Step 1: Teach the AI

Copy and paste your working code from **Page 157** so the AI knows the setup of your hardware. Teach the AI what it doesn't know yet.

Copy Prompt

Prompt #1:
"You are my CircuitPython coding helper for EDU PICO with Raspberry Pi Pico 2 W. Keep my pins & libraries the same. Show complete code in python block. Add short comments.

Here is my current Temp & Humidity code:
<PASTE YOUR CODE FROM PAGE 157>

Here is the code for the OLED display:
<PASTE YOUR CODE FROM PAGE 48>"

Step 2: Ask the AI

Tell the AI what you want. Describe the outcome as specific as possible.

Copy Prompt

Prompt #2:
"Display the current temperature and humidity on the OLED display. Update the readings every 2 seconds. Show labels like "Temp:" and "Humidity:" clearly on the display. Keep the layout easy to read."

Step 3: Check the Answer

Copy and paste the code in Thonny, then run the code. Check that the OLED display should show both Temperature (°C) and Humidity (%) and continuously update the values every 2 seconds.

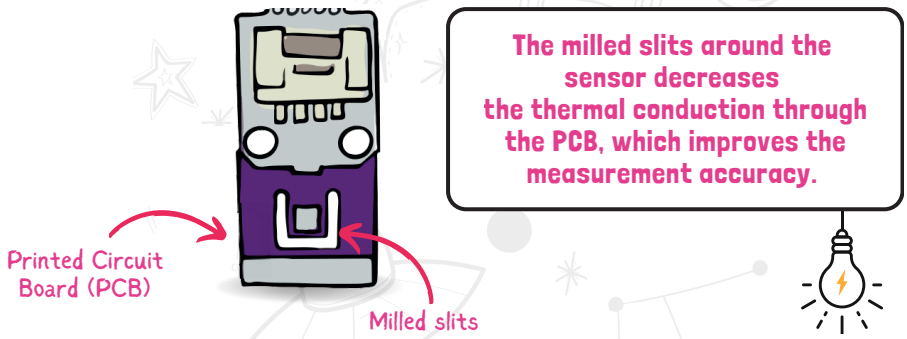
THE MORE YOU KNOW

Understanding the environment is crucial in many electronic projects. The AHT20 sensor allows you to gather real-time data on humidity and temperature, enabling you to make informed decisions in your coding and design processes.

However, to achieve an accurate measurement, we must understand how the sensor module consistently enables good ventilation to prevent the sensor from being affected by the heat radiated from the components nearby.

Milled Slits on Printed Circuit Board (PCB)

The milled slits or empty spaces around the sensor decreases thermal conduction through the printed circuit board (PCB). This is because the slits act as a thermal barrier, reducing the amount of heat that can be conducted through the PCB.



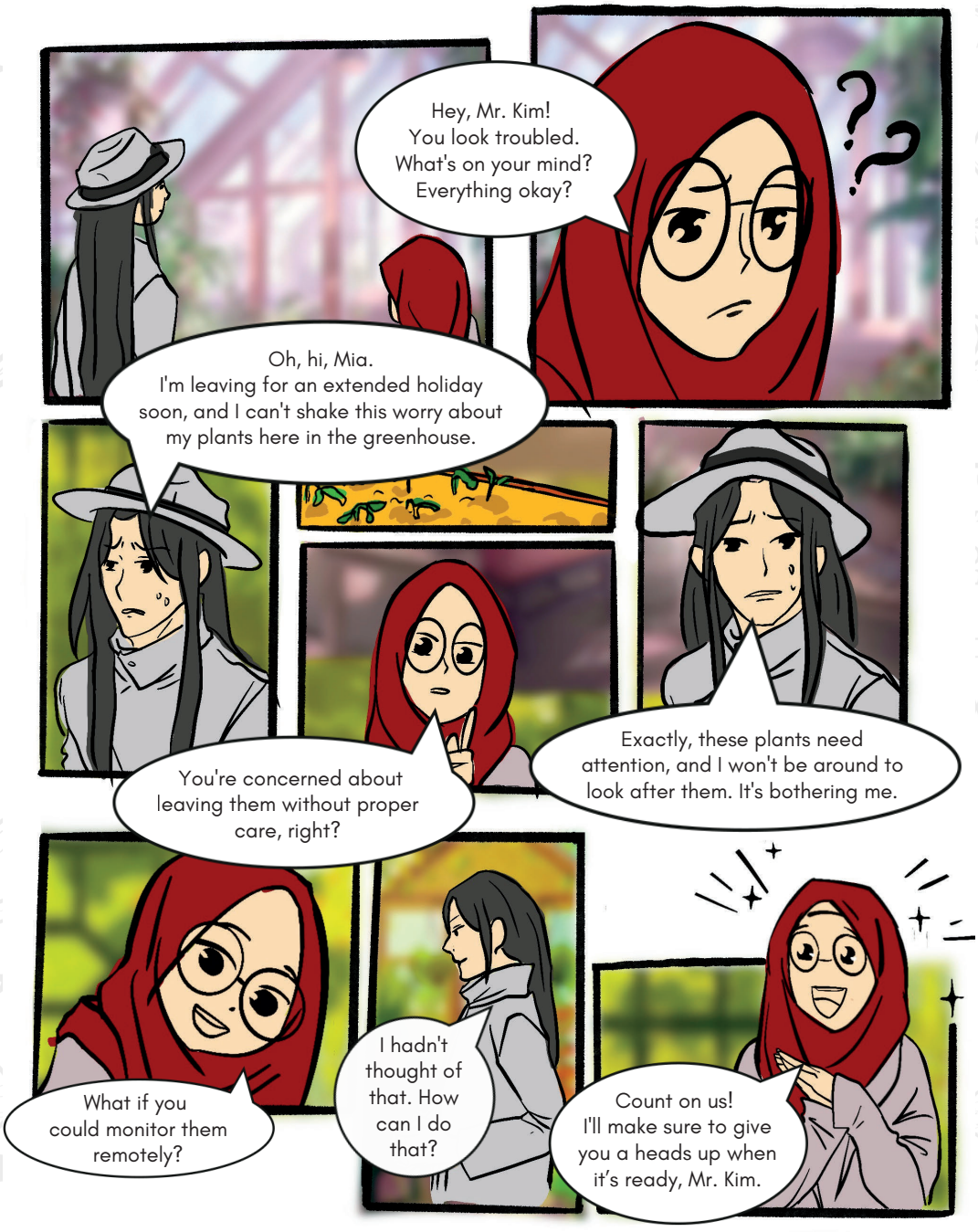
When the measurement frequency is too high, the temperature of the sensor module will heat up, which may affect the measurement accuracy. To keep the temperature from rising, it is recommended to include a 2 second interval for the data collection cycle.

2 second interval

```
while True:
    temperature = AHT_sensor.temperature
    humidity = AHT_sensor.relative_humidity
    print(f"Temperature={temperature:.2f} C, Humidity={humidity:.2f} %")
    time.sleep(2)
```

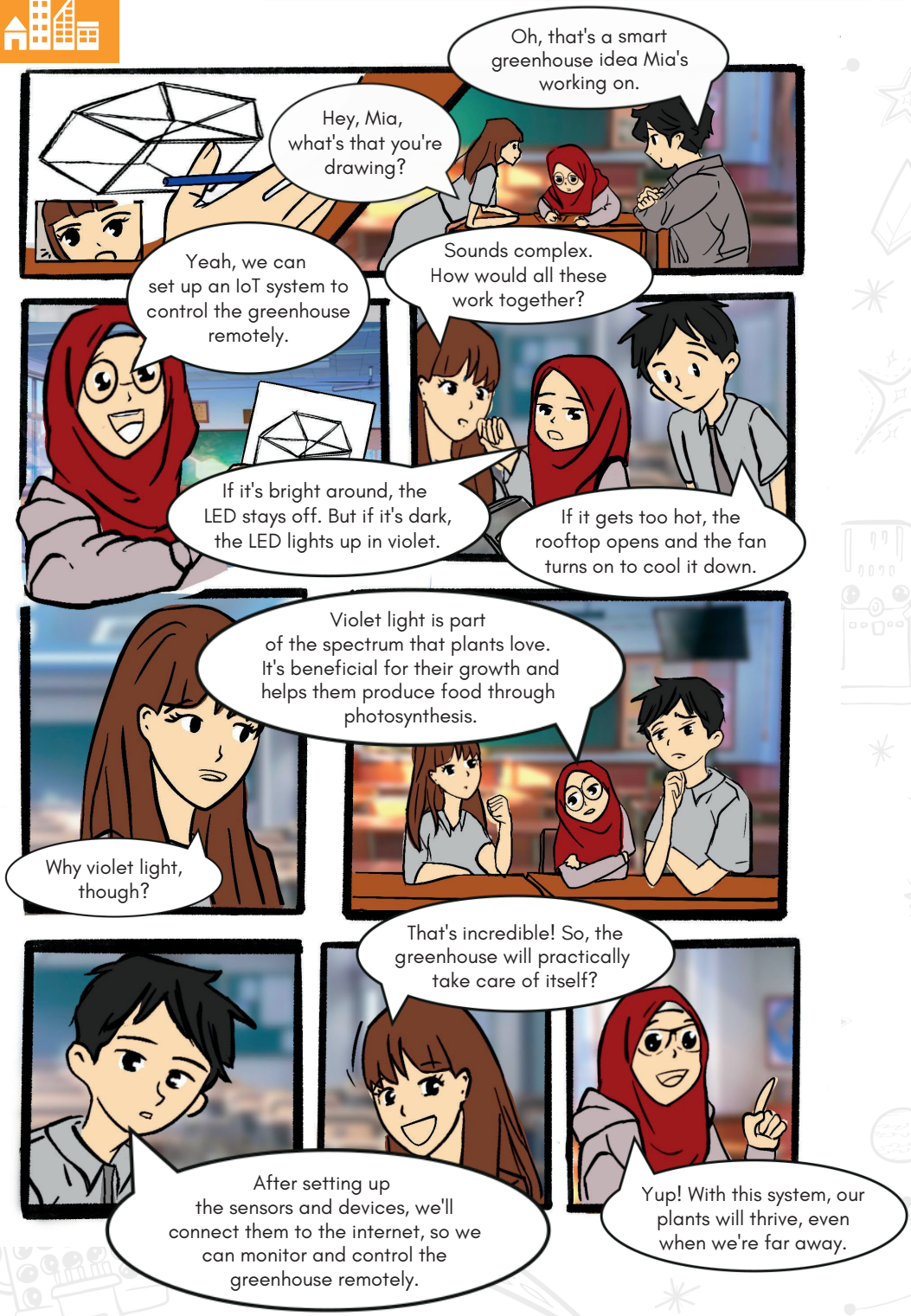
PROLOGUE: CLIMATE CONTROL GREENHOUSE

11 SUSTAINABLE CITIES AND COMMUNITIES



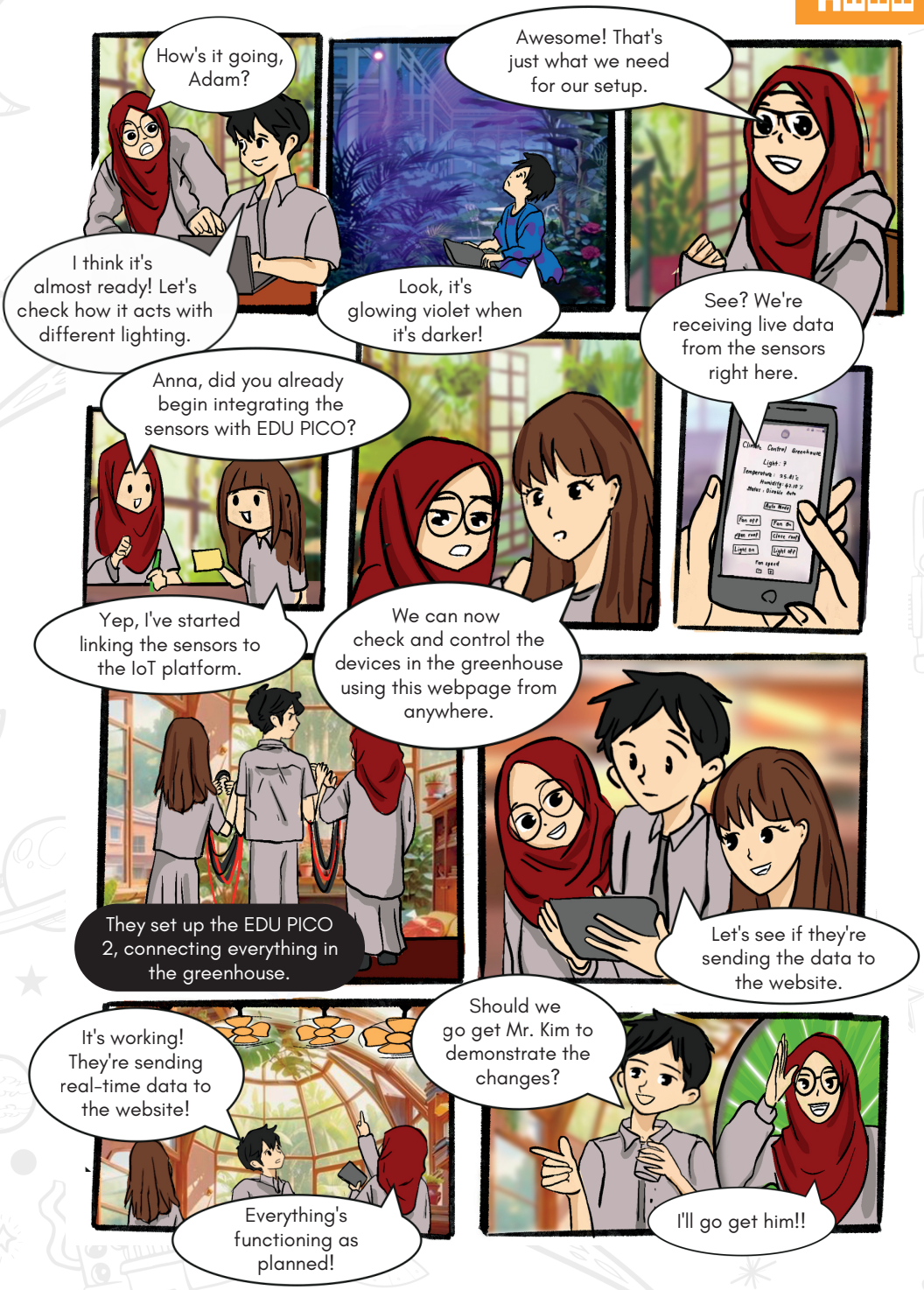
11 SUSTAINABLE CITIES AND COMMUNITIES

PROLOGUE: CLIMATE CONTROL GREENHOUSE



PROLOGUE: CLIMATE CONTROL GREENHOUSE

11 SUSTAINABLE CITIES AND COMMUNITIES



11 SUSTAINABLE CITIES AND COMMUNITIES

PROLOGUE: CLIMATE CONTROL GREENHOUSE

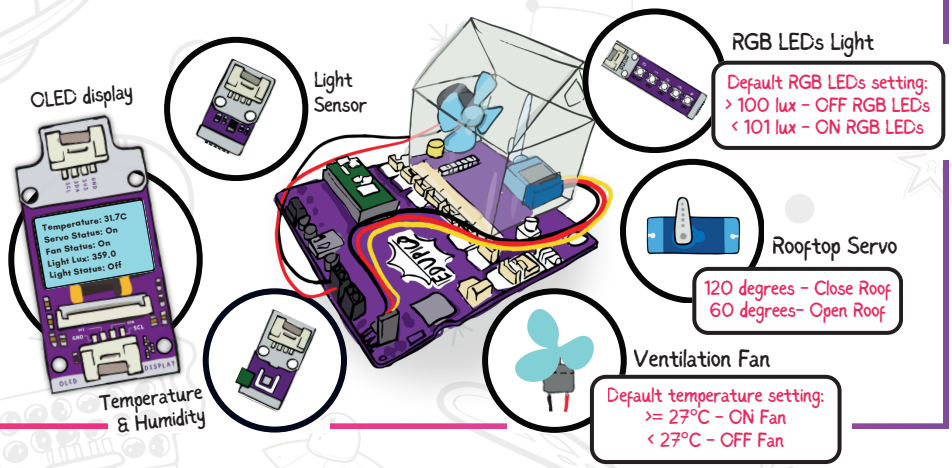


Climate Control Greenhouse

In this project, you will learn how to construct a control system that regulates the humidity and temperature of a closed greenhouse environment. To achieve that, you are required to integrate AHT20 temperature humidity sensor and OPT4048 light sensor as input devices; RGB LEDs (for light), servo motor (for rooftop control), DC motor (for the fan), and OLED display for printing the greenhouse status.

How Does This Activity Work?

- **Libraries:** board, busio, time, adafruit_ssd1306, neopixel, pwmio, adafruit_ahtx0, adafruit_motor, adafruit_opt4048.
- **I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
  - OPT4048 light sensor, AHT20 temperature humidity sensor, and SSD1306 OLED share similar I2C pins configuration.
- **Input:**
  - Shine a light on the OPT4048 light sensor to simulate surrounding brightness in the greenhouse.
  - Test the AHT20 temperature humidity sensor by placing your finger on the sensor module. The temperature should rise gradually.
- **Output:**
  - If the surroundings are bright, the RGB LEDs remain off; whereas if the surroundings are dark, the RGB LEDs will light up in violet.
  - If the surroundings are hot, the servo will activate to **60** degrees (opening the rooftop), and the DC motor fan will activate to ventilate the area to cool down the environment.
  - The diagram below shows the components used in this project.



Code



PROJECT_CLIMATE_CONTROL_GREENHOUSE.py

```
import board, busio, time, adafruit_ssd1306, neopixel
import adafruit_ahtx0
from pwmio import PWMOut
from adafruit_motor import servo, motor
from adafruit_opt4048 import OPT4048, Mode

i2c = busio.I2C(board.GP5, board.GP4)
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
oled.invert(True)

aht20 = adafruit_ahtx0.AHTx0(i2c)
opt = OPT4048(i2c)
opt.mode = Mode.CONTINUOUS

pixels = neopixel.NeoPixel(board.GP14, 5, brightness=0.2)

PWM_Servo = PWMOut(board.GP6, frequency=50)
servo = servo.Servo(PWM_Servo, min_pulse=500, max_pulse=2500)

PWM_M1A = PWMOut(board.GP10, frequency=10000)
PWM_M1B = PWMOut(board.GP11, frequency=10000)
motor = motor.DCMotor(PWM_M1A, PWM_M1B)

def light_on():
    pixels.fill((255, 0, 255))
    oled.text("Light Status: On", 5, 52, 1)

def light_off():
    pixels.fill((0, 0, 0))
    oled.text("Light Status: Off", 5, 52, 1)

def servo_on():
    servo.angle = 60
    oled.text("Servo Status: On", 5, 15, 1)

def servo_off():
    servo.angle = 120
    oled.text("Servo Status: Off", 5, 15, 1)
```

```
def temp_control(temp_threshold):
    temperature = aht20.temperature
    oled.text("Temperature: {:.1f} C".format(temperature), 5, 3, 1)
    if temperature >= temp_threshold:
        servo_on()
        motor.throttle = 0.25
        oled.text("Fan Status: On", 5, 27, 1)
    else:
        servo_off()
        motor.throttle = 0
        oled.text("Fan Status: Off", 5, 27, 1)

def light_control(light_threshold):
    R, G, B, lux = opt.rgb
    oled.text("Light Lux: {:.1f}".format(lux), 5, 39, 1)
    if light_lux < light_threshold:
        light_on()
    else:
        light_off()

try:
    while True:
        oled.fill(0)
        temp_control(temp_threshold=27)
        light_control(light_threshold=100)
        oled.show()
        time.sleep(2)

finally:
    oled.fill(1)
    oled.show()
    print("deinit I2C")
    i2c.deinit()
```

Click the Green Button  to run the code and Red Button  to stop.

What the Code Does </>

Import Necessary Libraries

Libraries

```
1 import board, busio, time, adafruit_ssd1306, neopixel
2 import adafruit_ahtx0
3 from pwmio import PWMOut
4 from adafruit_motor import servo, motor
5 from adafruit_opt4048 import OPT4048, Mode
```

These lines import the necessary libraries and modules for working with various hardware components which include the OLED display, temperature and humidity sensor (AHT20), colour sensor (OPT4048), servo motor, DC motor, and RGB LEDs.

Initialize Hardware Components

Hardware Initialization

```
7  i2c = busio.I2C(board.GP5, board.GP4)
8  oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
9  oled.invert(True)
10
11 aht20 = adafruit_ahtx0.AHTx0(i2c)
12 opt = OPT4048(i2c)
13 opt.mode = Mode.CONTINUOUS
14
15 pixels = neopixel.NeoPixel(board.GP14, 5, brightness=0.2)
16
17 PWM_Servo = PWMOut(board.GP6, frequency=50)
18 servo = servo.Servo(PWM_Servo, min_pulse=500, max_pulse=2500)
19
20 PWM_M1A = PWMOut(board.GP10, frequency=10000)
21 PWM_M1B = PWMOut(board.GP11, frequency=10000)
22 motor = motor.DCMotor(PWM_M1A, PWM_M1B)
```

Line 7: Initializes I2C communication with ***GP5*** for SCL and ***GP4*** for SDA.

Line 8 - 12: Configure the OLED, temperature, humidity, and colour sensor using the same I2C bus and pins. **Line 9** inverts the OLED display, changing the background to white and the text colour to black.

Note: When using I2C, each device has a unique address on the bus, allowing multiple devices to communicate with the Raspberry Pi Pico 2 W.

Line 15 - 22: Initializes RGB LEDs (***GP14***), DC motor (M1A = ***GP10***, M1B = ***GP11***) and servo motor (***GP6***).

Define Custom Functions

Configuring Functions for Controlling Light, and Servo

```
24 def light_on():
25     pixels.fill([255, 0, 255])
26     oled.text("Light Status: On", 5, 52, 1)
27
28 def light_off():
29     pixels.fill([0, 0, 0])
30     oled.text("Light Status: Off", 5, 52, 1)
31
32 def servo_on():
33     servo.angle = 60
34     oled.text("Servo Status: On", 5, 15, 1)
35
36 def servo_off():
37     servo.angle = 120
38     oled.text("Servo Status: Off", 5, 15, 1)
```

Line 24 - 30: **light_on()** fills the RGB LEDs with violet colour (preferable colour for plants to perform photosynthesis), and **light_off()** to turn off the RGB LEDs while updating the OLED display with the corresponding status messages.

Line 32 - 38: **servo_on()** function rotates the servo to **60** degrees, ideally to open the rooftop of the greenhouse, whereas **servo_off()** will rotate the servo back to its original state at **120** degrees, closing the greenhouse rooftop.

Temperature Control Function

```
40 def temp_control(temp_threshold):
41     temperature = aht20.temperature
42     oled.text("Temperature: {:.1f} C".format(temperature), 5, 3, 1)
43     if temperature >= temp_threshold:
44         servo_on()
45         motor.throttle = 0.25
46         oled.text("Fan Status: On", 5, 27, 1)
47     else:
48         servo_off()
49         motor.throttle = 0
50         oled.text("Fan Status: Off", 5, 27, 1)
```

Line 40 - 42: The **temp_control** function monitors the temperature from the AHT20 sensor while displaying the data on the OLED display.

Line 43 - 50: Activate the DC motor fan and servo motor when the temperature is above the **temp_threshold** value. The temperature threshold value is set at **Line 63**.

Light Control Function

```
52 def light_control(light_threshold):
53     R, G, B, lux = opt.rgb
54     oled.text("Light Lux: {:.1f}".format(lux), 5, 39, 1)
55     if lux < light_threshold:
56         light_on()
57     else:
58         light_off()
```

Lux is a measure of illuminance, the total amount of light that falls on a surface.



Line 53: The function reads the data from the OPT4048 light sensor and returns R (Red), G (Green), B (Blue) and lux (Total brightness) values.

Line 54: This line updates the OLED display with the calculated **light_lux** value. **.format(light_lux)** allows "{:.1f}" to be replaced with the actual value of **lux**.

Line 55 - 58: Calls **light_on()** function when the calculated lux value is below the preset threshold, and **light_off()** function if the lux value is above the threshold.

Enter a Continuous Loop

Main Loop

```
60 try:
61     while True:
62         oled.fill(0)
63         temp_control(temp_threshold=27)
64         light_control(light_threshold=100)
65         oled.show()
66         time.sleep(2)
67
68 finally:
69     oled.fill(1)
70     oled.show()
71     print("deinit I2C")
72     i2c.deinit()
```

Adjust this temperature value to control the output sensitivity.

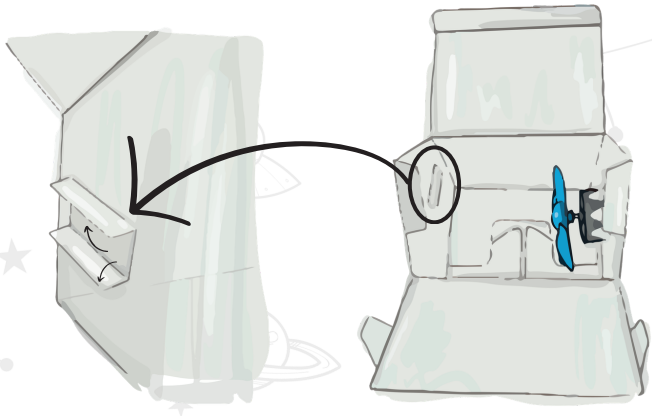
Line 60 - 66: Continuously calls the **temp_control** and **light_control** function while passing a temperature threshold of **27** degrees Celsius and light threshold of **100** lux respectively. These functions update the OLED display every **2** seconds and control the fan and servo motor depending on the temperature and lux data measured.

Line 68 - 72: The **finally** block ensures that cleanup operations are performed, regardless of whether an exception occurred or not during the main program execution.

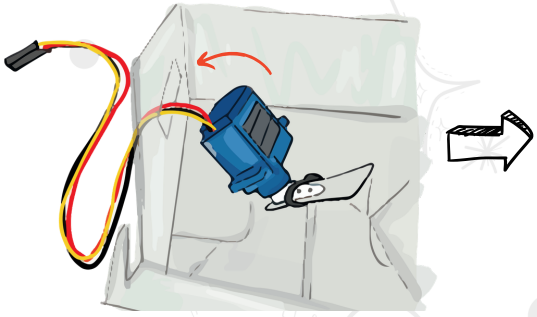
WHAT'S NEXT?

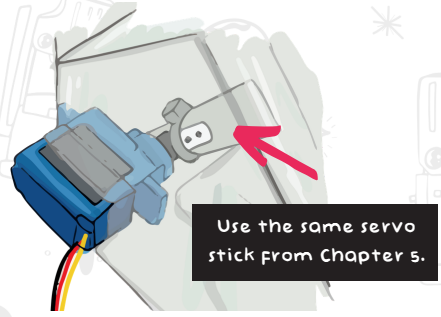
HOUSE ACCESSORY - SERVO ROOF

Step 1:
Locate the servo motor attachment flaps and flip it outwards as shown.



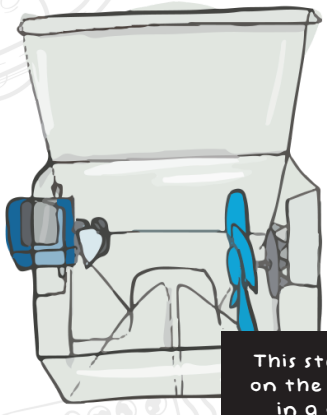
Step 2:
Attach the servo motor from inside out with the cable going through the box first.





Use the same servo stick from Chapter 5.

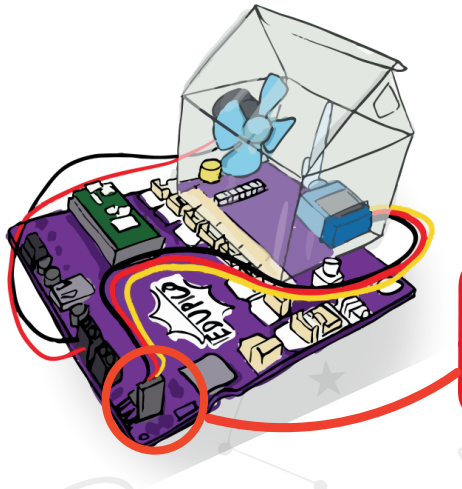
Step 3:
Close the front side of the roof back to it's original position.



Step 4:
Give the back roof a gentle push downwards and inwards to the box.

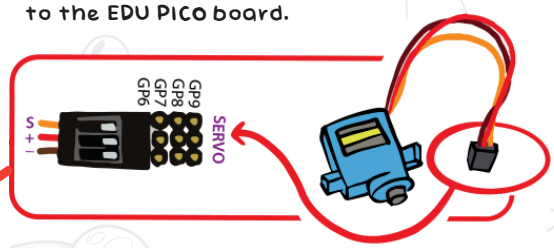


This step will strengthen the spring effect on the plastic roof, allowing it to maintain in a closed position when not stressed.



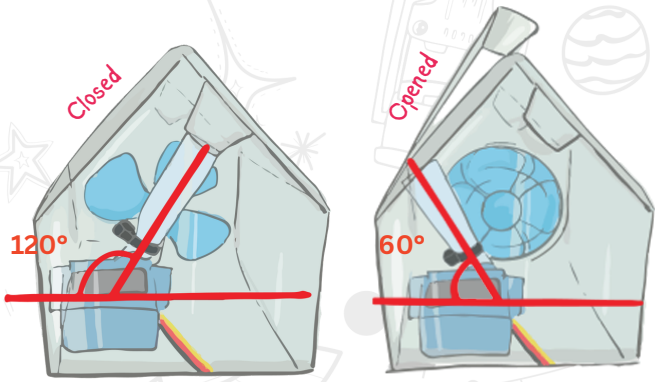
Step 5:

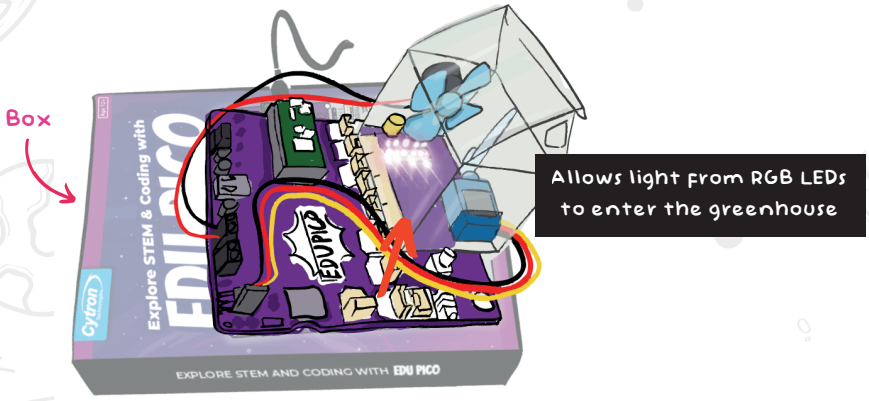
Connect the servo motor & DC motor to the EDU PICO board.



Step 6:

Test the servo motor horn position to open and close the roof. Open roof = 60°, close roof = 120°.





Final Placement

Bonus: Wireless Network (AP-Mode)

By the end of this activity, you'll be able to setup your own WiFi AP network and enable EDU PICO to communicate wirelessly with other devices. Let's start with the basics - an access point, or "AP" serves as a central hub for wireless communication. It acts as a bridge between the EDU PICO and other WiFi-enabled devices, like smartphones and laptops, to enable data sharing.

This setup is incredibly helpful for various applications, including IoT (Internet of Things) projects and remote-control systems. This bonus section will guide you through the configuration for your EDU PICO to work as an access point. This means that you can connect to it, share data, and control it remotely, all over WiFi.

How Does This Activity Work?

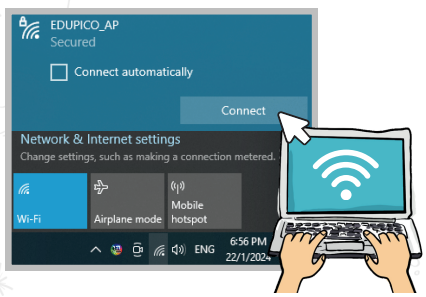
- **Libraries:** board, digitalio, wifi, socketpool, adafruit_httpserver**.**
- **USB Relay Configuration:**
  - Assign to ***GP22*** and connect USB LED light stick to the USB port.

Connecting with Computer

- **Output:**
  - The program starts the server and prints "Starting server..." followed by "Listening on http://[IP_ADDRESS]" where [IP_ADDRESS] is the IP address of the WiFi access point.

```
Shell
>>> %Run -c $EDITOR_CONTENT
 Starting server...
 Listening on http://192.168.4.1
```

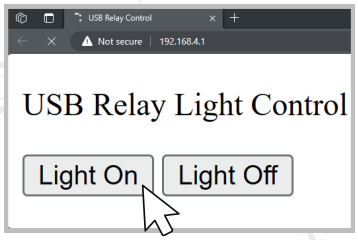
  - You can access the control interface by connecting to the WiFi access point (AP) (SSID: "EDUPICO_AP", Password: "12345678") using a device like a smartphone or a computer.

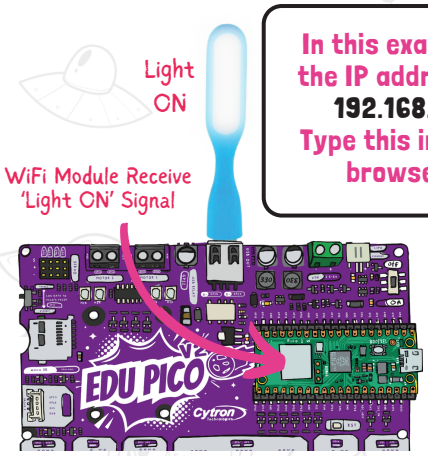


  - When you open a web browser with the server's IP address, you will see a webpage with "Light On" and "Light Off" buttons.

- Once you're on the webpage, clicking these buttons will send a signal to the server, and the relay state will change accordingly. Messages like "Light ON" and "Light OFF" will be printed on the shell console when the buttons are clicked.





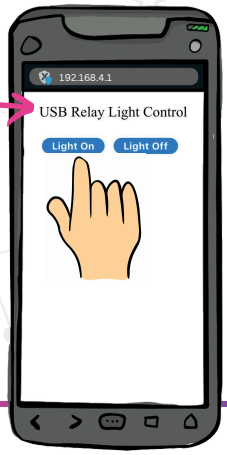


In this example, the IP address is 192.168.4.1 Type this in your browser.

Connecting with Smartphone

- To connect it to your smartphone, simply connect the device to the same WiFi and key in the same IP address to your browser.





Use the same IP address here to control the EDU PICO through your smartphone device!

Code



BONUS_AP_RELAY.py

```
import board, digitalio, socketpool, wifi
from adafruit_httpserver import Request, Response, Server, POST

AP_SSID = "EDUPICO_AP"
AP_PASSWORD = "12345678"

print("Creating access point...")
wifi.radio.start_ap(ssid=AP_SSID, password=AP_PASSWORD)
print(f"Created AP WiFi ID = {AP_SSID}")

ip = str(wifi.radio.ipv4_address_ap)
port = 5000

pool = socketpool.SocketPool(wifi.radio)
server = Server(pool, "/static", debug=False)

relay = digitalio.DigitalInOut(board.GP22)
relay.direction = digitalio.Direction.OUTPUT

@server.route("/")
def base(request: Request):
    html = f"""
        <html>
        <head>
            <meta http-equiv="refresh" content="5">
            <title>USB Relay Control</title>
        </head>
        <body style="font-family:Arial; text-align:center;
        margin-top:300px; font-size:60px;">
            <h2 style="font-size:60px;">USB Relay Light Control</h2>
            <form method="POST">
                <button name="action" value="on" type="submit"
                style="font-size:60px">Light On</button>
                <button name="action" value="off" type="submit"
                style="font-size:60px">Light Off</button>
            </form>
        </body>
        </html>
    """
    return Response(request, html, content_type='text/html')
```

```
@server.route("/", POST)
def control(request: Request):
    raw = request.raw_request.decode("utf-8")
    if "action=on" in raw:
        print("Relay ON")
        relay.value = True
    elif "action=off" in raw:
        print("Relay OFF")
        relay.value = False
    return base(request)

print("Starting HTTP server...")
print(f"Server running at: http://{ip}:{port}/")

server.serve_forever(ip, port=port)
```

Click the Green Button to run the code and Red Button to stop.

What the Code Does

Import Necessary Libraries

Libraries

```
import board, digitalio, socketpool, wifi
from adafruit_httpserver import Request, Response, Server, POST
```

Line 1: Provides WiFi functionality and socket communication. A socket represents one endpoint of a two-way communication link between two programs running on a network.

Line 2: Web server library for handling HTTP requests and responses.

Define Custom Functions

Set Up WiFi AP

```
AP_SSID = "EDUPICO_AP"
AP_PASSWORD = "12345678"
```

Line 4 - 5: These configures the Raspberry Pi Pico 2 W to act as a WiFi Access Point (AP) with a specified SSID "EDUPICO_AP" and password "12345678".

[Note: If more than one EDU PICO is running as WiFi AP in the same area, it is recommended to set up different SSIDs for each board for easier connection]

Set Up WiFi AP & Relay

```
print("Creating access point...")
wifi.radio.start_ap(ssid=AP_SSID, password=AP_PASSWORD)
print(f"Created AP WiFi ID = {AP_SSID}")

ip = str(wifi.radio.ipv4_address_ap)
port = 5000

pool = socketpool.SocketPool(wifi.radio)
server = Server(pool, "/static", debug=False)

relay = digitalio.DigitalInOut(board.GP22)
relay.direction = digitalio.Direction.OUTPUT
```

Line 8 - 9: **wifi.radio.start_ap()** tells the Pico 2 W to create its own Wi-Fi hotspot. It then proceeds to print the IP address at the shell console to indicate that the Pico 2 W has successfully created the WiFi AP.

Line 11: The **wifi.radio.ipv4_address_ap** gives the device's IP address on its own AP network (usually 192.168.4.1).

Line 12: port = 5000 sets the HTTP port number for your web server.

Line 14 - 15: The **SocketPool** manages TCP connections using the Pico's Wi-Fi hardware and the **Server()** creates a HTTP server object.

Line 17 - 18: This creates a DigitalInOut object connected to **GP22** and sets it as an output pin to control the relay module.

Webpage HTML

```
@server.route("/")
def base(request: Request):
    html = f"""
        <html>
        <head>
            <meta http-equiv="refresh" content="5">
            <title>USB Relay Control</title>
        </head>
        <body style="font-family:Arial; text-align:center;
        margin-top:300px; font-size:60px;">
            <h2 style="font-size:60px;">USB Relay Light Control</h2>
            <form method="POST">
                <button name="action" value="on" type="submit"
                style="font-size:60px">Light On</button>
                <button name="action" value="off" type="submit"
                style="font-size:60px">Light Off</button>
            </form>
        </body>
        </html>
    """
    return Response(request, html, content_type='text/html')
```

Line 20 - 39: This function defines the HTML webpage with a title, a paragraph of text, and two buttons. The webpage automatically refreshes every **5** seconds. When any button is clicked, the page will submit a POST request to the server with the values clicked, the page will submit a POST request to the server with the values **light_on** or **light_off**, respectively. This setup allows users to control the light connected to the USB relay by interacting with the buttons on the webpage.

Line 40: Sends the HTML content back as the HTTP response when someone visits the page.

Setting Up the HTTP Server

```
42  @server.route("/", POST)
43  def control(request: Request):
44      raw = request.raw_request.decode("utf-8")
45      if "action=on" in raw:
46          print("Relay ON")
47          relay.value = True
48      elif "action=off" in raw:
49          print("Relay OFF")
50          relay.value = False
51      return base(request)
52
53  print("Starting HTTP server...")
54  print(f"Server running at: http://{ip}:{port}/")
55
56  server.serve_forever(ip, port=port)
```

Line 42: The **server.route** handles POST requests when the buttons are clicked.

Line 43 - 51: The function reads the HTTP request to find out whether the user clicked the **"Light On"** or **"Light Off"** button. If **"on",** it will set the relay.value to True, which will activate the relay. Else if "off", the relay will be deactivated.

Line 53 - 54: Displays the Pico's web address in the shell console.

Line 56: server.serve_forever() keeps the server running indefinitely, listening for web requests on that IP and port.

Guidebook Summary

Chapter 1: Programming with CircuitPython

- [ ] Install Thonny IDE.
- [ ] Program our first CircuitPython program using Thonny IDE.
- [ ] Download program to EDU PICO 2.
- [ ] Save, open, and edit Python .py files in EDU PICO 2.
- [ ] Import CircuitPython libraries.

Chapter 2: Water Drinking Reminder (Button and Buzzer)

- [ ] Use input buttons to interact with Thonny's console.
- [ ] Use a piezo buzzer to produce sound.
- [ ] AI Task #1: Print "A" when button A is pressed and "B" when button B is pressed.
- [ ] AI Task #2: Make the buzzer play a short melody of "Twinkle Twinkle Little Star".
- [ ] Activities: Build a water drinking reminder.
- [ ] Challenge: Program drinking reminder to accept user input for the duration.

Chapter 3: Gesture Reaction Game (OLED and Gesture Sensor)

- [ ] Display text on the OLED display.
- [ ] Use gesture sensor as input.
- [ ] AI Task #1: Sweep the text "Hello EDU PICO" across the screen from left to right.
- [ ] AI Task #2: Use hand gestures to control the text movement on the OLED display.
- [ ] Activities: Build a hearing gesture reaction game with a gesture sensor and buzzer.
- [ ] Challenge: Program a tone testing feature for better gaming experience.

Guidebook Summary

Chapter 4: Colour Detection Game (RGB LEDs and Colour Sensor)

- [ ] Program EDU PICO 2 to light up RGB LEDs in different colours.
- [ ] Read colour data with colour sensor.
- [ ] AI Task #1: Make the RGB LEDs show a running rainbow colour effect.
- [ ] AI Task #2: Use the colour sensor to detect red, green, or blue and show that colour on the RGB LED module.
- [ ] Activities: Build a colour detection game.
- [ ] Challenge: Improve the game by programming a colour hint indicator using a single RGB LEDs.

Chapter 5: Automated Waste Bin (Servo Motor and Proximity Sensor)

- [ ] control a servo motor using pulse width modulation (PWM).
- [ ] read distance value using the PAJ7620 proximity sensor.
- [ ] read data with Thonny's plotter.
- [ ] construct a Trashbot smart bin with card box accessories.
- [ ] AI Task #1: Use Button A to trigger the servo motor to move from 0° to 180°.
- [ ] AI Task #2: Use the proximity sensor to trigger and control the buzzer's beeping interval.
- [ ] Activities: Build an Automated Waste Bin (Trashbot Smart Bin).
- [ ] Challenge: Upgrade Trashbot to include light and sound, making it more interactive.

Chapter 6: Noise Pollution Monitoring System (Potentiometer and Sound Sensor)

- [ ] Program EDU PICO 2 to read analog values from a potentiometer.
- [ ] Measure noise in dB with the PDM sound sensor.
- [ ] Construct a physical noise level meter with a card accessory.
- [ ] AI Task #1: Use the potentiometer to control the brightness of the RGB LED strip.
- [ ] AI Task #2: Use the sound sensor to turn the LED on and off when the sound level exceeds a specific threshold.
- [ ] Activities: Build a room environment noise pollution monitoring system.
- [ ] Challenge: Program a servo motor to serve as a sound level meter.

Guidebook Summary

Chapter 7: Smart Classroom (DC Motor and Relay)

- [ ] Program EDU PICO 2 to control a DC motor – spinning direction and speed control.
- [ ] Turn ON and OFF a USB switch relay.
- [ ] AI Task #1: Use the potentiometer to control the speed and direction of the DC motor.
- [ ] AI Task #2: Use Button A to toggle the relay ON and OFF (like a light switch).
- [ ] Activities: Build a smart classroom with a gesture sensor, USB relay, DC motor, and OLED.

Chapter 8: Climate Control Greenhouse (Light and Humidity Temperature Sensor)

- [ ] Program light sensor to measure ambient brightness.
- [ ] Program AHT20 sensor for humidity and temperature measurement.
- [ ] Program EDU PICO 2's Raspberry Pi Pico 2 W into a WiFi access point
- [ ] AI Task #1: Use the light sensor to automatically turn the RGB LED ON and OFF.
- [ ] AI Task #2: Display temperature and humidity live readings on the OLED display.
- [ ] Activities: Build a climate control greenhouse using temperature and humidity sensor, light sensor, DC and servo motor, RGB LEDs, and OLED.
- [ ] Bonus: Control the USB relay through a webpage using the Raspberry Pi Pico 2 W acting as a WiFi Access Point (AP).

CONGRATULATIONS!

As you reach the end of this guidebook, the EDU PICO 2 team would like to extend our warmest congratulations to you. The journey you've undertaken is not just about reaching the final chapter but about the skills you've acquired, the challenges you've conquered, and the projects you're now able to create.

May this achievement be a steppingstone to a world filled with endless possibilities. As you continue to pursue your passion for learning, experimenting, and innovating, remember that the journey does not end here, with the skills from EDU PICO 2, you can now keep on exploring and building cool projects!

Once again, congratulations on completing this chapter of your learning journey and we can't wait to see what you will be building with the EDU PICO 2!

Best Wishes,
EDU PICO team

Check out the EDU PICO resource hub for more projects here.

edupico-hub.cytron.io

Share with us your projects here
education@cytron.io



PM-BOK-EDU-PICO2 R1221225

LEARN, BUILD & EXPLORE WITH EDU PICO 2