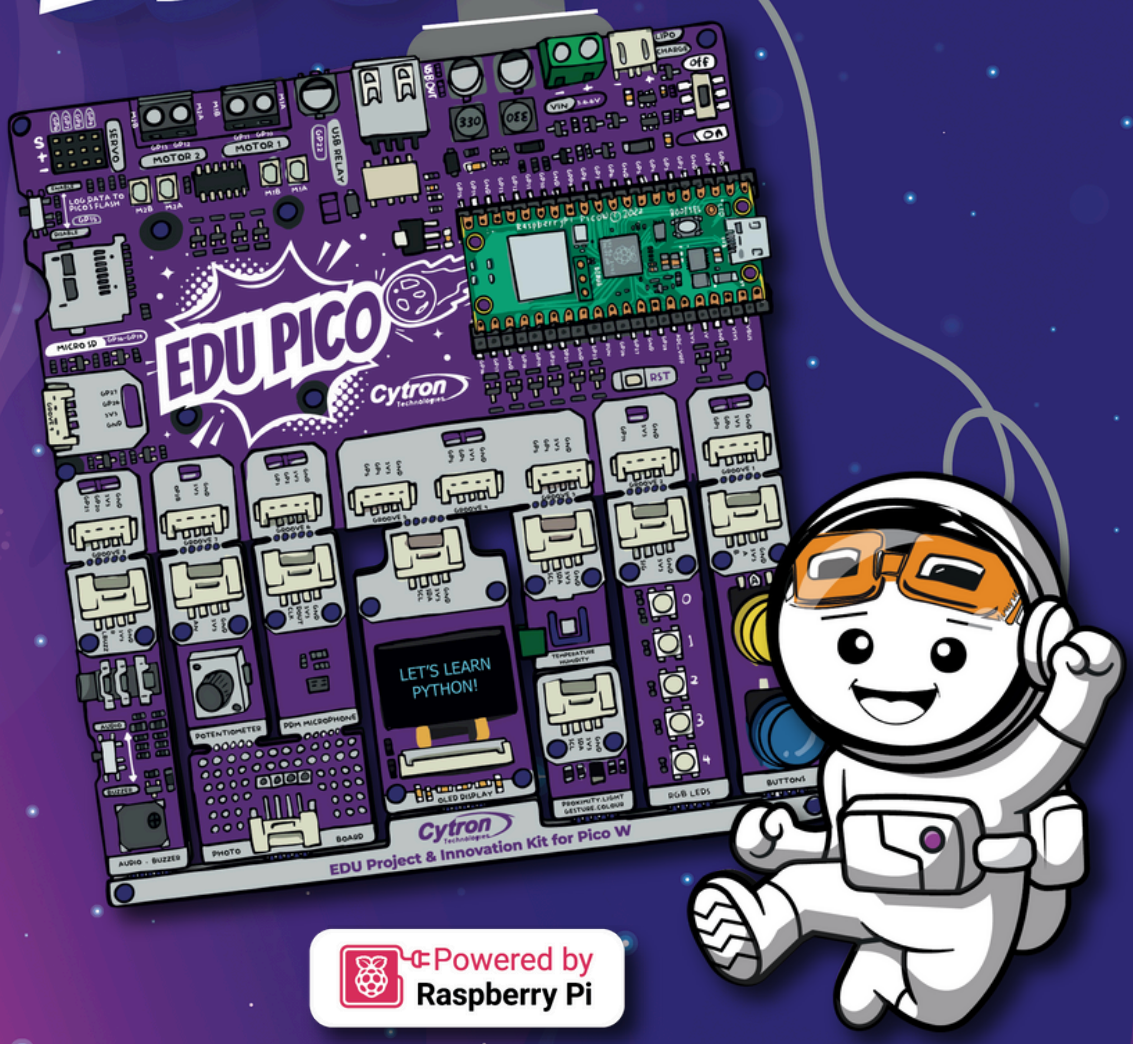
Cytron Technologies

Age 12+

Explore STEM & Coding with EDU PICO



Powered by Raspberry Pi

Author's Note

It is with great excitement to introduce this book, "Explore STEM and Coding with EDU PICO." This comprehensive guide is designed to accompany the EDU PICO, a product powered by the Raspberry Pi Pico W, and brought to you by Cytron Technologies.

A special acknowledgement goes to SC Lim, our Project Manager, for steering the ship and ensuring the successful development of both the EDU PICO kit and this accompanying guide. Salam, the Designer of EDU PICO, for lending his expertise in shaping the technical foundations of EDU PICO, making it a powerful tool for STEM education. Suhana, our Illustrator, for adding a creative touch to this book. Her artistic skills turned complicated ideas into easy-to-understand visuals, making the learning experience more enjoyable.

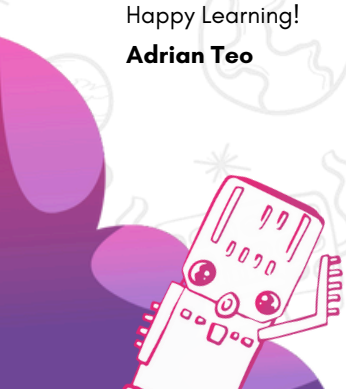
I sincerely thank the diligent reviewers—Hairil, Cheryl, Anna, Alhamed, Poomipat, Iffah, and ET Tan—for their meticulous assessments and valuable feedback. A heartfelt shoutout to our Trainees—Justin, Hao Khee, Anas, and Azeem—for their dedicated assistance. Their collective contributions have played an instrumental role in fine-tuning and enhancing the content of this book.

I want to express my deepest thanks to the entire team for their collaborative efforts, including everyone at Cytron Technologies, whose commitment to educational excellence has made this project possible.

Finally, this book is dedicated to the students and educators, who will embark on this learning journey. May EDU PICO be a gateway to discovering the wonders of programming and electronics, sparking a lifelong passion for learning.

I look forward to seeing the awesome projects you'll create with the knowledge you gain from these pages and EDU PICO!

Happy Learning!
Adrian Teo



Explore STEM and Coding with EDU PICO Project & Innovation Kit

Second Printing, March 2024

Published by Cytron Technologies

Copyright © 2024 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 support@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

Chapter Summary

Discover the EDU PICO, an all-in-one Raspberry Pi Pico W learning kit, specially designed for beginners to venture into text-based Python programming. Unfolding across 8 chapters, 7 projects, and over 10 stimulating challenges, this kit delivers an enriching learning journey.

Design thinking is instilled through comic-style narratives that ingeniously link each project to sustainable development goals. Let's embark on a journey where programming proficiency merges with creative problem-solving and innovation!

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.
- save, open, and edit Python .py files in EDU PICO.

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.
- create and use variables.
- use while loop.
- use a conditional if statement.

Introduction: Button and Buzzer.
Activities: Build a water drinking reminder (Hydration Companion).
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.
- program using If.. elif conditions.
- setup dictionaries and for loops.
- create and use functions.

Introduction: OLED and Gesture Sensor.
Activities: Build a hearing gesture reaction game with a gesture sensor and buzzer (Do-Re-Mi-Fa Arcade).
Challenge 1: Improve game test tone by displaying notes on OLED.
Challenge 2: Program a hearing-gesture directory for better gaming experience.

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

In this chapter, we learn to:

- program EDU PICO to light up RGB LEDs in different colours.
- read colour data with colour sensor.
- setup lists to store multiple items.

Introduction: RGB LEDs and Colour Sensor.
Activities: Build a colour detection game (Colour Blindness Tester).
Challenge 1: Program EDU PICO to avoid repeating 2 similar colour during gameplay.
Challenge 2: Program a colour hint indicator by lighting up 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 data with Thonny's plotter.
- construct a Trashbot smart bin with card box accessories.

Introduction: Servo Motor and Proximity Sensor.
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 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.
Activities: Build a noise pollution monitoring system (Room Environment Noise Indicator).
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 to control a DC motor - spinning direction and speed control.
- turn ON and OFF a USB switch relay.
- program EDU PICO's Raspberry Pi Pico W into a WiFi access point for IoT applications.

Introduction: DC Motor and USB Relay.
Activities: Build a smart classroom integrated with a gesture sensor, USB relay, DC motor, and OLED.
Bonus: Control the USB relay through a webpage using the Raspberry Pi Pico W acting as a WiFi Access Point (AP).

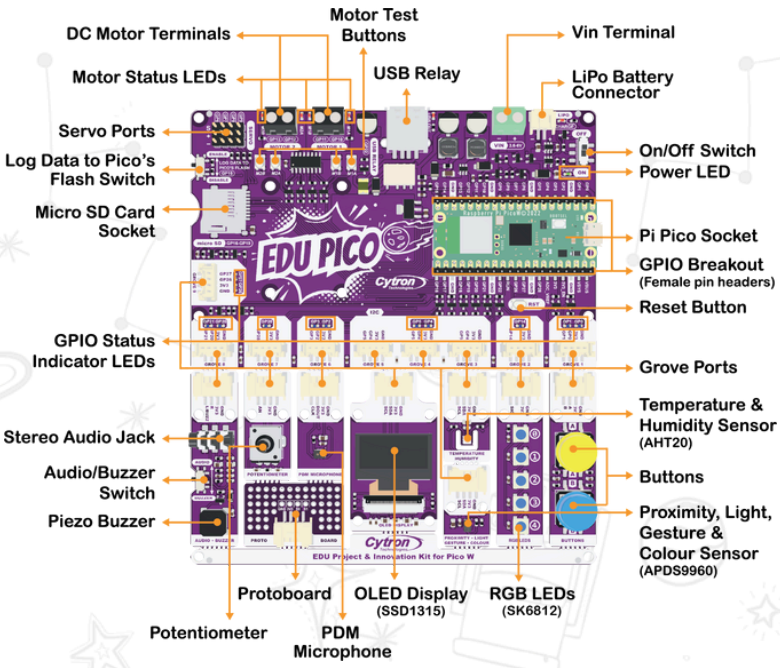
Chapter 8: Climate Control Greenhouse (Light and Humidity Temperature Sensor)

In this chapter, we learn to:

- program light sensor to measure ambient brightness.
- program AHT20 sensor for humidity and temperature measurement.
- perform basic data logging on Raspberry Pi Pico W local storage.

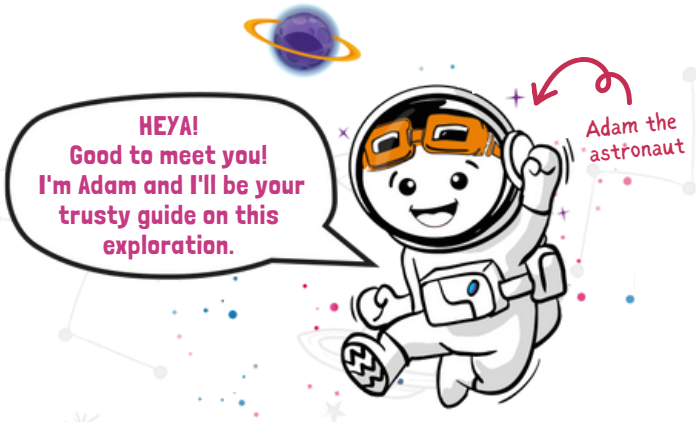
Introduction: Light Sensor, Humidity and Temperature Sensor, and Data Logging.
Activities: Build a climate control greenhouse while integrating temperature and humidity sensor, light sensor, DC and servo motor, RGB LEDs, and OLED.
Bonus 1: Introduction to the Internet of Things (IoT), learn to connect the Raspberry Pi Pico W to a router's WiFi and control the ON/OFF switch of the USB Relay.
Bonus 2: Introduction to Data Logging, record Raspberry Pi Pico W CPU's temperature data into the Pico's onboard storage.

HELLO EDU PICO



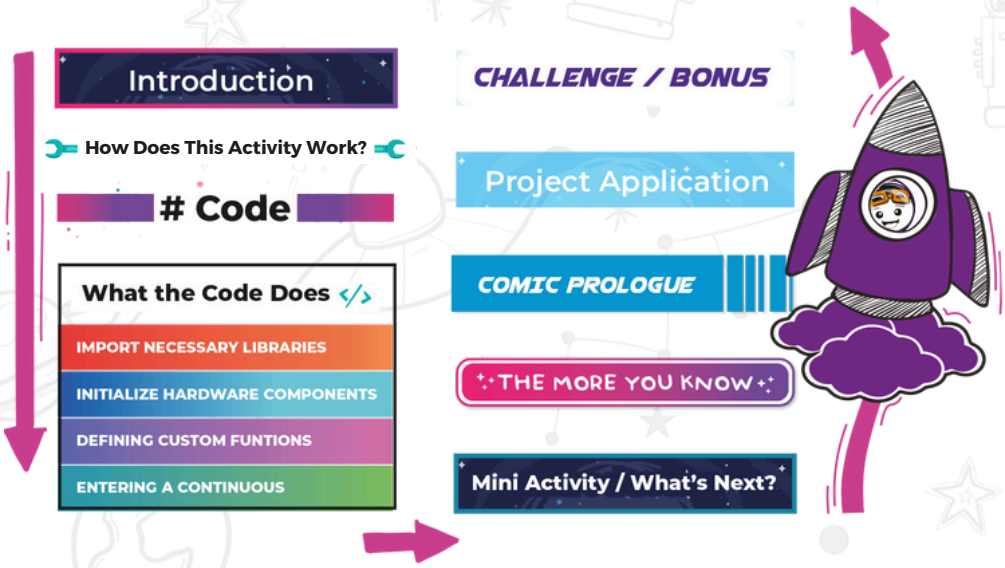
Preparing your EDU PICO

How To Use This Guidebook?



All programs and content discussed in this guidebook are available for download on the EDU PICO resource hub website here: https://edupico-hub.cytron.io.

Learning Flow



Every Chapter shares a common flow to help you fully experience learning with the EDU PICO. Remember, this is an exploratory guide, so don't rush, take your time and enjoy the process!

CHAPTER 1

Programming with CircuitPython

1.1 Start Here - Thonny IDE

1.2 Introduction to CircuitPython

1.3 Hello EDU PICO

1.4 CircuitPython Syntax

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

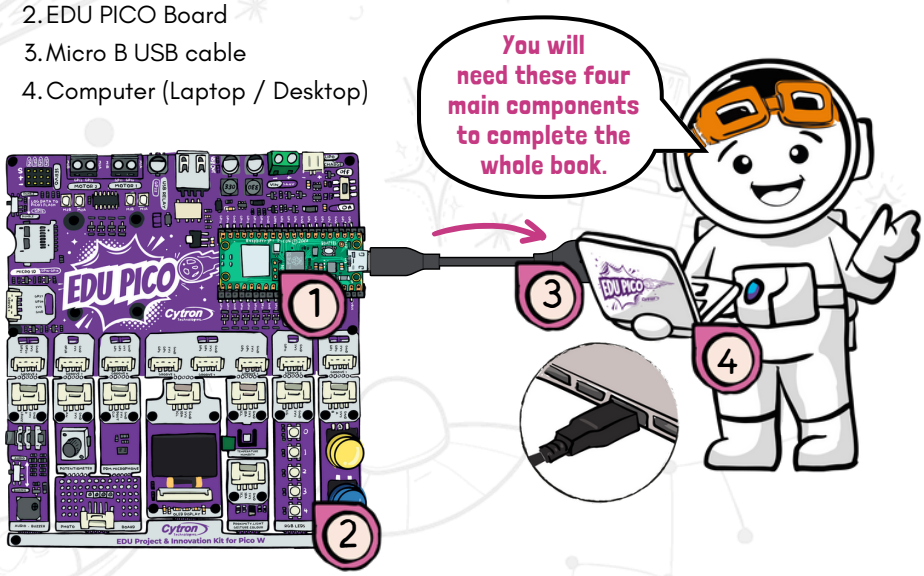
Start Here - Thonny IDE

EDU PICO is an all-in-one learning tool that allows you to practice electronics and CircuitPython programming using the Raspberry Pi Pico 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.

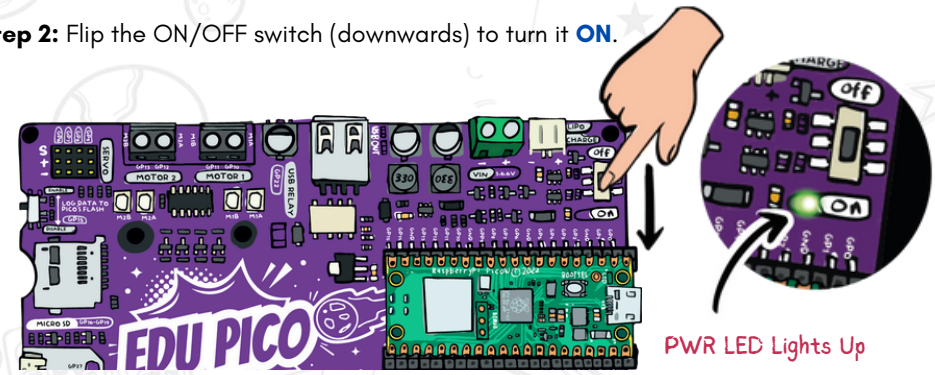
What You Need to Start?

Step 1: Connect the EDU PICO to your computer.

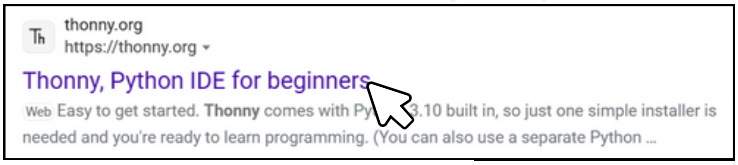
1. Raspberry Pi Pico W Microcontroller
2. EDU PICO Board
3. Micro B USB cable
4. Computer (Laptop / Desktop)

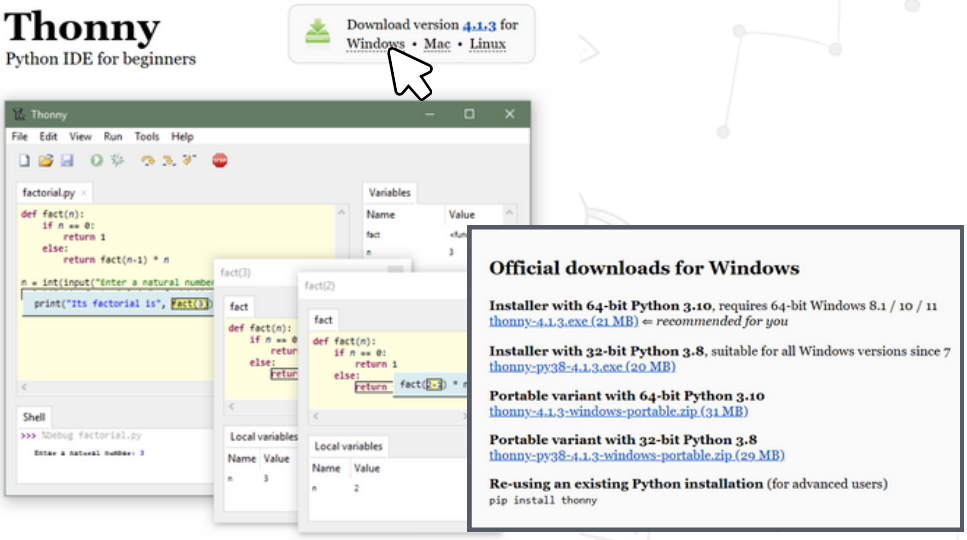


Step 2: Flip the ON/OFF switch (downwards) to turn it **ON**.



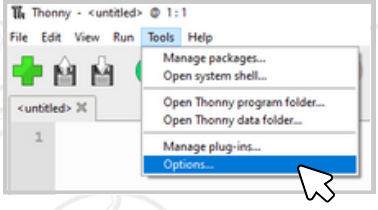
Step 3: Download and install **Thonny IDE** on your computer.





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.

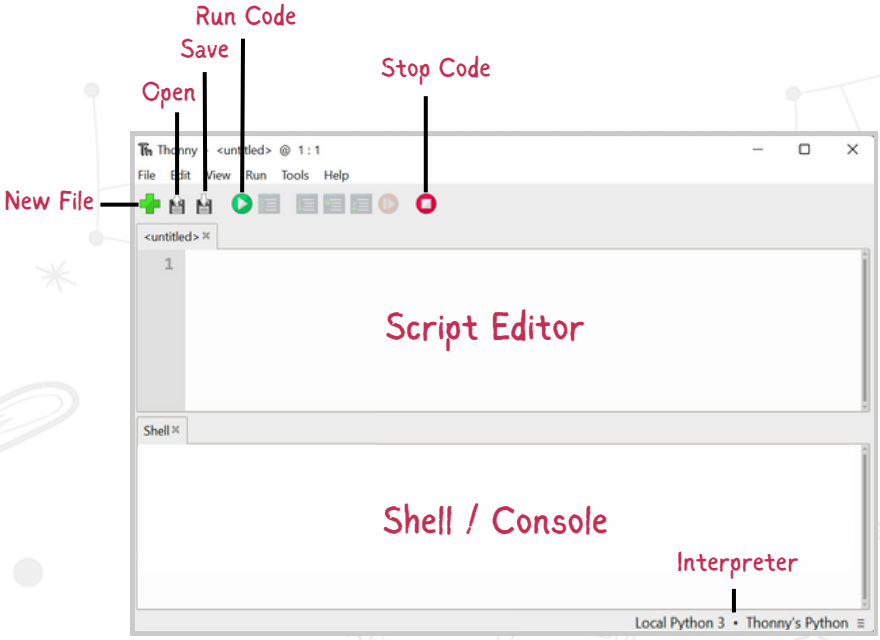
Select **Tools > Options > Theme & Font** as shown in the figure below. Adjust the IO font, UI theme, Syntax theme, and Editor font accordingly.



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.

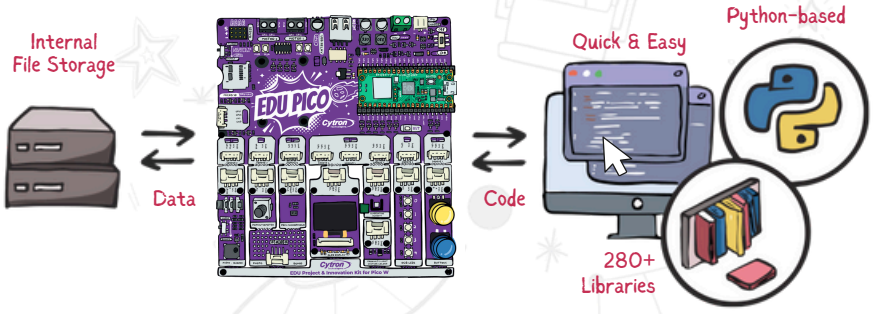
Lastly, Thonny IDE was originally designed for teaching Python and emphasizes learning and understanding code. Hence why it is an excellent choice for educators and students to practice programming microcontrollers with Python.

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 and Raspberry Pi Pico W.

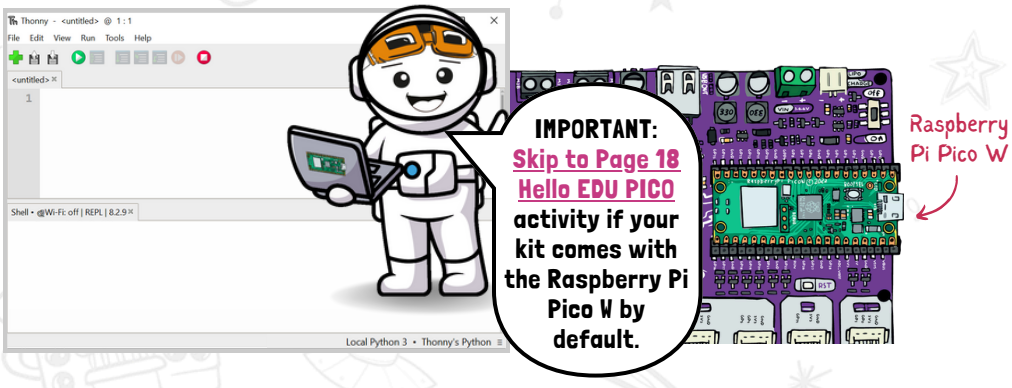
Why CircuitPython?

- **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.

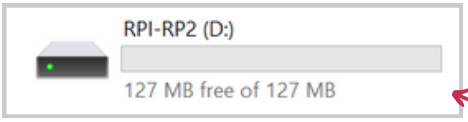


Preparing EDU PICO CircuitPython Firmware

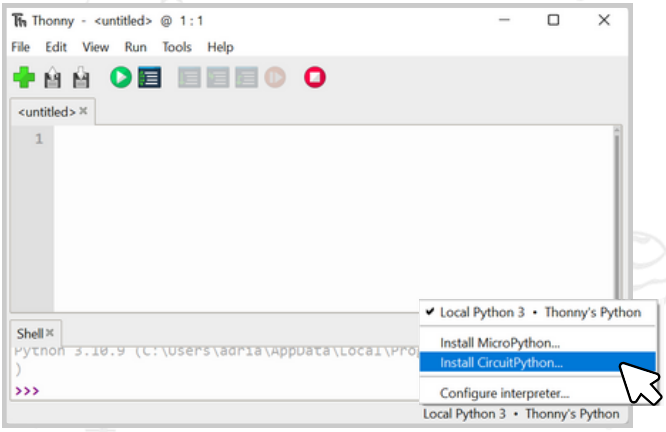
Step 1: Launch Thonny IDE and make sure your computer has internet connection.



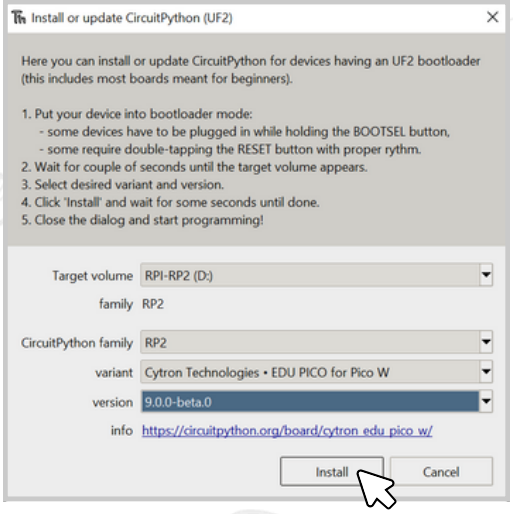
Step 2: Press and hold the reset (RST) and BOOTSEL buttons, then release the RST button, but continue to hold the **BOOTSEL button** until the **RPI-RP2** drive appears.



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

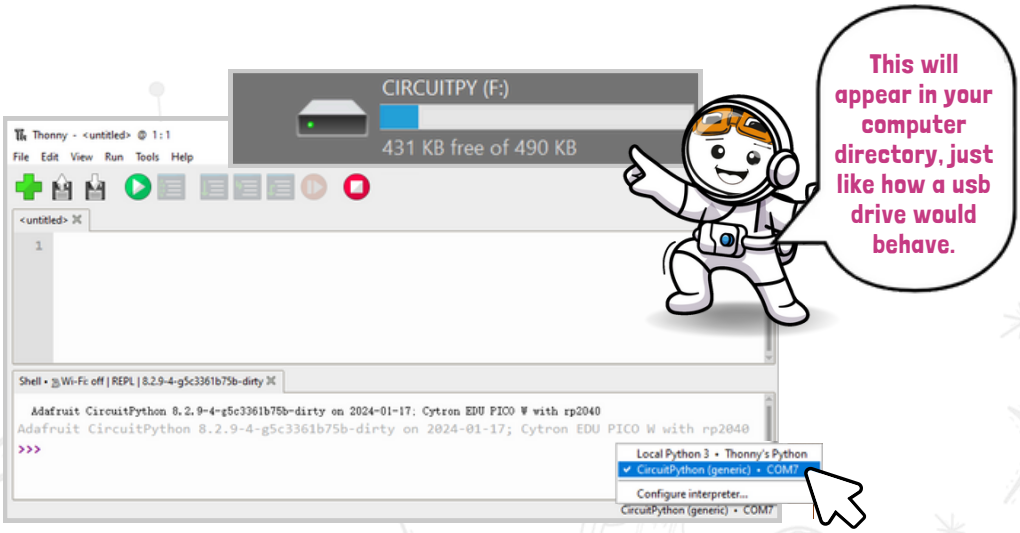


Step 4: Select **RPI-RP2** for the target volume, and **Cytron Technologies - EDU PICO** for the variant. Then select the latest stable version and Click **Install**.



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.



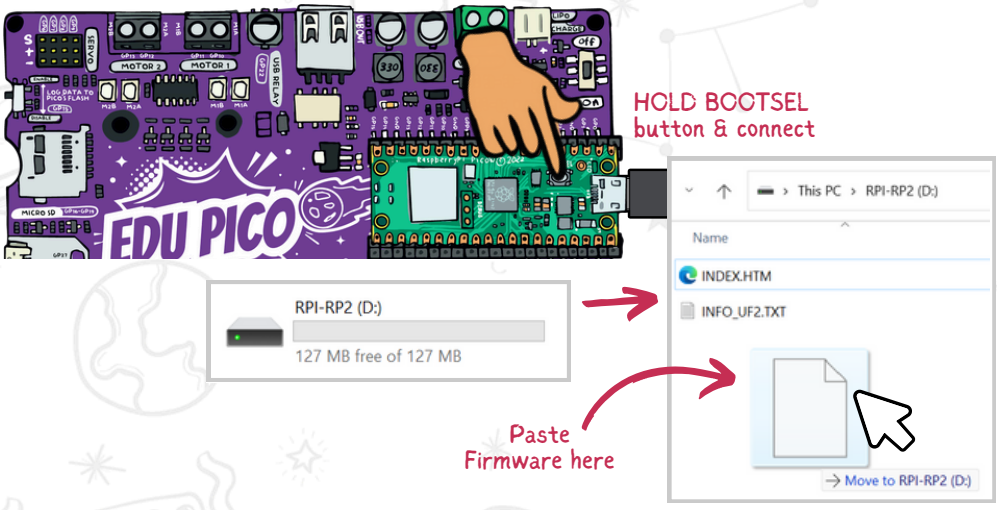
Download EDU PICO Firmware

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

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

Copy the firmware into the **RPI-RP2** Drive to install the firmware.

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

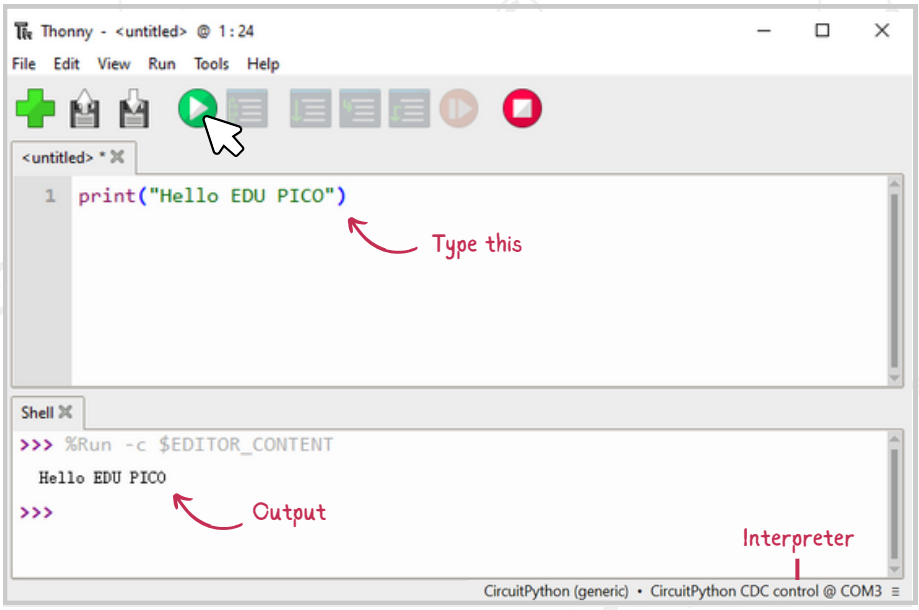


Go ahead and launch the Thonny IDE and you're all set for your first EDU PICO program!

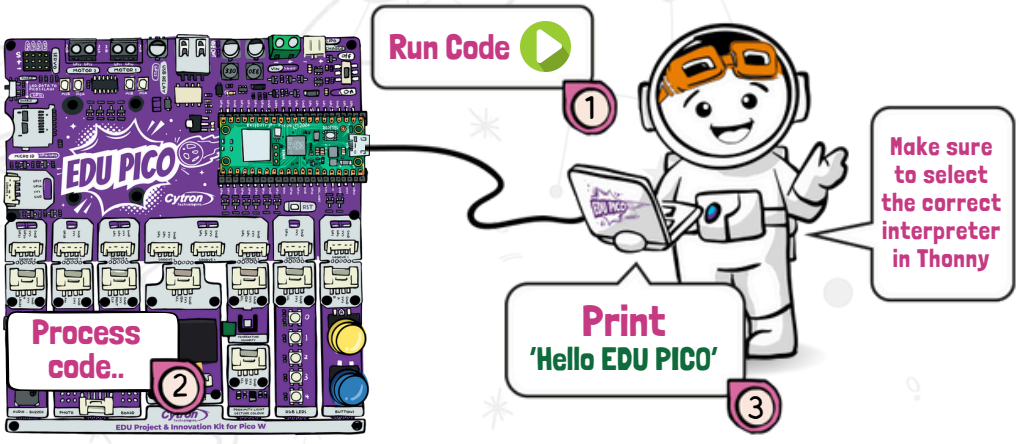
Hello EDU PICO

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

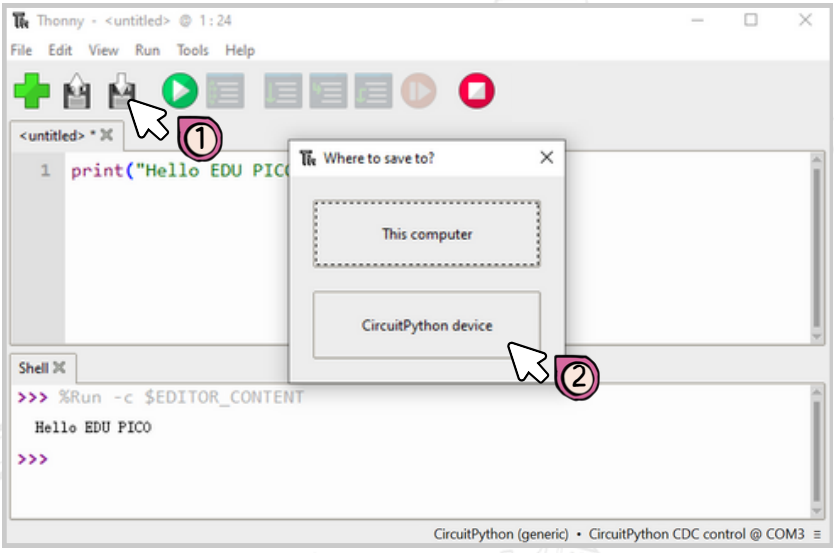
Step 1: Program the EDU PICO to print "Hello EDU PICO" text at the shell.



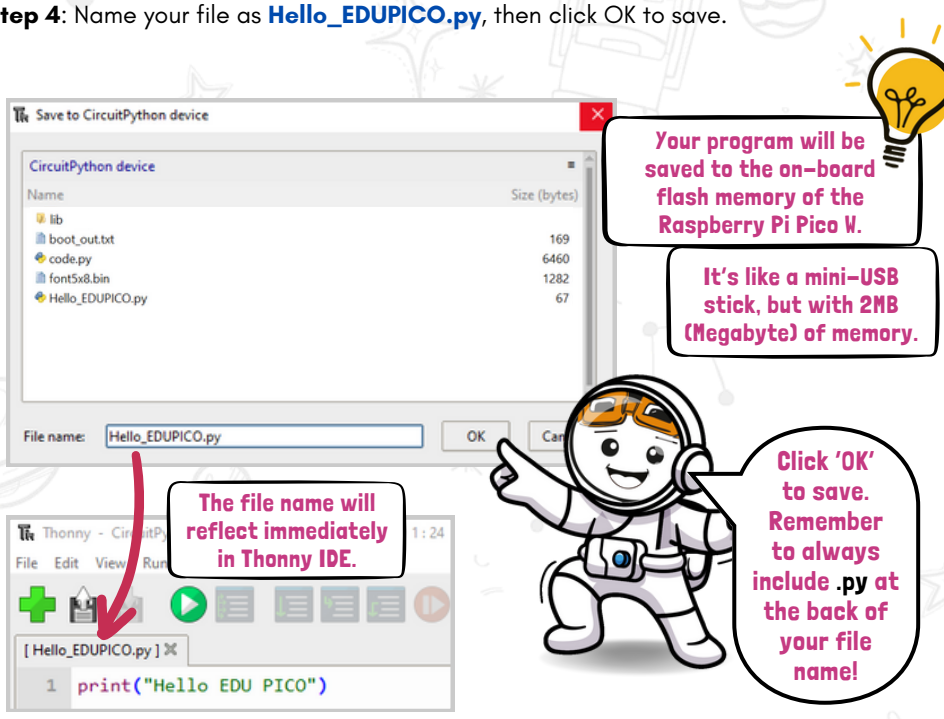
Step 2: Click the Green Button to run the code and Red Button to stop.



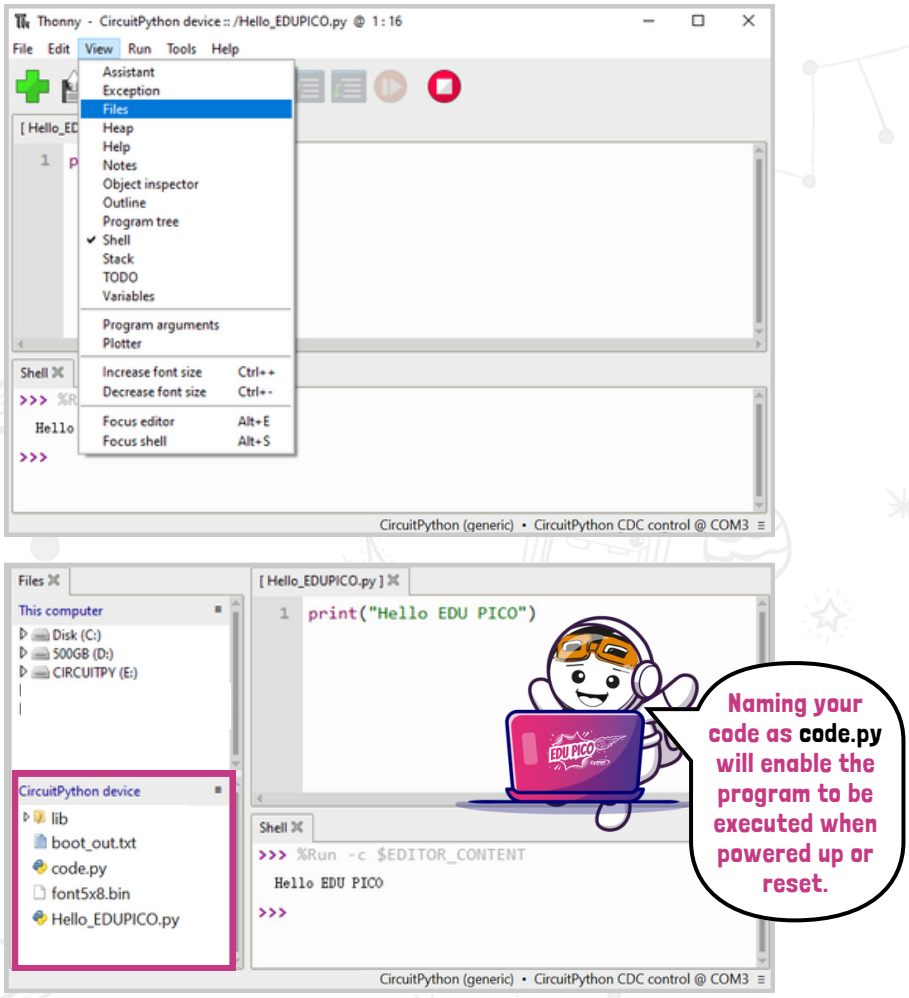
Step 3: Click **Save** and save your code into the CircuitPython device.



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 W easily; simply open the code by double-clicking the file.

To prevent **code.py** from running automatically when the EDU PICO 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 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.

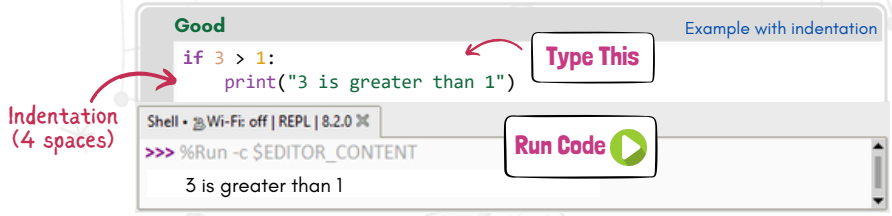
Hence, it is always recommended to use a different name for your code to avoid any naming conflicts with the CircuitPython library.

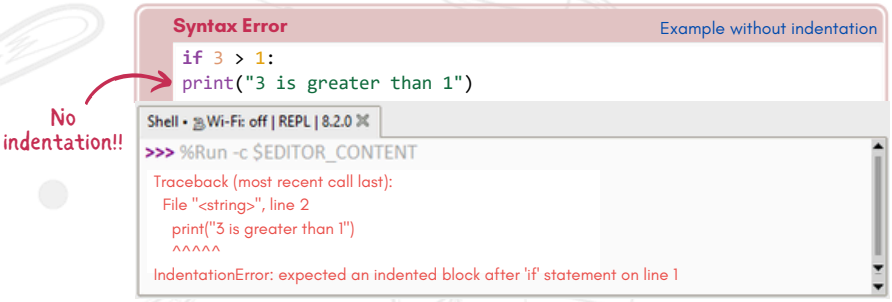
For an example, naming your file as neopixel.py will result in an error because the Neopixel library is already being named as neopixel.mpy.

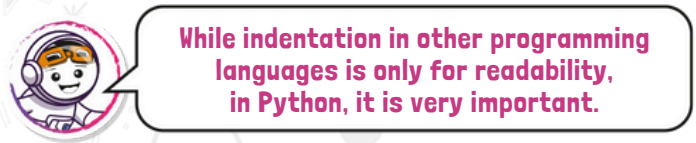
In the next section, we will learn the basic syntax of CircuitPython.

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.







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

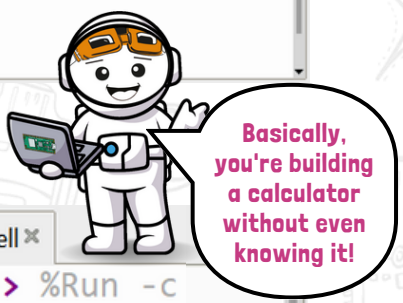
```
x = 10
y = 3

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
```

Type This

Run Code

Shell

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



2. Assignment Operator

ASSIGNMENT_OPERATOR.py

```
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)
```

Type This

Run Code

Shell

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

3. Comparison Operator

COMPARISON_OPERATOR.py

```
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
```

Type This

Shell

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

Run Code

COMPARISON_OPERATOR_2.py

```
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
```

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

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

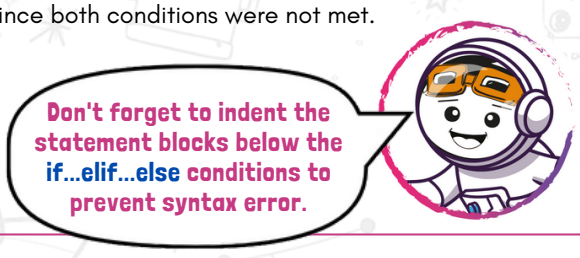
```
x = 3
y = 10

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.



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**.

While_Loop.py

```
x = 1
while x < 4:
  print(x)
  x += 1
```

Type This

Shell

```
>>> %Run -c $ED
1
2
3
```

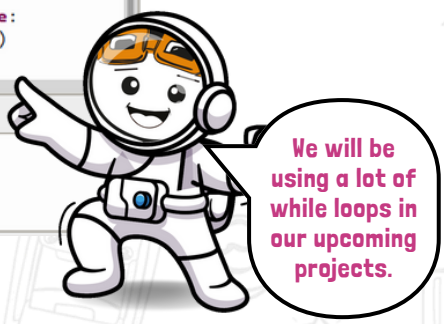
Run Code

While_Loop_2.py — Inifinity Loop

```
x = 1
while True:
  print(x)
  x += 1
```

Shell

```
239
240
241
242
```



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.

Function_1.py — Define and call print_function

```
def print_function(): #define function
  print("Print from Function")

print_function() #call function
```

Type This

Shell

```
>>> %Run Function_1.py
  Print from Function
>>>
```

Run Code

Function_2.py — Call function with argument & return value

```
def multiplication(x):
  return 3 * x

print(multiplication(2))
print(multiplication(5))
```

Type This

Shell

```
>>> %Run -c $EDITOR_CONTENT
6
15
```

Run Code

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))
```

Shell

Run Code

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

Congratulations, you're all set to embark on your next exploration! Rest assured, I'll be with you every step of the way.

CHAPTER 2

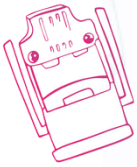
Water Drinking Reminder

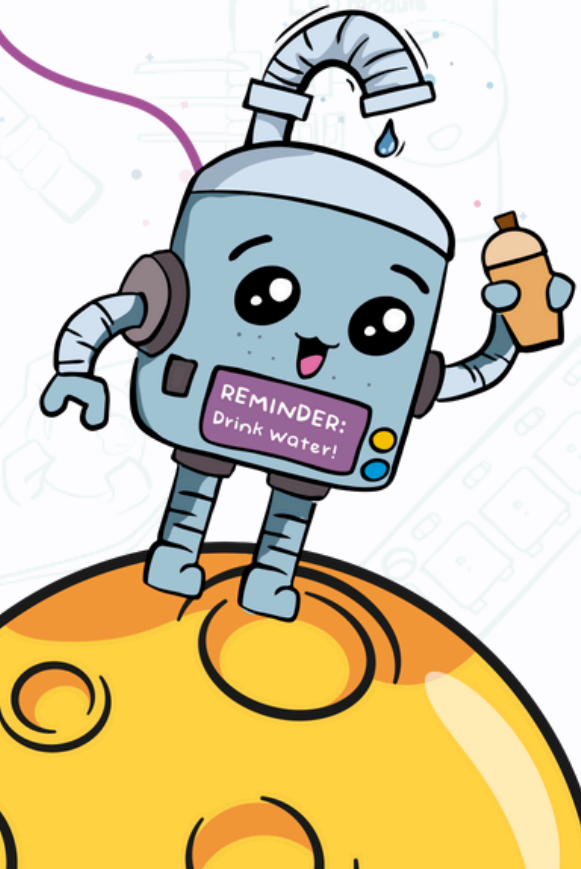
Buttons & Buzzer

2.1 Introduction to Buttons

2.2 Introduction to Buzzer

2.3 Project: Water Drinking Reminder





Hello, Makers! Today, we're going to embark on a fascinating journey by delving into two fundamental electronic components: buttons and piezo buzzer. These seemingly ordinary devices play vital roles in our daily lives, from the buttons on our smartphones to buzzers that alert various events.

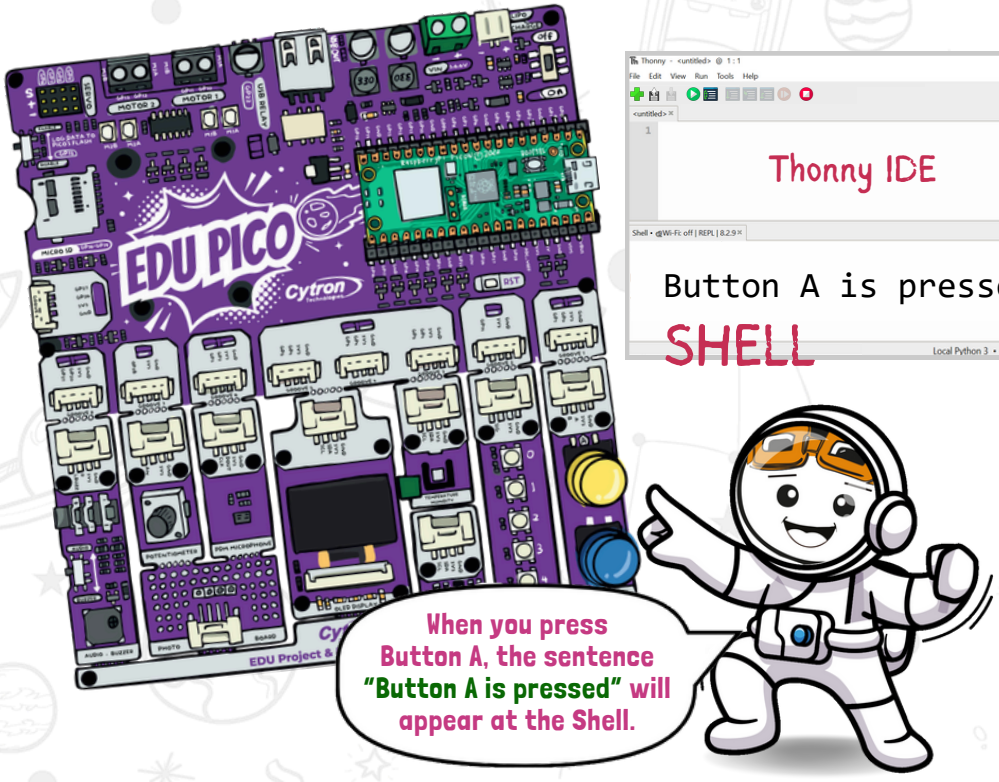
In this chapter, we'll learn how they work and why they are so crucial in real-life applications. So, buckle up and get ready to press some buttons and create some buzz!

Introduction to Button

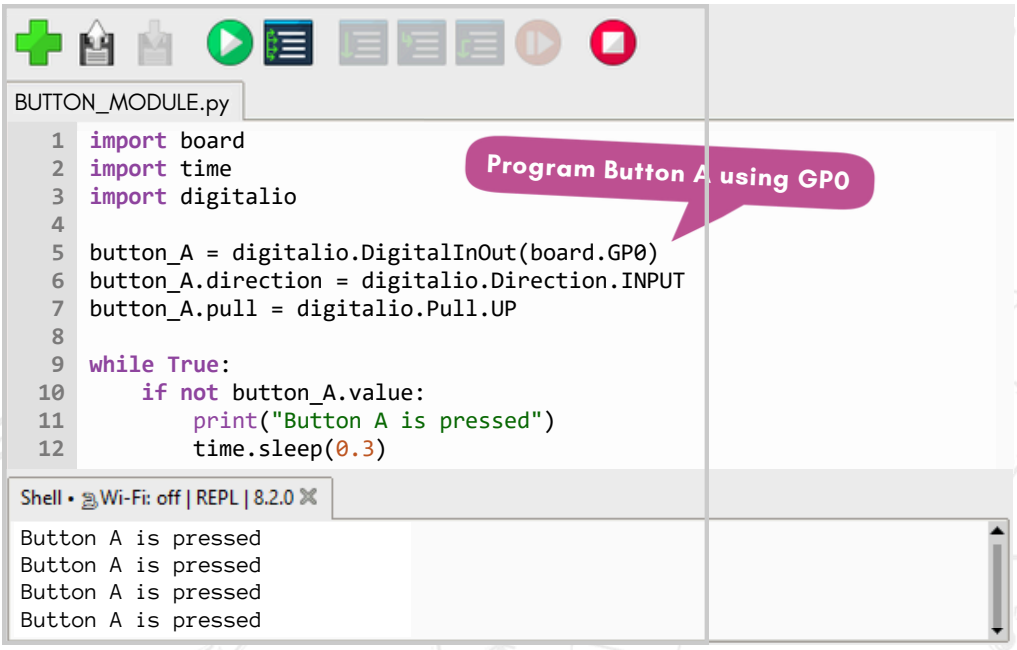
Buttons are essential electronic components in modern technology that allow us to input commands and interact with various devices. They come in all shapes and sizes, from the tiny buttons on your calculator to the larger ones on your computer keyboard.

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.



Code




```
1  import board
2  import time
3  import digitalio
4
5  button_A = digitalio.DigitalInOut(board.GP0)
6  button_A.direction = digitalio.Direction.INPUT
7  button_A.pull = digitalio.Pull.UP
8
9  while True:
10     if not button_A.value:
11         print("Button A is pressed")
12         time.sleep(0.3)
```

```
Button A is pressed
Button A is pressed
Button A is pressed
Button A is pressed
```

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

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

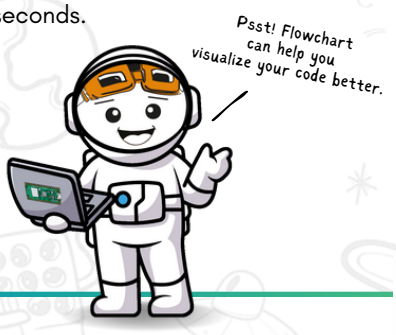
```
9 while True:
```

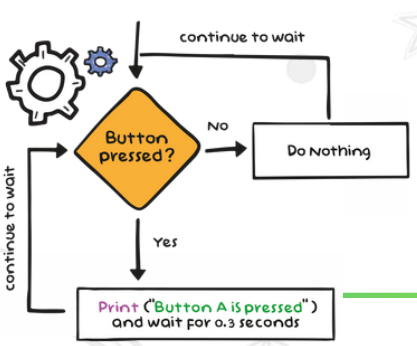
while True initiates an infinite loop. The code inside this loop will run repeatedly as long as the condition remains true.

Read Button Value

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

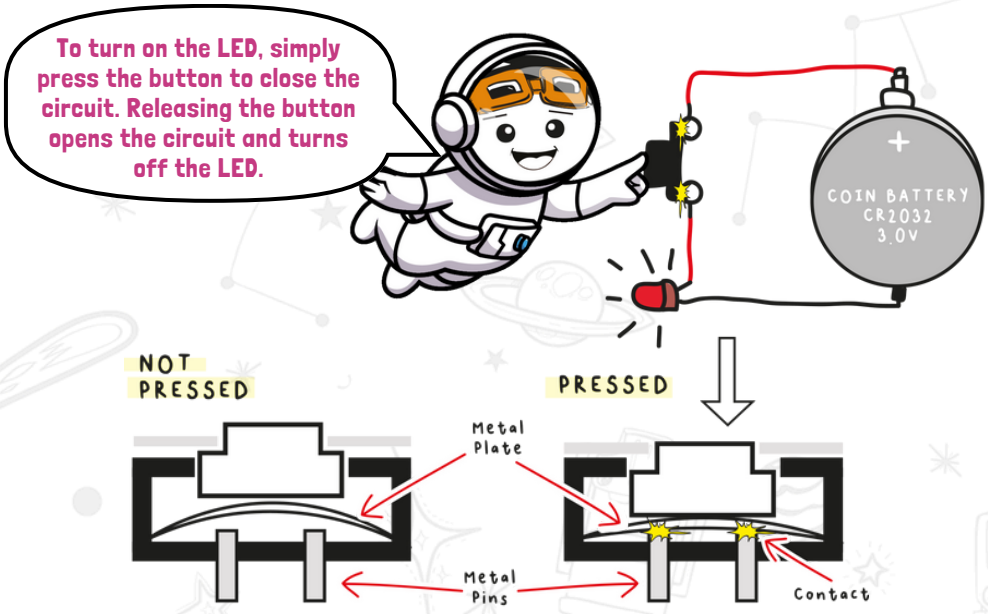
Line 10 - 12: Checks the value of button A. **button_A.value** returns True if the button is not pressed (logic high) and False if the button is pressed (logic low).
If the condition is True, it prints the message "Button A is pressed" and is delayed for **0.3** seconds.





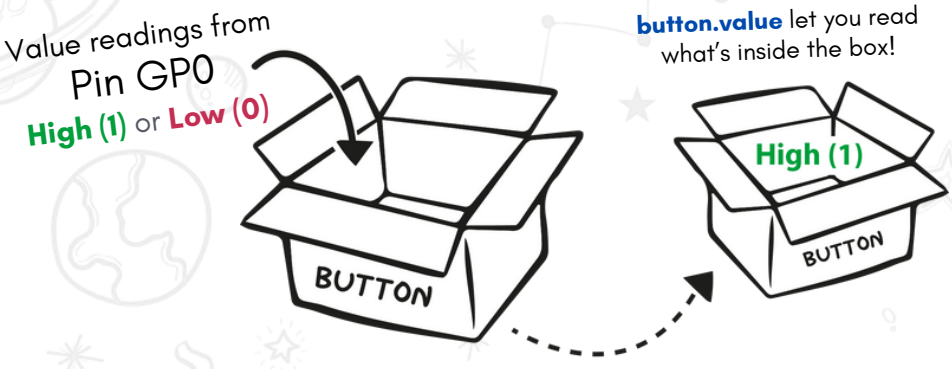
THE MORE YOU KNOW

Buttons



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.

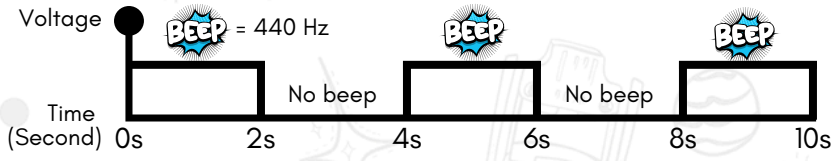


Introduction to Buzzer

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!

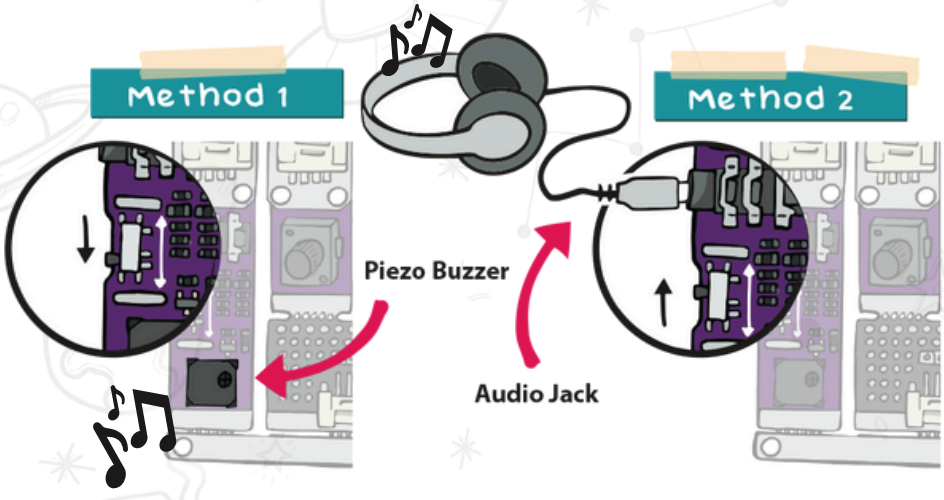
How Does This Activity Work?

- **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).

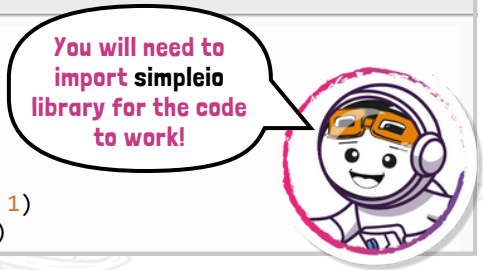
2 Methods to use the Audio Buzzer Module



Code

BUZZER_MODULE.py

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



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_pin = board.GP21
```

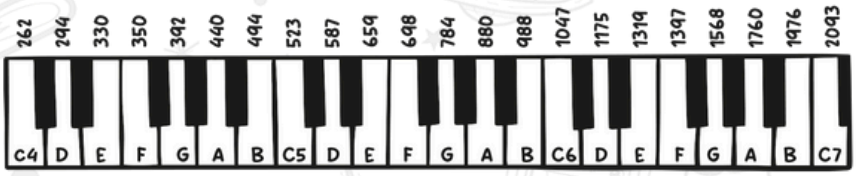
Line 4: Assigns ***GP21*** to the variable **buzzer_pin** to control the buzzer module.

Tone Frequency

```
6  while True:
7      simpleio.tone(buzzer_pin, 440, 1)
8      simpleio.tone(buzzer_pin, 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.

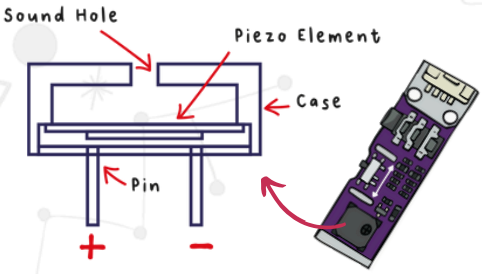
THE MORE YOU KNOW



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.

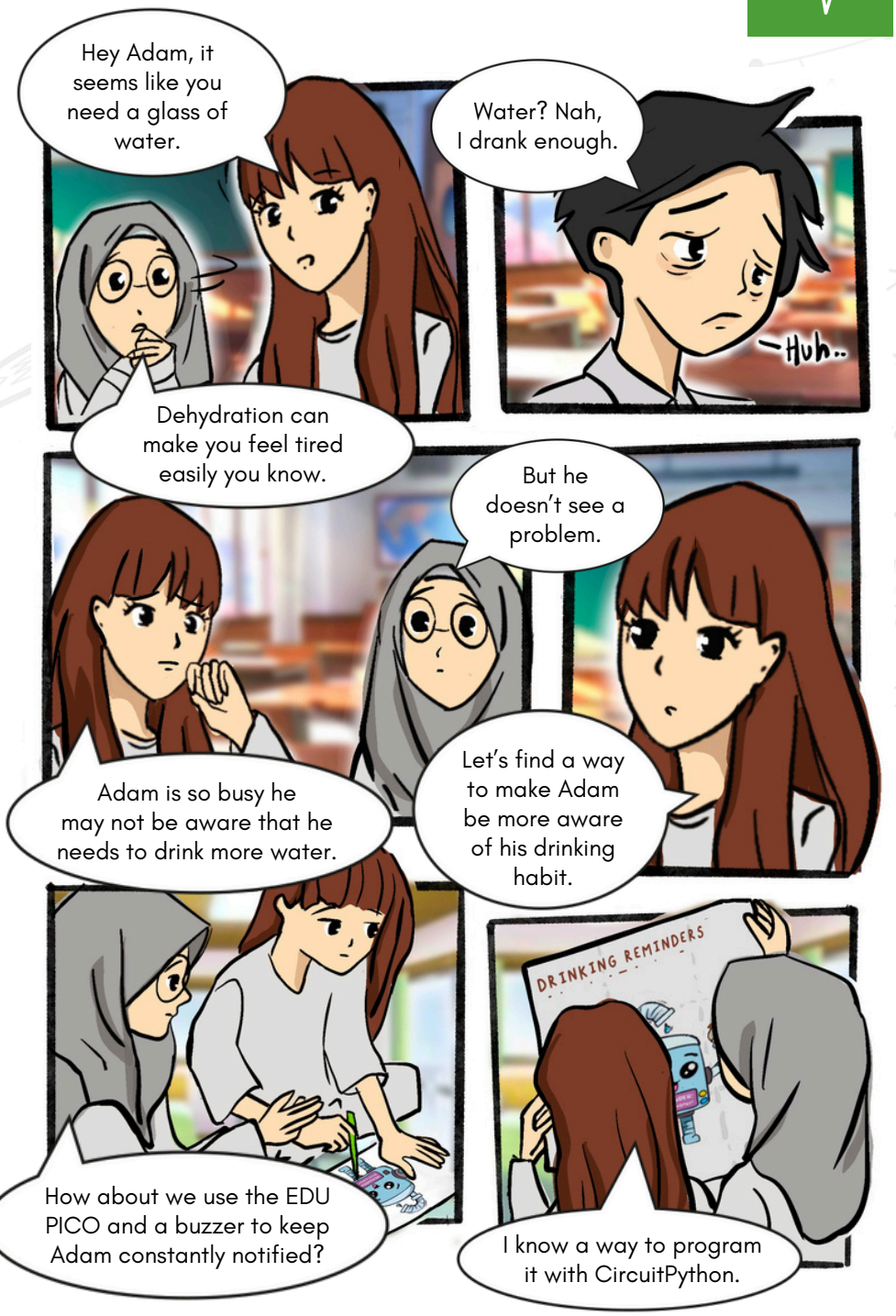
Piezo Buzzer

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.

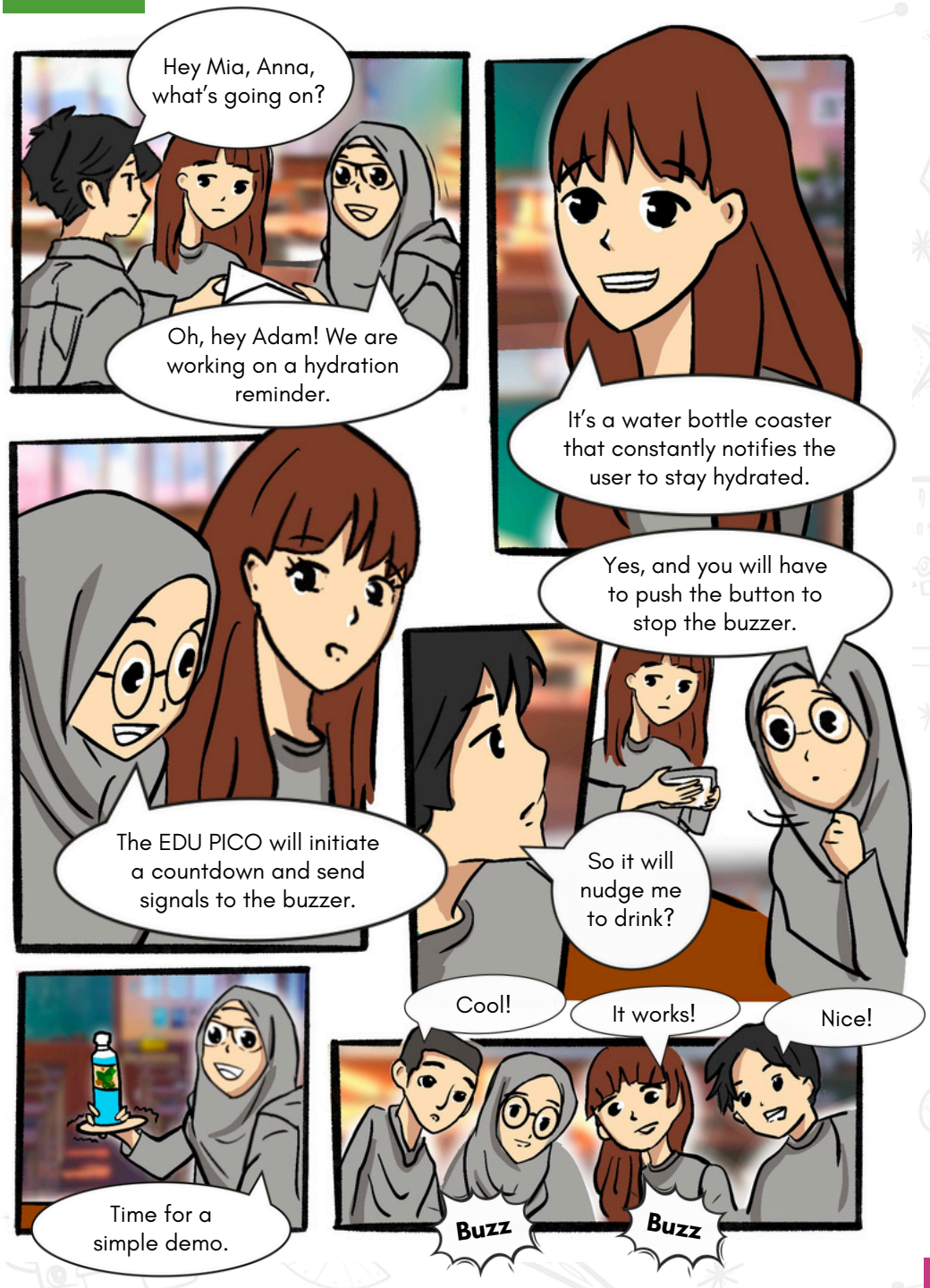


PROLOGUE: WATER DRINKING REMINDER

3 GOOD HEALTH AND WELL-BEING

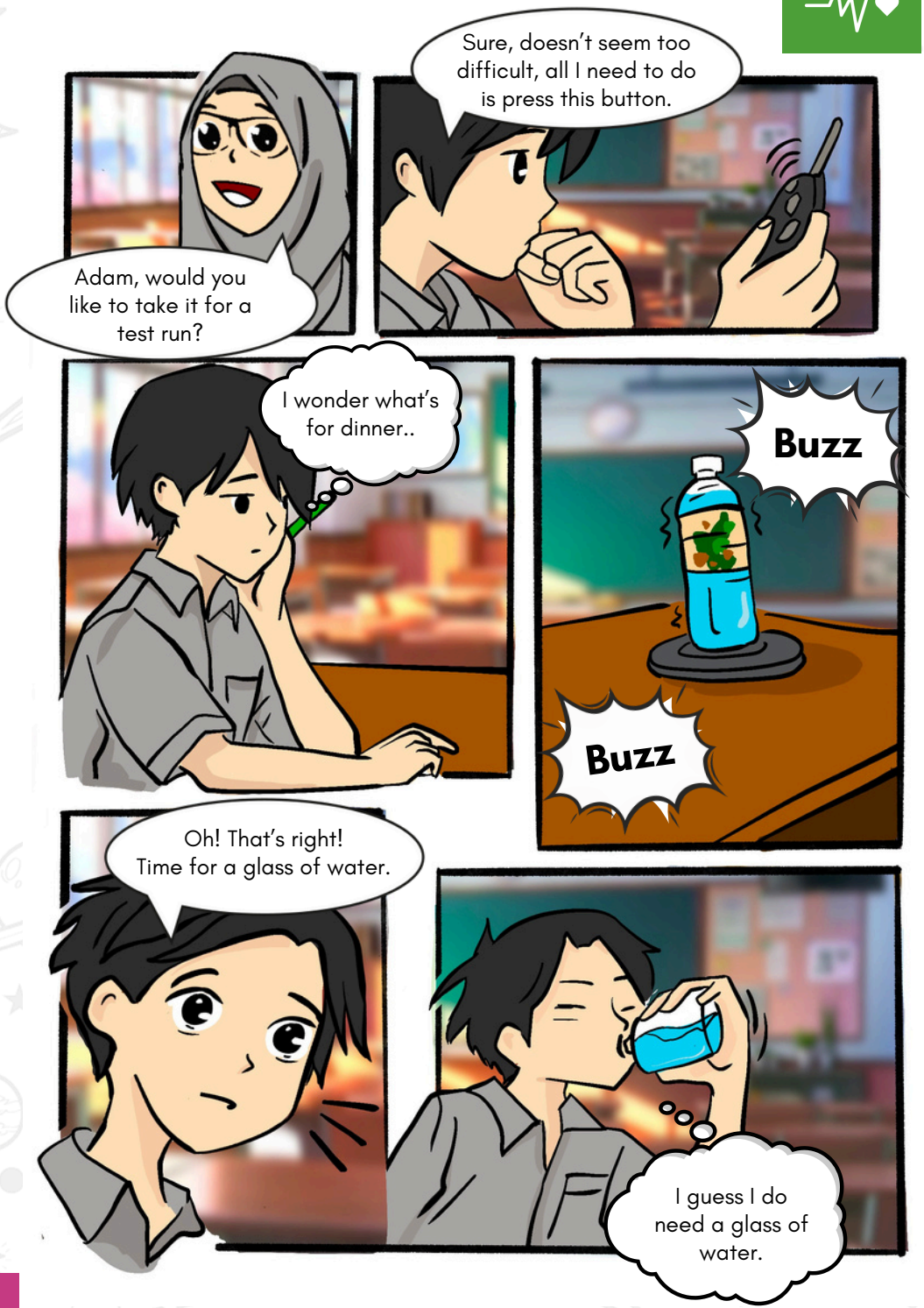


PROLOGUE: WATER DRINKING REMINDER

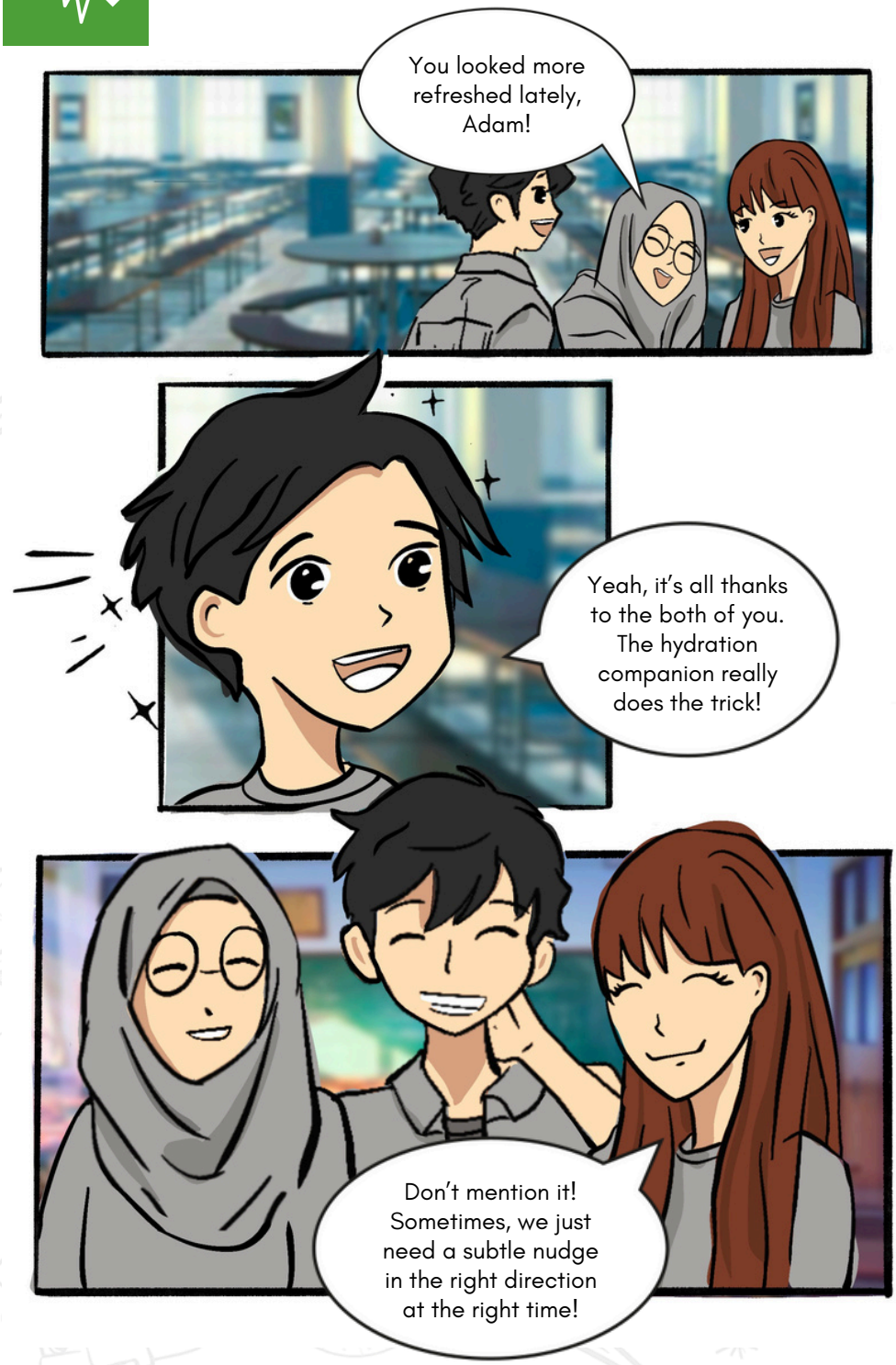


PROLOGUE: WATER DRINKING REMINDER

3 GOOD HEALTH AND WELL-BEING



PROLOGUE: WATER DRINKING REMINDER



Water Drinking Reminder

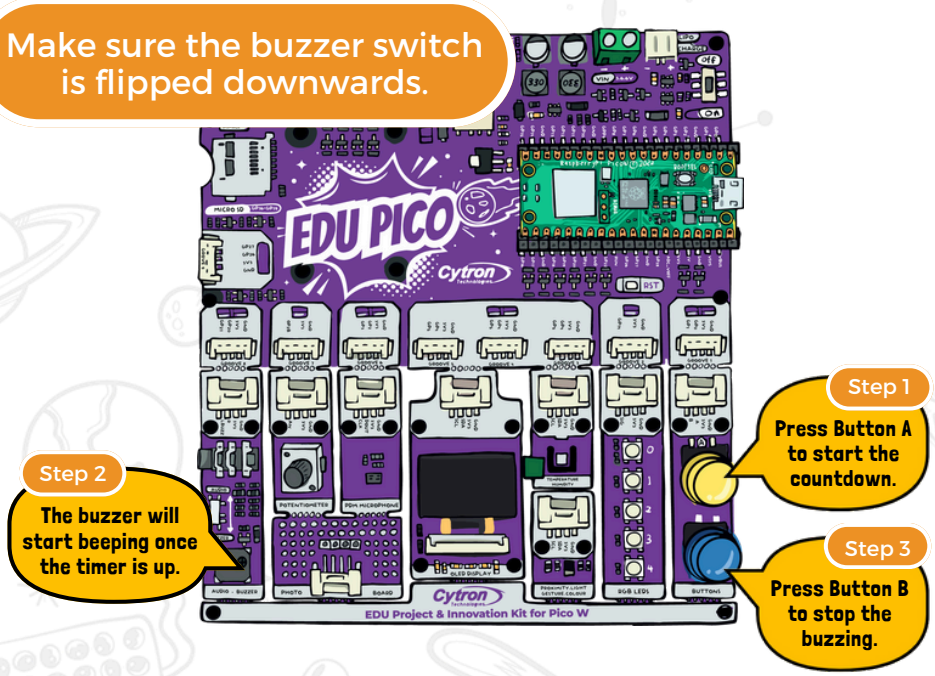
Hydration Companion

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!

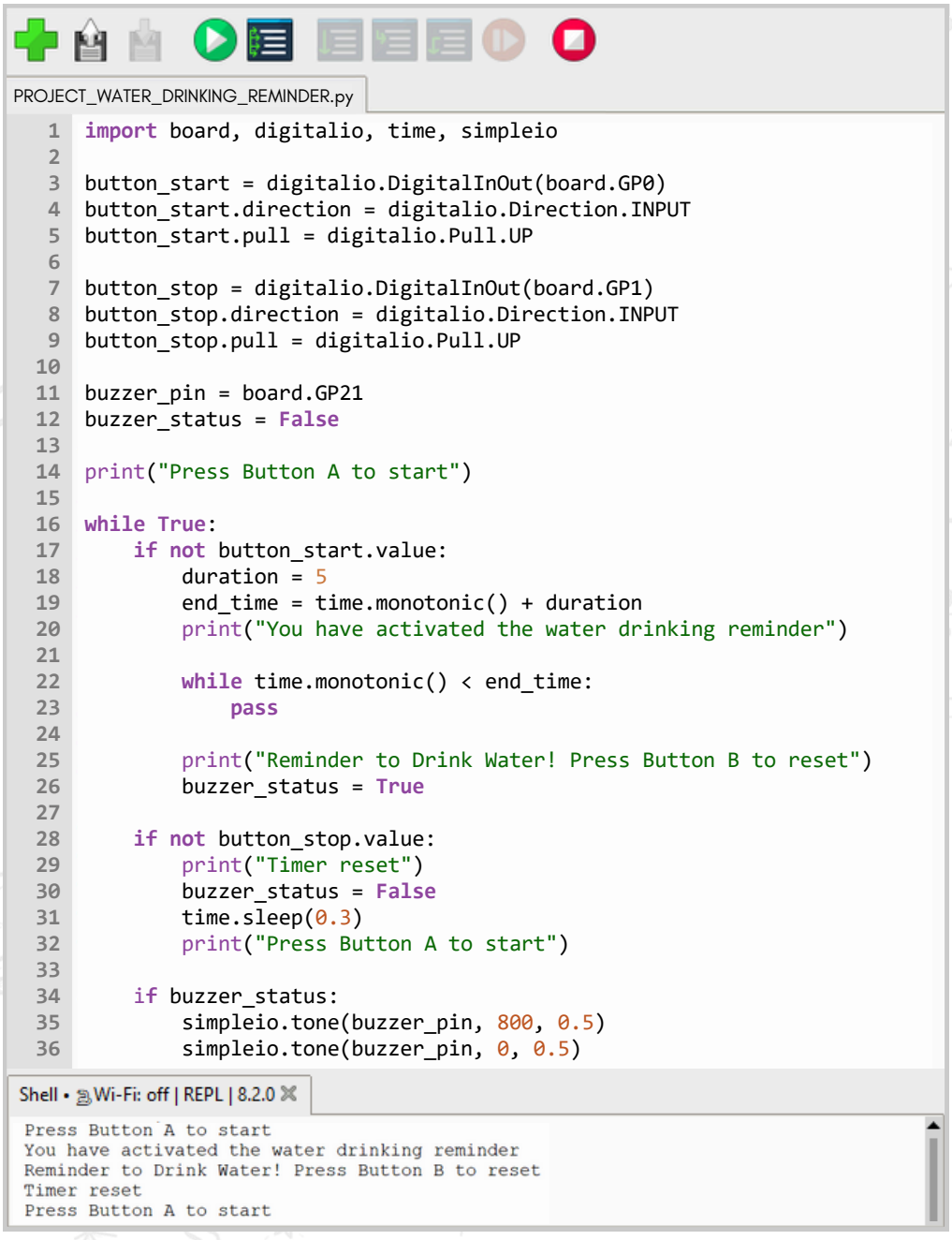
How Does This Activity Work?

- **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 • Wi-Fi: off | REPL | 8.2.0

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

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

What the Code Does </>

Import Necessary Libraries

Libraries

```
2 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: Prepares the buzzer to make sounds and sets 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 effectively waits 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: Prints "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 an **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.

What's Next?

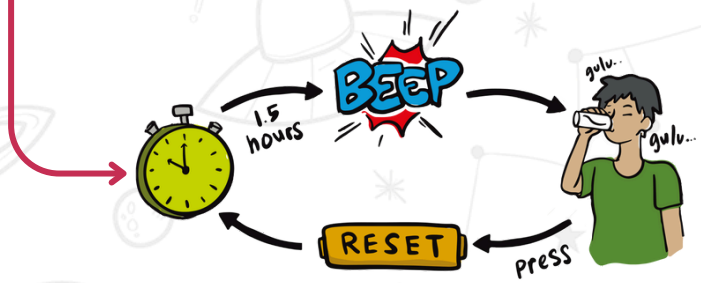
Once you're done with the example code, it's time we turn this project into a real-life application! Before doing that, let's list out all the facts about drinking water healthily.

According to the World Health Organization (WHO), we should drink 2 to 3 litres (8 to 12 cups) of water per day. Say that we have to drink 8 cups in a day, to spread out across 12 hours in a day, the timer duration can be calculated this way:

12 hours / 8 cups = 1.5 hours (90 minutes = 5,400 seconds)

button_start

```
16  while True:
17      if not button_start.value:
18          duration = 5400
19          end_time = time.monotonic() + duration
20          print("You have activated the water drinking reminder")
```



Let's take it one step further to identify the amount of water we should drink per intake. We can use this formula to identify our daily water intake:

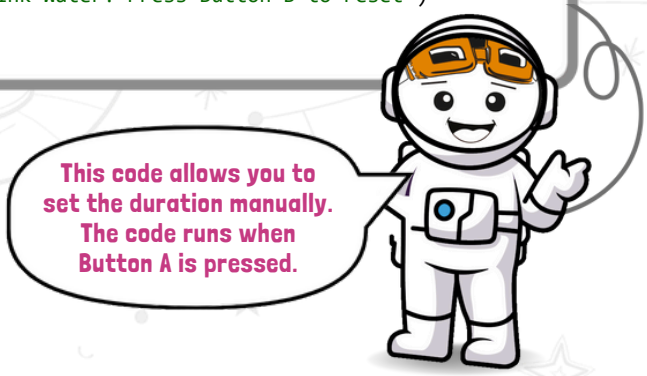
Daily water intake (in millilitres) = Body weight (in kilograms) x 33

For example, if you weigh 50 kilograms, you should drink between 1,650 to 3,300 millilitres of water per day. It's important to note that the amount of water intake required by an individual can vary depending on their body's needs and the environment they are in. It is also important to drink consistently throughout the day and not wait until you feel thirsty.

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
```



CHAPTER 3

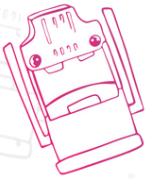
Gesture Reaction Game

OLED & Gesture Sensor

Introduction to OLED Display 3.1

Introduction to Gesture Sensor 3.2

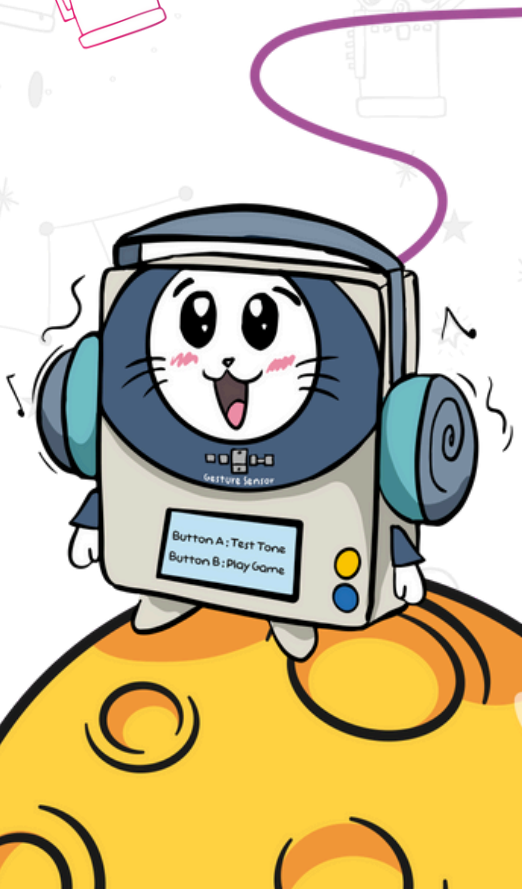
Project: Gesture Reaction Game 3.3



Hi Makers! In this chapter, we are about to explore these two amazing electronic components: gesture sensor and OLED display. These are very useful tools that can make our devices respond to our movements and bring vivid images to life on screens.

Imagine a world where you could control things just by waving your hand, or where your TV or watch could display brilliant, colourful images with perfect clarity. Well, today, we're going to explore the science and technology that make these wonders possible.

So, get ready to wave, swipe, and see the magic happen before your eyes as we discover the incredible world of gesture sensors and OLED displays.



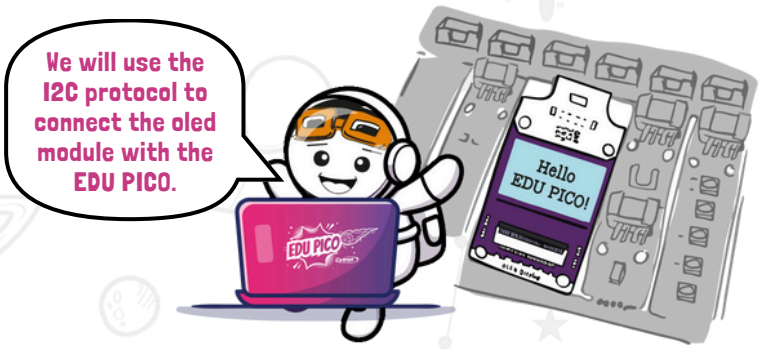
Introduction to OLED

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?

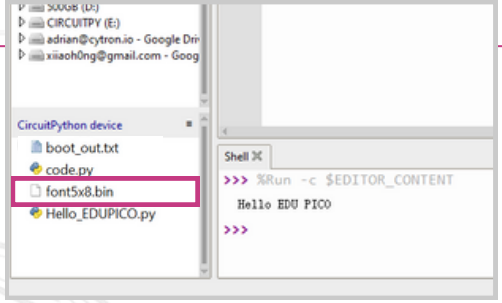
- **Libraries:** board, busio, time, adafruit_ssd1305, **font5x8.bin** ← Located in CircuitPython Root Directory
- **OLED I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Output:**
  - Invert OLED display at initialization with a bluish-white background.
  - Print "Hello EDU PICO" text in the middle of the OLED in black colour:
    - "Hello" with a coordinate of x = 50, y = 20.
    - "World" with a coordinate of x = 40, y = 35.

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



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!", 40, 35, 1)
11 oled.show()
```

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

What the Code Does </>

Import Necessary Libraries

Libraries

```
1 import adafruit_ssd1306
```

adafruit_ssd1306 enables operation with SSD1306 OLED displays.

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: Configures 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, 25, 1)
10 oled.text("EDU PICO!", 40, 40, 1)
11 oled.show()
```

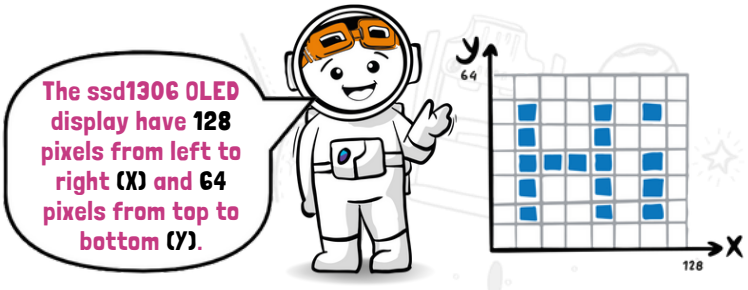
Line 7: Clears the OLED display by filling it with black (**0** represents black font).

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

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

Line 10: Displays the text "EDU PICO!" with the coordinates of x = **40** and y = **40**.

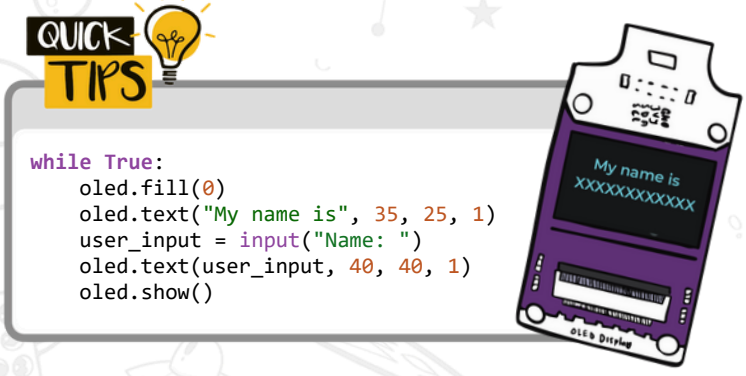
Line 11: Updates the OLED display to show the changes made.



In summary, this code initializes the OLED display and prints "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 OLED. Replace the original code from line 7 to 11 with the code provided below:




```
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()
```

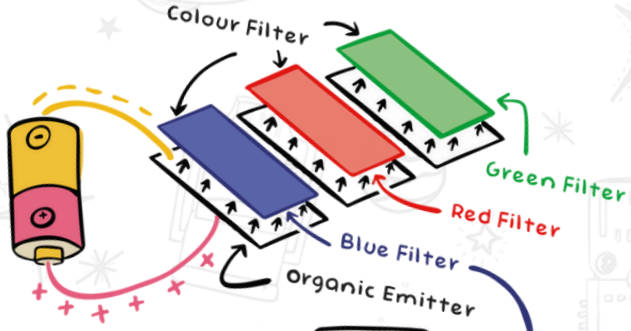
THE MORE YOU KNOW

OLED Pixel

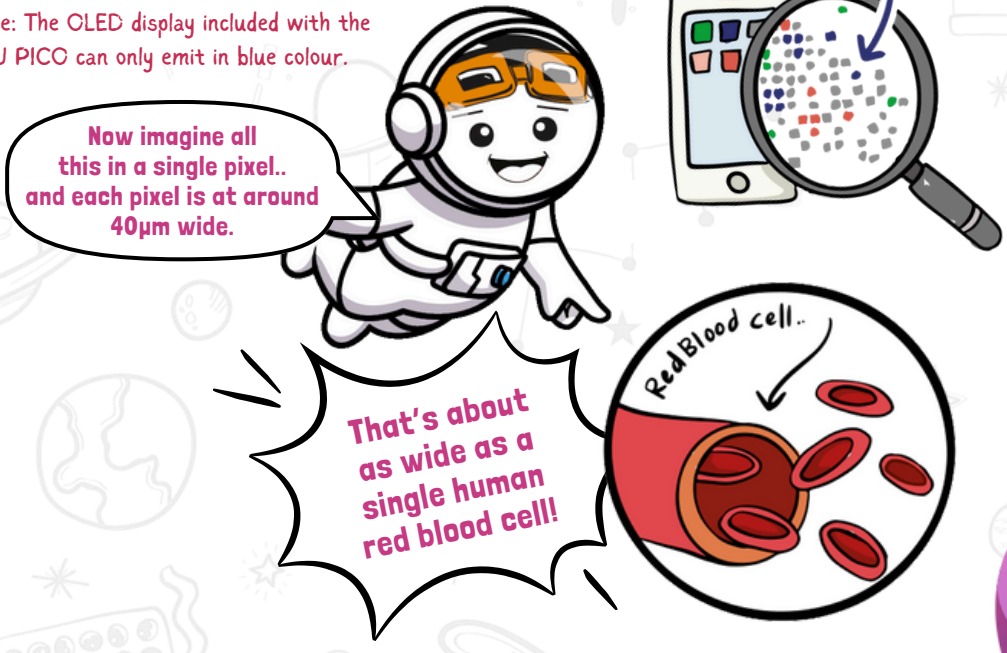
OLED (Organic Light-Emitting Diodes) are becoming the most common display technology in our devices. Most smartphones and many TVs today use OLED displays.

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

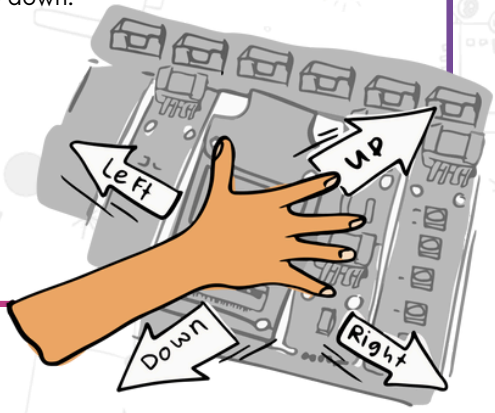
In this section, we'll explore how APDS9960 sensor can be used to detect hand gestures. The APDS9960 is a tiny electronic device that can "see" hand gestures and measure the amount of light around it. It's like having a tiny robot eye!

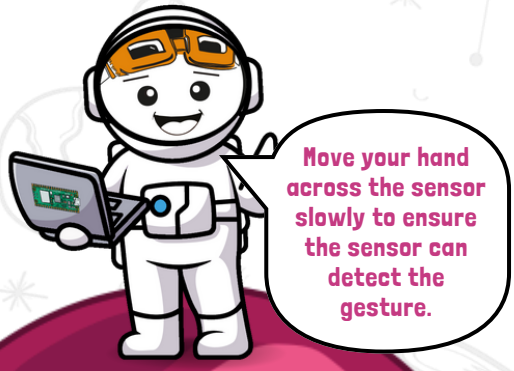
How Does This Activity Work?

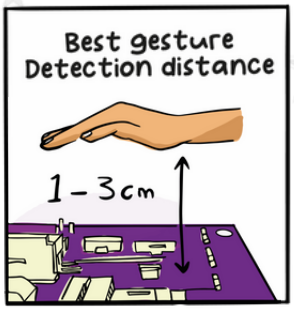
- **Libraries:** board, busio, adafruit_apds9960.
- **APDS9960 I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.

- **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:**

```
Shell • Wi-Fi: off | REPL | 8.2.0
>>> %Run -c $EDITOR_CONTENT

  left
  right
  up
  down
```







Code

GESTURE_SENSOR_MODULE.py

```
1  import board, busio
2  from adafruit_apds9960.apds9960 import APDS9960
3
4  i2c = busio.I2C(board.GP5, board.GP4)
5  apds = APDS9960(i2c)
6  apds.enable_proximity = True
7  apds.enable_gesture = True
8
9  while True:
10     gesture_value = apds.gesture()
11     if gesture_value == 1:
12         print("up")
13     elif gesture_value == 2:
14         print("down")
15     elif gesture_value == 3:
16         print("left")
17     elif gesture_value == 4:
18         print("right")
```

Shell • Wi-Fi: off | REPL | 8.2.0

```
left
right
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  from adafruit_apds9960.apds9960 import APDS9960
```

Line 2: Imports the APDS9960 class from **adafruit_apds9960.apds9960** module that corresponds to the APDS9960 sensor which is commonly used for gesture and proximity sensing.

APDS Functions

```
4  i2c = busio.I2C(board.GP5, board.GP4)
5  apds = APDS9960(i2c)
6  apds.enable_proximity = True
7  apds.enable_gesture = True
```

Line 5: Initializes an I2C communication bus with pins ***GP5*** (SCL - Serial Clock) and ***GP4*** (SDA - Serial Data), then create an instance that represents the APDS9960 sensor to allow the user to interact with it.

Line 6 - 7: Enable the proximity and gesture detection features of the APDS9960 sensor. Setting these properties to **True** activates the sensor's ability to detect when an object is nearby (proximity) and be able to recognize specific hand gestures.

Main Loop

```
9  while True:
10     gesture_value = apds.gesture()
11     if gesture_value == 1:
12         print("up")
13     elif gesture_value == 2:
14         print("down")
15     elif gesture_value == 3:
16         print("left")
17     elif gesture_value == 4:
18         print("right")
```

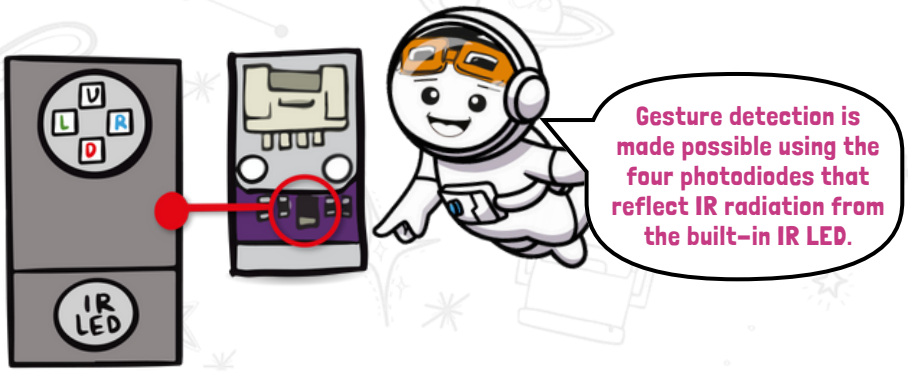
Line 10: Calls the **gesture()** method to detect a hand gesture. The method returns a numeric value representing the detected gesture.

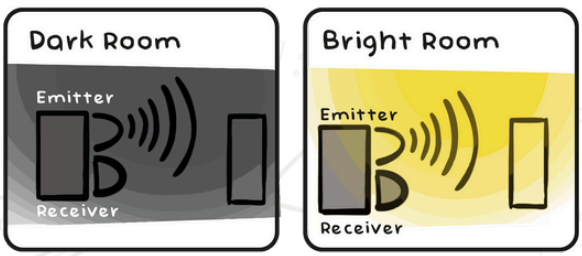
Line 11 - 18: Check the value of the gesture and print a corresponding message depending on the detected gesture. The APDS9960 library assigns specific numeric codes (e.g., **1** for "up", **2** for "down", **3** for "left", **4** for "right") to different gestures.

THE MORE YOU KNOW

APDS9960 Gesture Sensor

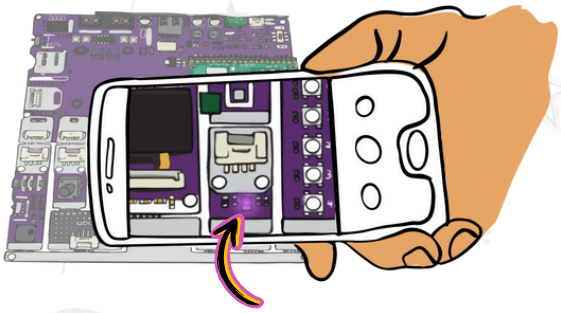
Imagine the APDS9960 as a little camera with tiny sensors that can detect infrared light (which we can't see with our eyes). When you move your hand or an object in front of the sensor, the invisible light reflects on the object, allowing the sensor to detect the reflected infrared light, almost like 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 APDS9960 sensor is still functioning properly.

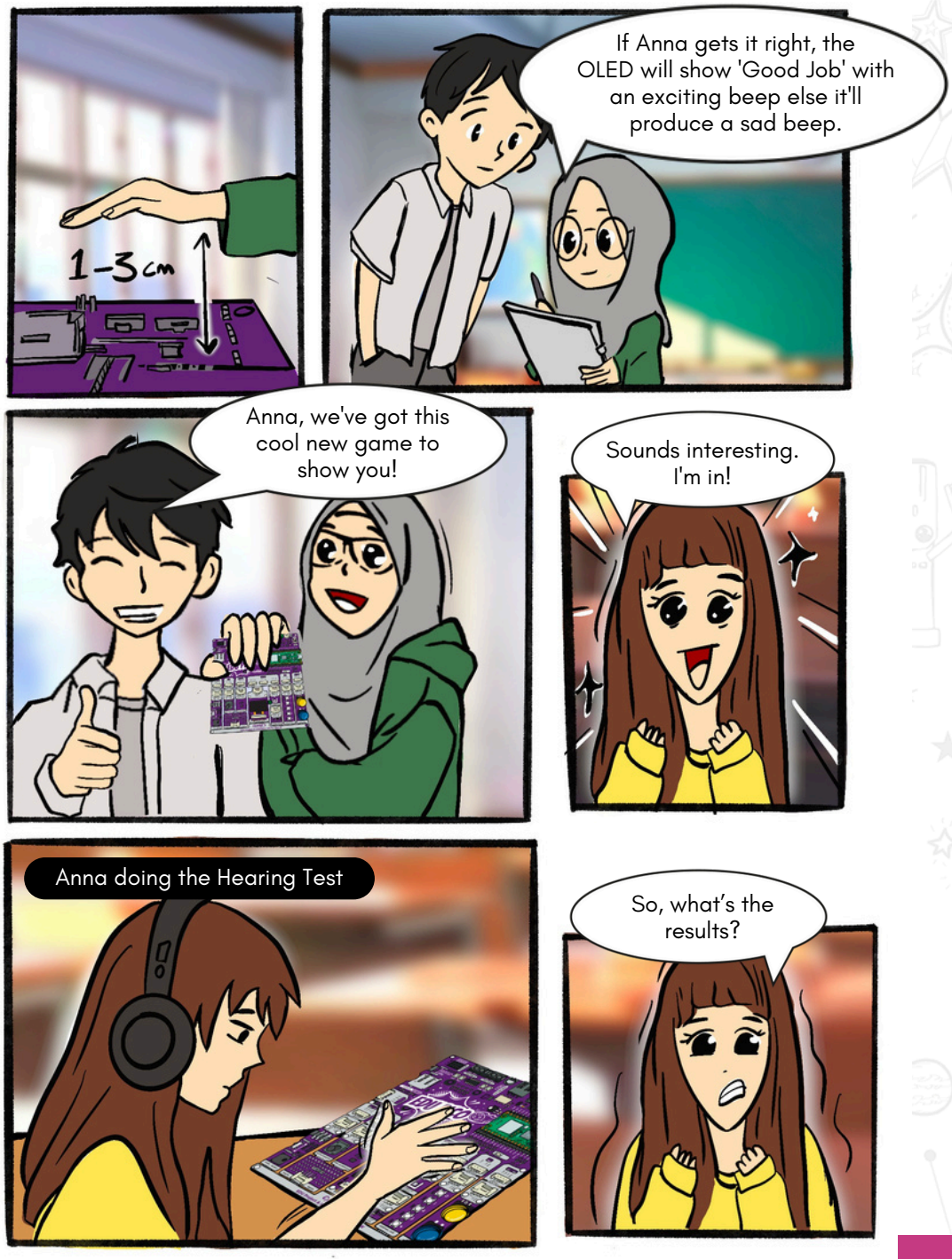


PROLOGUE: GESTURE REACTION GAME

PROLOGUE: GESTURE REACTION GAME

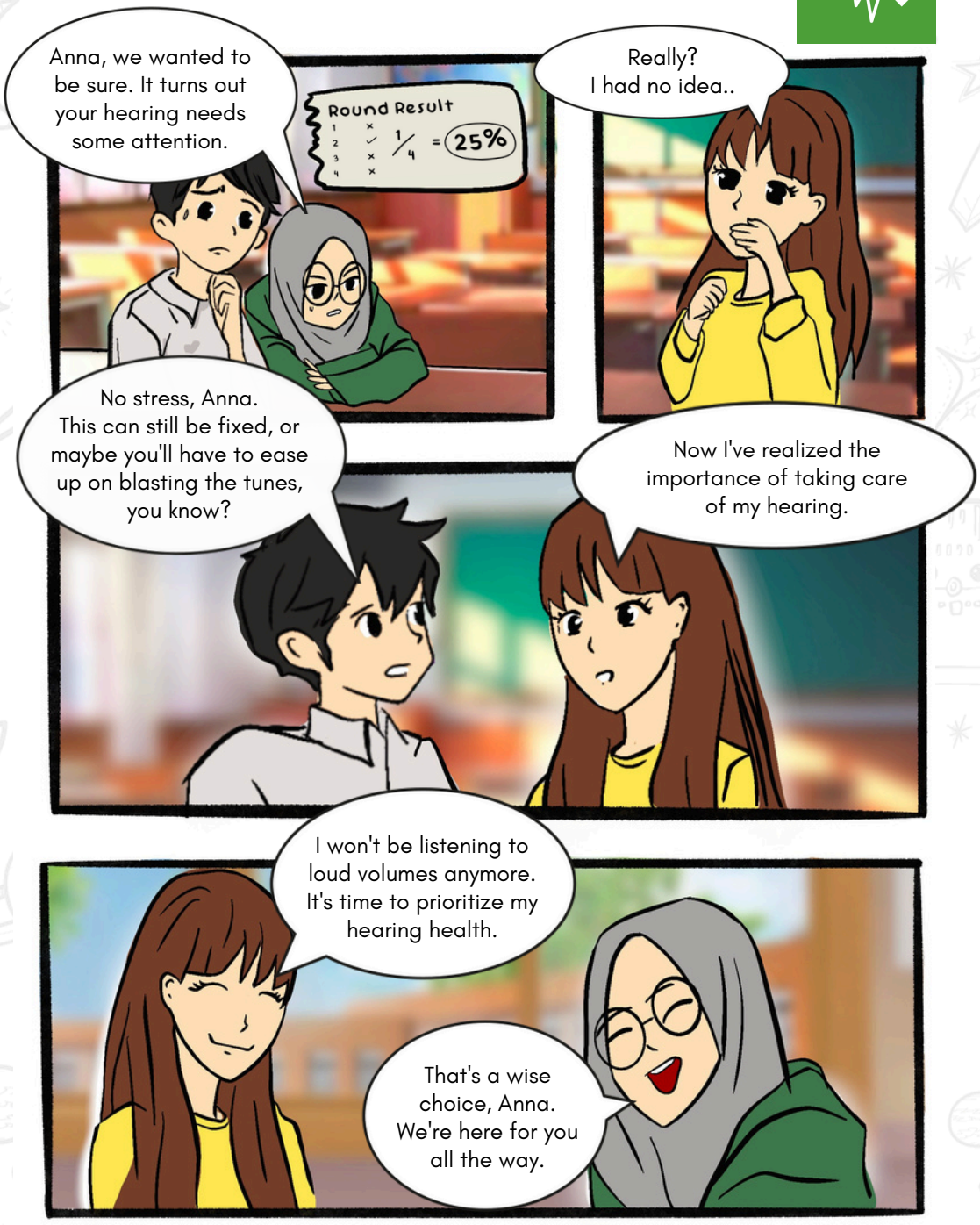
3 GOOD HEALTH AND WELL-BEING

PROLOGUE: GESTURE REACTION GAME



PROLOGUE: GESTURE REACTION GAME

3 GOOD HEALTH AND WELL-BEING



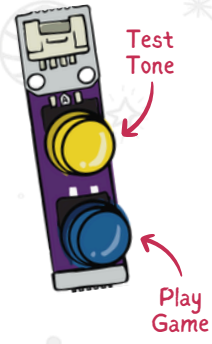
Gesture Reaction Game

Do-Re-Mi-Fa Arcade

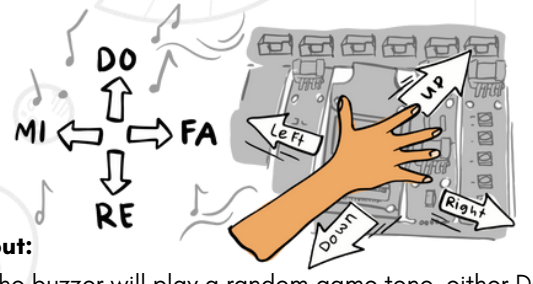
Let's build our first arcade game. But this won't be your typical game, instead, this is going to help players improve their hand-eye coordination. In this project, we'll learn to integrate OLED display, buzzer, and APDS9960 gesture sensor, so be prepared for a slight increase in difficulty. Ready? Let's get started!

How Does This Activity Work?

- **Libraries:** board, digitalio, time, simpleio, busio, adafruit_apds9960 adafruit_ssd1306, random, font5x8.bin.
- **OLED & APDS9960 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 "Go" countdown.
  - The player has to react to the random tone with either UP, DOWN, LEFT or RIGHT hand gestures.



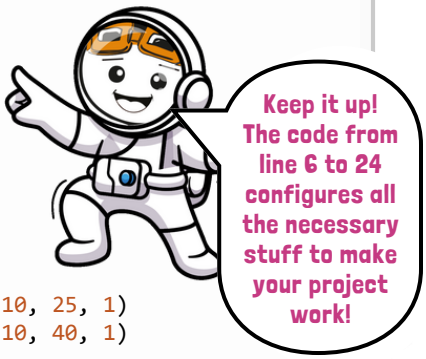
- **Output:**
  - The buzzer will play a random game tone, either DO, RE, MI, or FA.
  - The OLED will print "Try again" with a sad beep if the player gives a wrong answer.
  - Getting the correct tone gesture will produce an exciting beep with a "Good Job" displayed on the OLED.

Code

PROJECT_HEARING_REACTION_GAME.py

```
1  import board, digitalio, time, simpleio, busio
2  from adafruit_apds9960.apds9960 import APDS9960
3  import adafruit_ssd1306
4  import random
5
6  FREQ_DO = 261
7  FREQ_RE = 293
8  FREQ_MI = 329
9  FREQ_FA = 349
10
11 tone_map = {1: FREQ_DO, 2: FREQ_RE, 3: FREQ_MI, 4: FREQ_FA}
12
13 buzzer = board.GP21
14 button_tone = digitalio.DigitalInOut(board.GP0)
15 button_start = digitalio.DigitalInOut(board.GP1)
16 button_tone.direction = digitalio.Direction.INPUT
17 button_start.direction = digitalio.Direction.INPUT
18
19 i2c = busio.I2C(board.GP5, board.GP4)
20 apds = APDS9960(i2c)
21 apds.enable_proximity = True
22 apds.enable_gesture = True
23
24 oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
25
26 def gamestart():
27     oled.fill(0)
28     oled.text("Ready", 50, 25, 1)
29     oled.show()
30     time.sleep(0.5)
31     oled.text("Go!", 50, 40, 1)
32     oled.show()
33
34 while True:
35     oled.fill(0)
36     oled.text("Button A: Test Tone", 10, 25, 1)
37     oled.text("Button B: Play Game", 10, 40, 1)
38     oled.show()
39
40     if not button_tone.value:
41         oled.fill(0)
42         oled.text("Playing Test Tone", 15, 35, 1)
43         oled.show()
44         for tone_freq in (FREQ_DO, FREQ_RE, FREQ_MI, FREQ_FA):
45             simpleio.tone(buzzer, tone_freq, 0.3)
```

Single Tab



```
47      if not button_start.value:
48          gamestart()
49          tone_code = random.randint(1, 4)
50          selected_tone = tone_map[tone_code]
51          simpleio.tone(buzzer, selected_tone, 1)
52
53  Double Tab  while True:
54              gesture_value = apds.gesture()
55              oled.fill(0)
56              if gesture_value == tone_code:
57                  oled.text("Good Job!", 35, 30, 1)
58                  oled.show()
59                  for i in range(3):
60                      simpleio.tone(buzzer, 1100, 0.1)
61                      simpleio.tone(buzzer, 0, 0.1)
62                  break
63              elif gesture_value != 0:
64                  oled.text("Try Again!", 35, 30, 1)
65                  oled.show()
66                  for i in range(3):
67                      simpleio.tone(buzzer, 100, 0.1)
68                      simpleio.tone(buzzer, 0, 0.1)
```

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

What the Code Does

Libraries

```
4  import random
```

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.

Constant Variable and Dictionary

```
6  FREQ_DO = 261
7  FREQ_RE = 293
8  FREQ_MI = 329
9  FREQ_FA = 349
10
11 tone_map = {1: FREQ_DO, 2: FREQ_RE, 3: FREQ_MI, 4: FREQ_FA}
```

Line 6 - 9: Define each **DO RE MI FA** musical tone as a constant variable.
Line 11: Create a dictionary and assign numbers from **1** to **4** to their corresponding musical tones where each frequency has already been defined in line **6** to **9**.

Define a Custom Function

gamestart() function

```
26 def gamestart():
27     oled.fill(0)
28     oled.text("Ready", 50, 25, 1)
29     oled.show()
30     time.sleep(1)
31     oled.text("Go!", 50, 40, 1)
32     oled.show()
```

Line 26 - 32: A function that initializes and displays a "Ready" message on the OLED display followed by a "Go!" message after a brief delay.

Enter a Continuous Loop

Main Loop

```
34 while True:
35     oled.fill(0)
36     oled.text("Button A: Test Tone", 10, 25, 1)
37     oled.text("Button B: Play Game", 10, 40, 1)
38     oled.show()
```

Line 35: **oled.fill(0)** clears the OLED display by filling it with black (**0**).
Line 36 - 37: **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 38: **oled.show()** updates the OLED display to make the text visible.

button_tone (Button A)

```
40     if not button_tone.value:
41         oled.fill(0)
42         oled.text("Playing Test Tone", 15, 35, 1)
43         oled.show()
```

Line 40 - 43: 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".

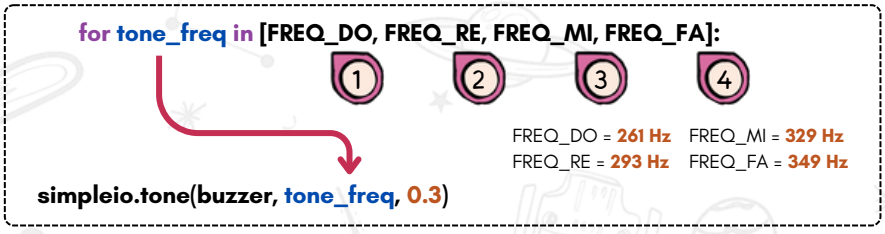
button_tone (Button A)

```
44          for tone_freq in (FREQ_DO, FREQ_RE, FREQ_MI, FREQ_FA):
45              simpleio.tone(buzzer, tone_freq, 0.3)
```

Line 44 - 45: Starts a loop that iterates through a list of musical tone frequencies **DO RE MI FA** 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 the example below, we have a list of 4 musical tones). This also means that the loop will iterate 4 times.



During the first iteration, **tone_freq** will take on the value **FREQ_DO** = **261 Hz**. To put it simply, the values in the code should look like this in the first iteration, [simpleio.tone(buzzer, **261**, **0.3**)]. Then the second iteration, with **FREQ_RE** = **293 Hz**, third with **FREQ_MI** = **329 Hz**, and lastly **FREQ_FA** = **349 Hz**.

Check if Start Button (Button B) is Pressed

```
47      if not button_start.value:
48          gamestart()
49          tone_code = random.randint(1, 4)
50          selected_tone = tone_map[tone_code]
51          simpleio.tone(buzzer, selected_tone, 1)
```

Line 47: Checks if Button B is pressed. If it's pressed, it calls the **gamestart()** function (defined at line 26 - 32) to display "Ready" and "Go!" messages.

Line 49: Stores a randomly generated number ranging from **1** to **4** in variable **random_tone_code**.

Line 50 - 51: **tone_map** array values were defined at Line 11. The code selects a single frequency from **tone_map** array based on the random **tone_code** value and plays a corresponding frequency. For an example:

selected_tone = tone_map[2] = FREQ_RE = 293

Continuous Game Loop - Correct Gesture

```
53          while True:
54              gesture_value = apds.gesture()
55              oled.fill(0)
56              if gesture_value == tone_code:
57                  oled.text("Good Job!", 35, 30, 1)
58                  oled.show()
59                  for i in range(3):
60                      simpleio.tone(buzzer, 1100, 0.1)
61                      simpleio.tone(buzzer, 0, 0.1)
62                  break
```

Line 53 - 58: This game loop is triggered when Button B is pressed. It continuously checks for gestures detected by the APDS9960 sensor. If the detected **gesture_value** matches the **tone_code** (random number between 1 to 4), the OLED will then display "Good Job!".

Line 59 - 60: Plays a success tone sequence before exiting the for loop. The loop will repeat 3 cycles, running line 60 and 61 code for 3 times.

Gesture sensor value	Gesture direction	Tone assigned
0	No gesture detected	-
1	Up	Do
2	Down	Re
3	Left	Mi
4	Right	Fa

Every gesture detected by the APDS9960 sensor will produce a value as shown above, for example: "If a random tone code of **2** is generated, the tone it produced will be **RE**, hence the player will have to swipe **Down** on the gesture sensor to get the correct answer." The program compares the gesture sensor value with the tone code to validate whether the answer provided by the player is correct.

Continuous Game Loop - Wrong Gesture

```
63              elif gesture_value != 0:
64                  oled.text("Try Again!", 35, 30, 1)
65                  oled.show()
66                  for i in range(3):
67                      simpleio.tone(buzzer, 100, 0.1)
68                      simpleio.tone(buzzer, 0, 0.1)
```

If a non-zero gesture (other than the expected one) is detected, it displays "Try Again!" and plays a failure tone sequence through a for loop of range(3).

WHAT'S NEXT?

Pitch Ear Training

One effective way to develop your ear for recognizing notes is through pitch ear training. Simply play the same note repeatedly, and to make it more challenging, modify the tone for each repetition based on the table below. The closer each tone is, the more difficult it becomes for the player to distinguish.

Difficulty	Tone Frequencies (Hz)
Easy	DO = 262, MI = 330, SO = 392, TI = 494
Intermediate	DO = 262, RE = 294, MI = 330, FA = 350

Turn this project into a melody pattern recognition game by adding 2 tones for each gesture!

Hearing Pure-tone Test

We tend to ignore our hearing health until it's too late. This project will allow us to perform a common hearing test known as the pure-tone test. Here's how we run the test:

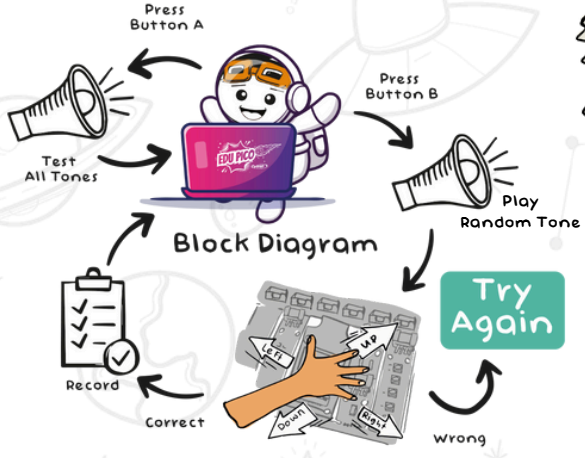
Step 1: Locate a quiet spot to perform the test (ideally in a soundproof room).

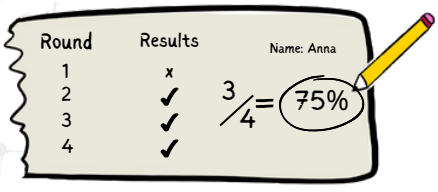
Step 2: Press button A to test play the tone, and button B to produce a random tone.

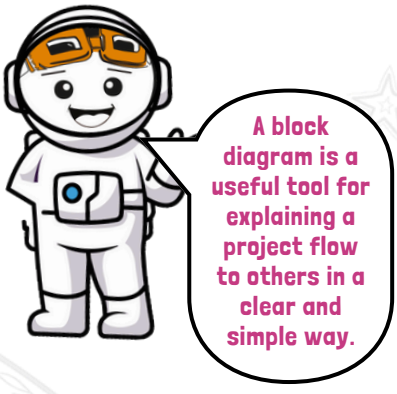
Step 3: After hearing the tone, refer to the gesture directory to decide which correct hand gesture you should swipe above the sensor.

Step 4: Record your result on each turn to calculate your hearing accuracy as shown below.

Step 5: Repeat steps 3 - 5 for 10 rounds minimum.







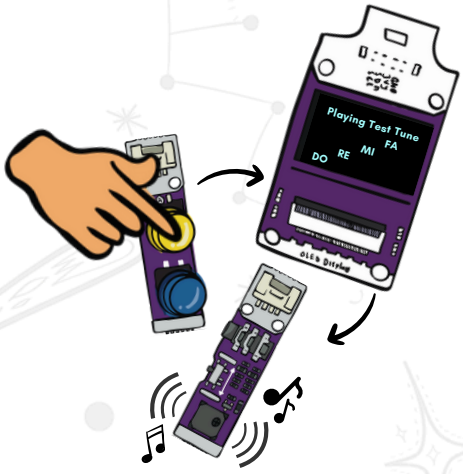
A block diagram is a useful tool for explaining a project flow to others in a clear and simple way.

Disclaimer: This hearing test is intended for informational and educational purposes only. It is not a medical diagnosis, and the results obtained from this test should not be considered as such.

CHALLENGES

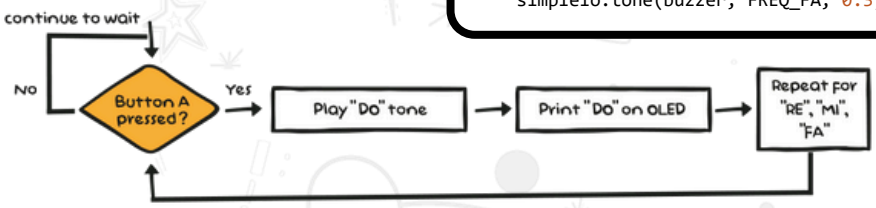
#1 "DO RE MI FA" TONE TESTER

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)
```



#2 - PLAYER'S EXPERIENCE

After playing for a few rounds, it can become frustrating if you don't know which gesture direction represents which tone. To enhance the gaming experience, consider implementing a straightforward gesture directory to guide the player while they guess the gesture as shown in the image below.

QUICK TIPS

```
while True:
    gesture = apds.gesture()
    oled.fill(0)
    oled.text("DO:Up", 50, 25, 1)
    oled.text("RE:Down", 45, 55, 1)
    oled.text("MI:Left", 5, 40, 1)
    oled.text("FA:Right", 75, 40, 1)
    oled.show()
```

CHAPTER 4

Colour Detection Game

RGB LEDs & Colour Sensor

In this chapter, we will be exploring these two incredible components: RGB LEDs and the APDS9960 colour sensor module with EDU PICO.

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

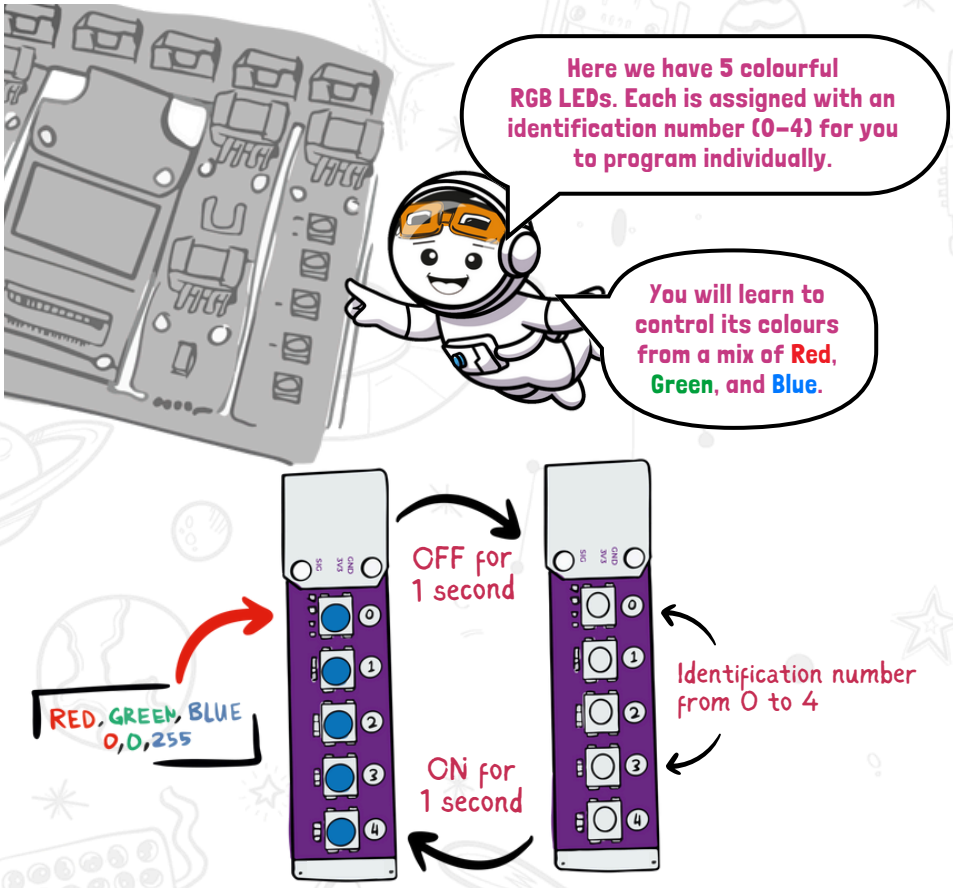
We'll start by understanding what RGB LEDs are and how they can be programmed to produce a mesmerizing rainbow of colours. Then, we'll introduce you to the APDS9960 colour sensor, a powerful tool that can "see" colours just like you do.

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 (blue colour) and off, 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. It is a must to have to complete this activity.

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** (***GP14***), **num_pixels** to **5**, and the brightness parameter to **0.2** (20% brightness).

Enter a Continuous Loop

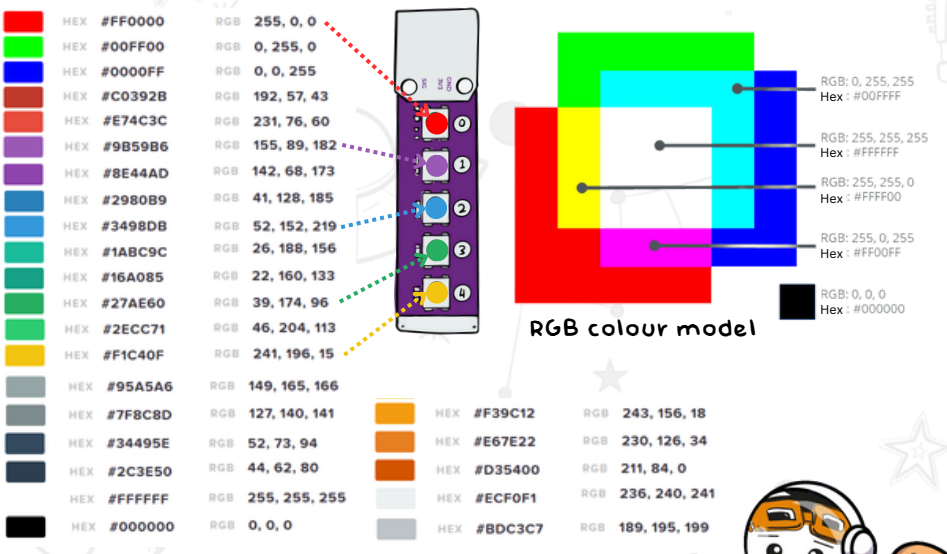
Light Up Pixel Continuously

```
7  while True:
8      pixels.fill((0,0,255))
9      time.sleep(1)
10     pixels.fill((0,0,0))
11     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 integrated 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.

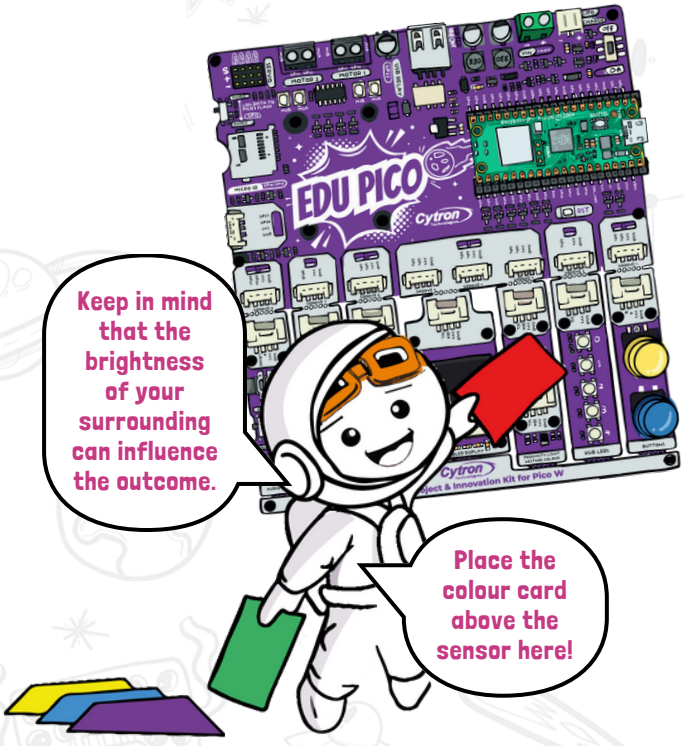


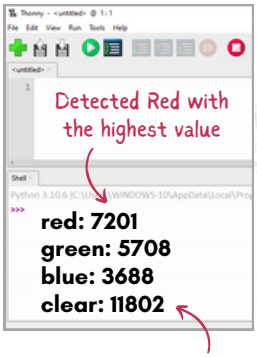
Introduction to Colour Sensor

In the previous chapter, we explored the APDS9960 sensor, which was initially used to detect gestures. However, you might be surprised to learn that this sensor is not limited to just detecting gestures. It can also be used to identify and distinguish different colours!

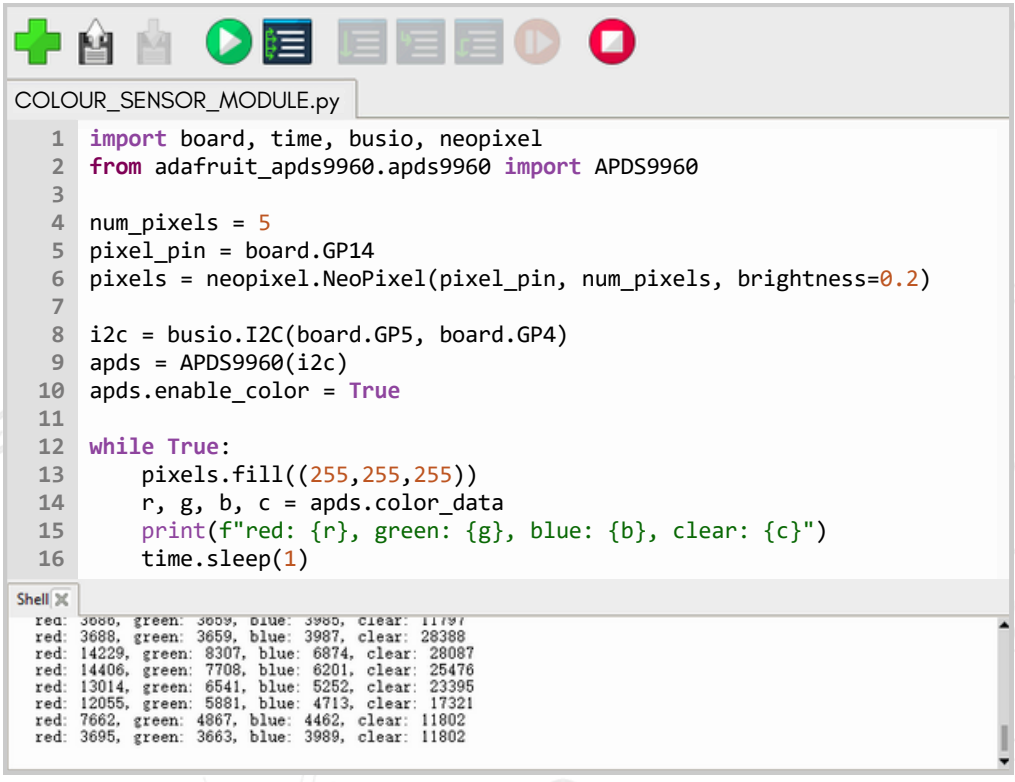
How Does This Activity Work?

- **Libraries:** board, time, busio, neopixel, adafruit_apds9960.
- **RGB LEDs Configuration:** num_pixels = **5**, pixel_pin = ***GP14***.
- **APDS9960 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.





Code



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

What the Code Does

Libraries

```
1 import board, time, busio, neopixel
2 from adafruit_apds9960.apds9960 import APDS9960
```

adafruit_apds9960.apds9960 provides functionality to work with the APDS9960 colour sensor.

RGB LEDs and APDS9960 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  apds = APDS9960(i2c)
10 apds.enable_color = True
```

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 - 9: The script initializes the I2C communication using pins ***GP5*** and ***GP4***. An instance of the APDS9960 colour sensor is created.

Line 10: This line enables colour detection on the APDS9960 sensor.

Main Loop

```
12 while True:
13     pixels.fill((255,255,255))
14     r, g, b, c = apds.color_data
15     print(f"red: {r}, green: {g}, blue: {b}, clear: {c}")
16     time.sleep(1)
```

Line 13: Light up the RGB LEDs in white. By doing so, it creates a bright surrounding next to the APDS9960 sensor which will allow the sensor to detect the object's colour clearly and more accurately.

Line 14: **apds.colour_data** retrieves colour information from the APDS9960 sensor, providing separate variables for red (r), green (g), blue (b), and clear (c) 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 {c} are the placeholders within the f-string. They are used to indicate where the values of the variables r, g, b and, c should be inserted within the text.

MINI ACTIVITY

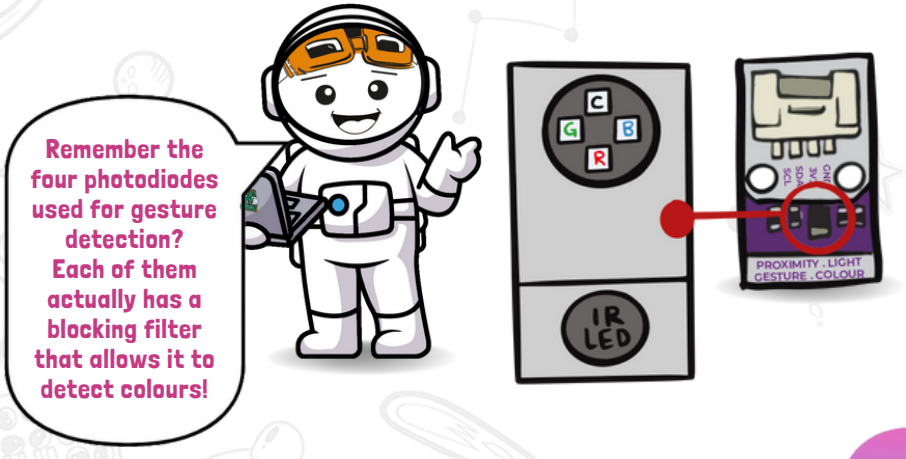
Modify the code to be able to compare and identify between red, green, or blue by using the **if..elif..else** statement. While you're at it, make use of the AND logical operator to determine the most prominent colour in the data.

QUICK TIPS

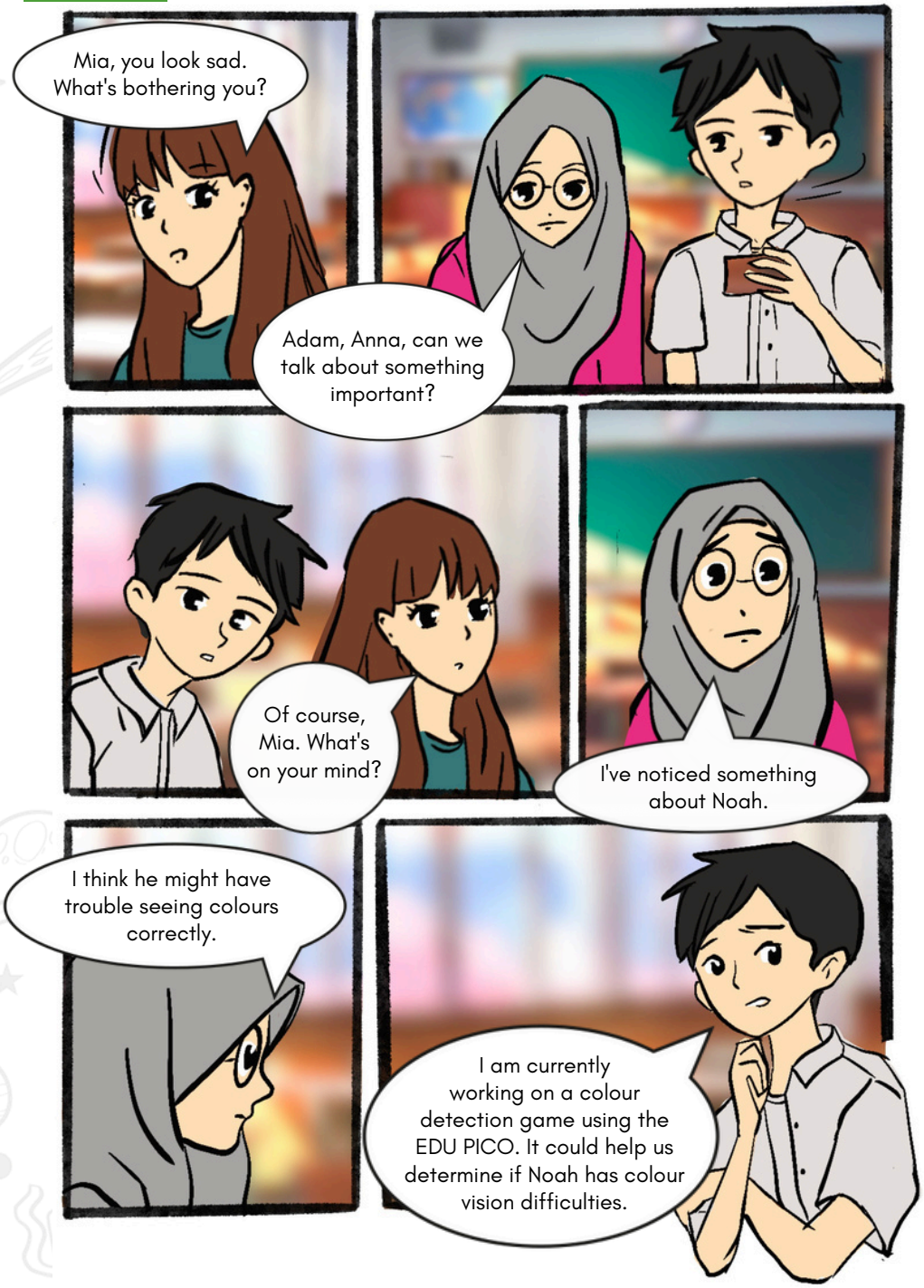
```
while True:
    pixels.fill((255,255,255))
    r, g, b, c = apds.color_data
    print(f"red: {r}, green: {g}, blue: {b}, clear: {c}")
    if r > g and r > b:
        print("Red Detected")
    elif g > r and g > b:
        print("Green Detected")
    else:
        print("Blue Detected")
    time.sleep(1)
```

THE MORE YOU KNOW

Colour sensing with the APDS9960 has diverse applications. For instance, it can be used in colour sorting machines, where objects are categorized based on their colours. Additionally, the colour data from the sensor can also be used to calculate ambient light levels (i.e. Lux), which will be covered in a later chapter.

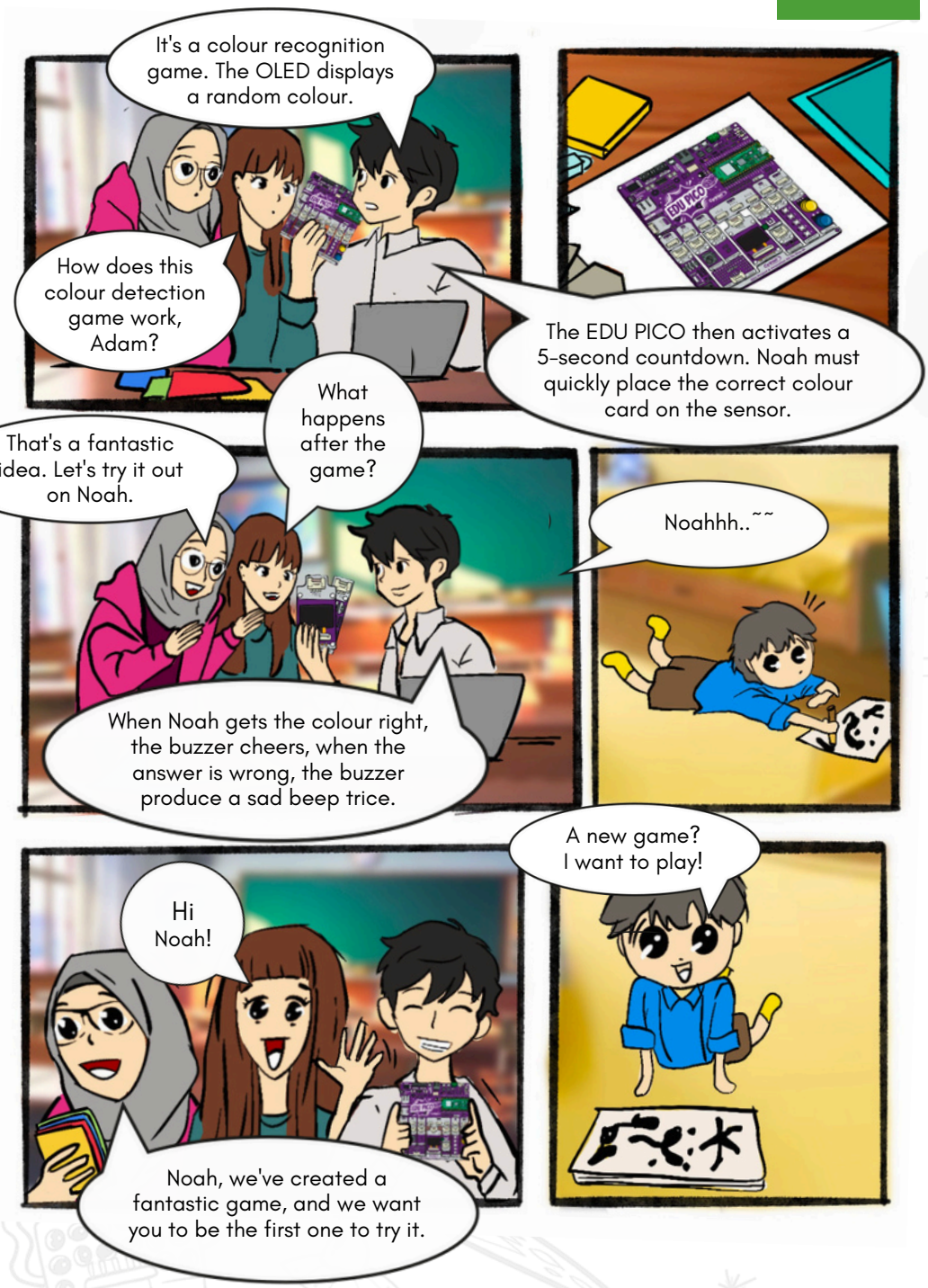


PROLOGUE: COLOUR DETECTION GAME



PROLOGUE: COLOUR DETECTION GAME

3 GOOD HEALTH AND WELL-BEING



PROLOGUE: COLOUR DETECTION GAME

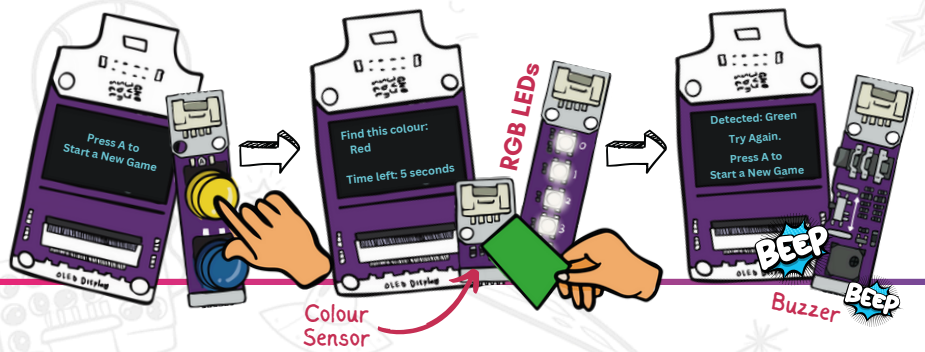
Colour Detection Game

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_apds9960, font5x8.bin.
- **OLED & APDS9960 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 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.
  - 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_apds9960.apds9960 import APDS9960
import adafruit_ssd1306

i2c = busio.I2C(board.GP5, board.GP4)
apds = APDS9960(i2c)
apds.enable_color = True
apds.enable_proximity = True

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(f"{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(f"{countdown}", 70, 50, 1)
        oled.show()
        time.sleep(1)

    if apds.proximity < 10:
        oled.fill(0)
        oled.text("No object detected", 10, 15, 1)
        oled.text("Press A to", 35, 35, 1)
        oled.text("Start a New Game", 15, 45, 1)
        oled.show()
        for i in range(3):
            simpleio.tone(buzzer, 100, 0.1)
            simpleio.tone(buzzer, 0, 0.1)
    else:
        r, g, b, c = apds.color_data
        print(f"red: {r}, green: {g}, blue: {b}, clear: {c}")
        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(f"Detected: {detected_colour}", 20, 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_apds9960.apds9960 import APDS9960
3 import adafruit_ssd1306
```

Import a total of 9 libraries for the following project to work. Libraries include **APDS9960** for colour sensor, **adafruit_ssd1306** for OLED display, **neopixel**, and **random** for generating a random number.

Initialize Hardware Components

Initialization

```
4
.
.
19
20 MELODY_NOTE = [523, 659, 784, 0, 659, 784]
21 MELODY_DURATION = [0.12, 0.12, 0.12, 0.1, 0.12, 0.2]
22
23 colours = ["Red", "Green", "Blue", "Yellow", "Purple"]
24
25 oled.fill(0)
26 oled.text("Press A to", 35, 25, 1)
27 oled.text("Start a New Game", 15, 35, 1)
28 oled.show()
```

Line 4 - 19: Initializes APDS9960 sensor, OLED display, buzzer (***GP21***), button A (***GP0***) and RGB LEDs module (***GP14***).

Line 20: 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 21: 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.

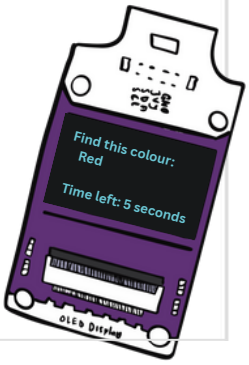
Line 23: The **colours** array contains the names of different colours used in the game.

Line 25 - 28: Produce an initial message to instruct the player to press button A to start a new game.

Enter a Continuous Loop

Prepare the Game

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



Line 31: Clear the RGB LEDs strip (turn off all pixels).

Line 32: Wait for **button_start** (Button A) to be pressed to start the game.

Line 34: Clear the OLED display to prepare for the game.

Line 36: Randomly selects a colour from the **colours** array defined at line 23.

Line 38: Prints the randomly selected colour (stored in the **random_colour** variable) on the OLED display.

Line 41: Light up all RGB LEDs in white to indicate the start of the game.

Line 42: Prints "Time left: seconds", where the blank space will be replaced with the seconds value in the later code.

Starting the Game - The Countdown

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

Line 44: The for loop iterates through a range of numbers from **5** down to **0**.

Line 45: 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 46: This line displays the current value of the countdown as text on the OLED.

Line 48: This line introduces a **1** second interval to synchronize with the countdown.

Check the Proximity Value

Detect Object Presence

```
50      if apds.proximity < 10:
51          oled.fill(0)
52          oled.text("No object detected", 10, 15, 1)
53          oled.text("Press A to", 35, 35, 1)
54          oled.text("Start a New Game", 15, 45, 1)
55          oled.show()
56          for i in range(3):
57              simpleio.tone(buzzer, 100, 0.1)
58              simpleio.tone(buzzer, 0, 0.1)
```

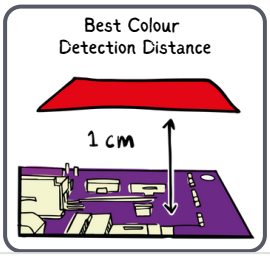
Line 50: If the proximity value is less than **10**, it indicates that there is no object detected (e.g: colour card) above the proximity sensor.

Line 51 - 58: If no object is detected, the OLED prints the message shown in the illustration above, followed by 3 beeps on the buzzer indicating the input on the APDS9960 sensor is invalid.

Check the Colour Data Value

Detect Colour

```
59      else:
60          r, g, b, c = apds.color_data
61          print(f"red: {r}, green: {g}, blue: {b}, clear: {c}")
62          if r > g and r > b:
63              if g > b:
64                  detected_colour = "Yellow"
65              else:
66                  detected_colour = "Red"
67          elif b > r and b > g:
68              if r > g:
69                  detected_colour = "Purple"
70              else:
71                  detected_colour = "Blue"
72          elif g > r and g > b:
73              detected_colour = "Green"
```

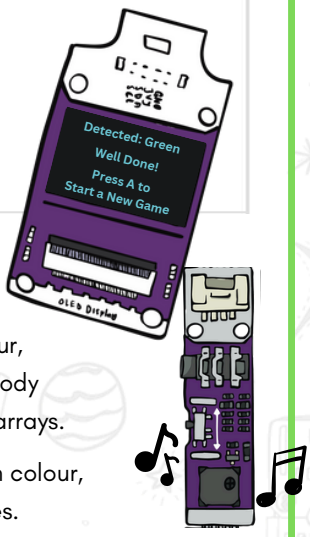


Line 60 - 61: Read colour data and print the red, green, blue, and clear values to the shell console.

Line 62 - 73: The following lines determine the detected colour based on the RGB values by comparing them to one another.

Compare Colour

```
79          if detected_colour == random_colour:
80              oled.text("Well Done!", 35, 25, 1)
81              oled.text("Press A to", 35, 45, 1)
82              oled.text("Start a New Game", 15, 55, 1)
83              oled.show()
84              for i in range(len(MELODY_NOTE)):
85                  simpleio.tone(buzzer, MELODY_NOTE[i], MELODY_DURATION[i])
86          else:
87              oled.text("Try Again.", 35, 25, 1)
88              oled.text("Press A to", 35, 45, 1)
89              oled.text("Start a New Game", 15, 55, 1)
90              oled.show()
91              for i in range(3):
92                  simpleio.tone(buzzer, 100, 0.1)
93                  simpleio.tone(buzzer, 0, 0.1)
```



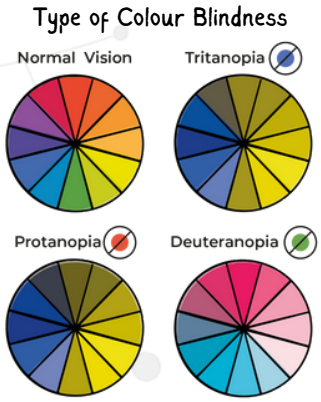
Line 79: This condition checks if the detected colour is equal to the random colour chosen at the beginning of the game.

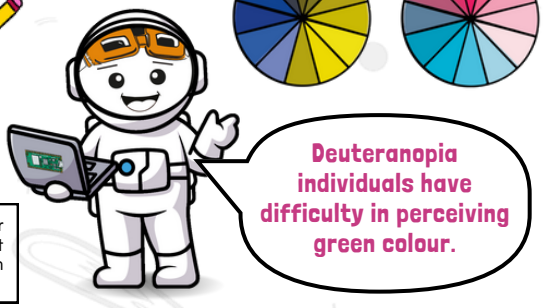
Line 80 - 85: If the detected colour matches the random colour, the OLED prints the text shown on the right, followed by a melody iterating through **MELODY_NOTE** and **MELODY_DURATION** arrays.

Line 86 - 93: If the detected colour doesn't match the random colour, the OLED will print "Try Again" and the buzzer will beep 3 times.

WHAT'S NEXT?

Bring awareness to colour blindness by letting your friends experience the project! If the player is colour blind, this project will allow them to vaguely identify the type of colour blindness they are in. Collect the results as shown below. The more mistakes made for either green and red, or blue and yellow, the easier it is to identify which type of colour blindness category you're in.





Disclaimer: This colour blindness test is intended for informational and educational purposes only. It is not a medical diagnosis, and the results obtained from this test should not be considered as such.

CHALLENGES

#1 IMPROVE GAMING EXPERIENCE

It can be frustrating as a player to receive three identical colours in a row. Unfortunately, using the random function means there's still a chance that will happen.

We can include a "check-code" before generating a random colour to eliminate the possibility of repeating the same colours. While we are at it, let's build a function to generate the random colour, this will help us keep the code neat too.

```
def get_random_colour():
    colours = ["Red", "Green", "Blue", "Yellow", "Purple"]
    x = random.choice(colours)
    while x == prev_colour:
        x = random.choice(colours)
    return x

while True:
    pixels.fill((0,0,0))
    while button_start.value:
        pass
    oled.fill(0)
    oled.show()
    random_colour = get_random_colour()
    prev_colour = random_colour
```

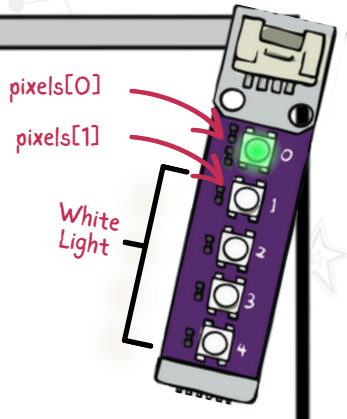
QUICK TIPS

#2 COLOUR HINT INDICATOR

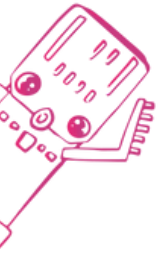
You probably wonder what else can be done for our fellow gamers. For one, we can improve the game by providing hints to the player. Here's how it works, the flow of the game will still be the same, but instead of lighting up all RGB LEDs in white, the first RGB LEDs ID:0 will light up according to the same random colour. This will allow the user to colour match and increase their chance of getting the right answer too. 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



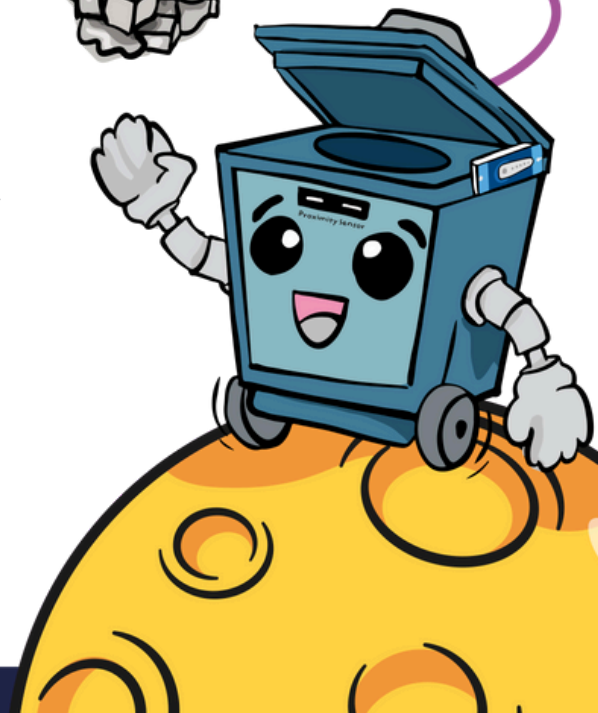
Automated Waste Bin

Servo Motor & Proximity Sensor

This chapter introduces two key components: the servo motor and the APDS9960 proximity sensor. The servo motor is a crucial component in the creation of robotic beings. It enables the positioning of solar panels and the automation of doors, among other applications.

The APDS9960 comes with various built-in functionality, one of them is to measure proximity for detecting nearby objects. It empowers machines to perceive the world around them, much like how our senses do for us. With its ability to detect the proximity of objects, it can create innovative solutions like touchless switches and gesture-controlled devices.

By learning to harness the power of the servo motor and APDS9960, you will be able to gain insight into crafting responsive and intelligent systems.



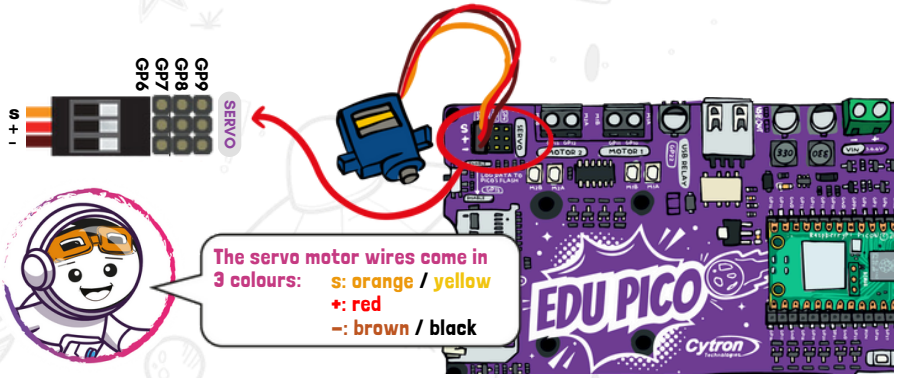
Introduction to Servo Motor

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.

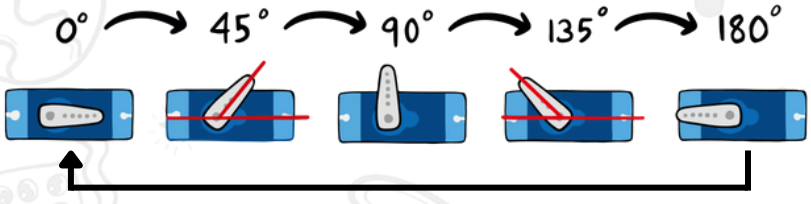
How Does This Activity Work?

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

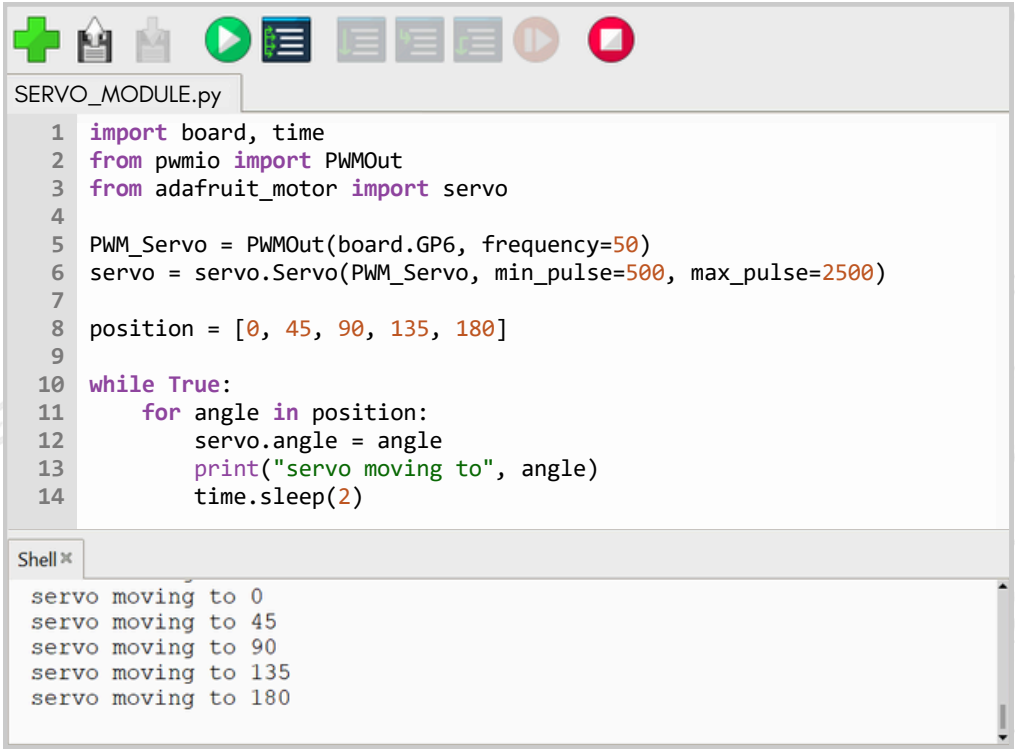
  - ***GP6*** with orange wire connected to S, red to (+) and brown to (-), as shown in the illustration below.



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



Code



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** module 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 code in a shorter amount of time) 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. 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].

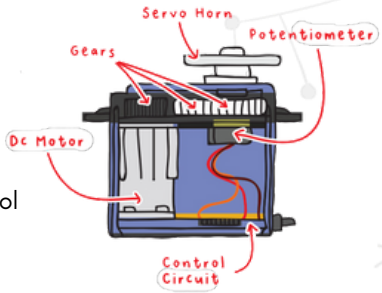
Line 12: Sets the angle of the servo motor to the value stored in the **angle** variable in every iteration; this command also controls the servo motor's position by specifying the angle it should move.

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

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

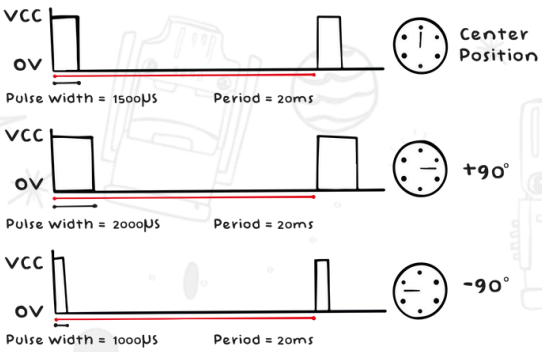
THE MORE YOU KNOW

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 (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)

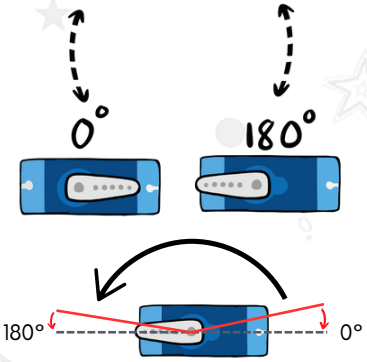
Modern RC servos rely on pulse-width modulation to determine the angle of mechanical rotation. Although a standard RC servo expects a pulse every 20ms (milliseconds), the duration of this pulse can vary significantly across different servos. In EDU PICO's case, we are using a pulse of 20ms where a PWM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

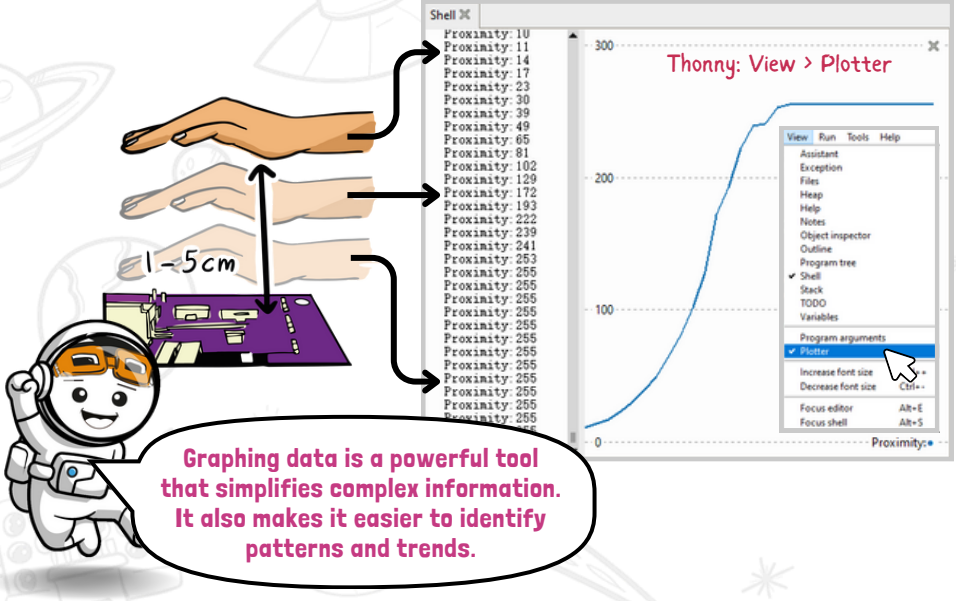


Introduction to Proximity Sensor

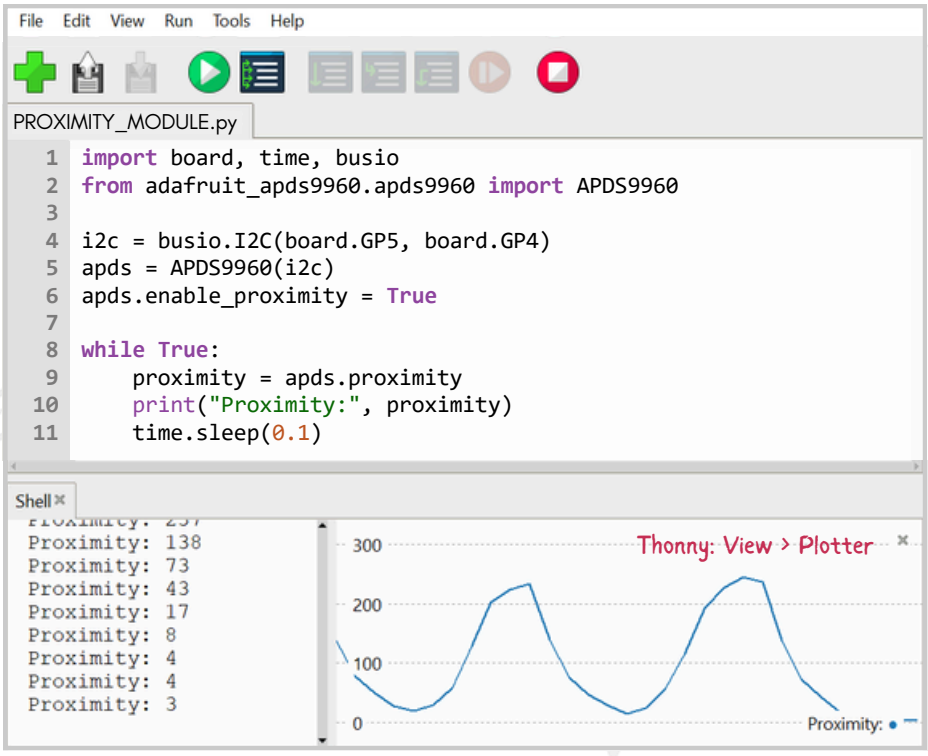
The proximity sensor detects the presence or absence of objects using the reflection of infrared light (IR) without physical contact. They are used in various applications, such as touchless switches in public restrooms, automatic faucets, and smartphones. Proximity sensors are also important in industrial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

How Does This Activity Work?

- **Libraries:** board, time, busio, adafruit_apds9960.
- **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 **0.1** second.
- **Action Required:** Enable the plotter function in Thonny IDE to have a better visual on the proximity value.



Code



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

What the Code Does

APDS9960 Initialization

```
4 i2c = busio.I2C(board.GP5, board.GP4)
5 apds = APDS9960(i2c)
6 apds.enable_proximity = True
```

Line 4 - 5: Initializes the I2C communication bus for the APDS9960 sensor for the user to interact with it. The configuration is similar to when using the gesture and colour sensor.

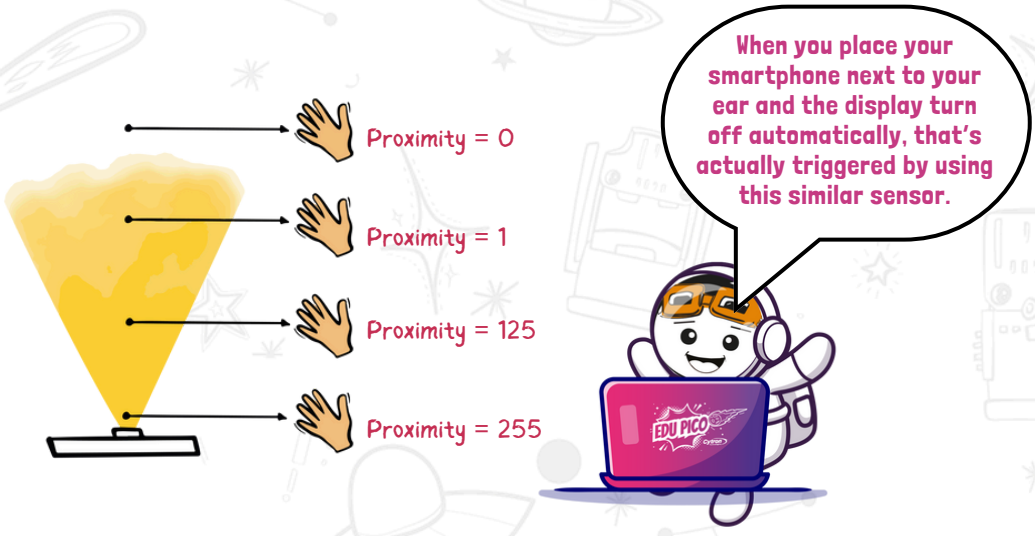
Line 6: Enables the proximity feature of the APDS9960 sensor. It allows the sensor to detect objects or the proximity of objects in front of it.

Main Loop

```
8  while True:
9      proximity = apds.proximity
10     print(f"Proximity:{proximity}")
```

Line 9: Reads the proximity value from the APDS9960 sensor and assigns it to the **proximity** variable. The proximity value represents how close an object is to the sensor. The nearer the object, the greater the value.

Line 10: Prints the current proximity value along with the string "Proximity:" text to the shell console, allowing you to monitor and see the proximity value as it changes.



MINI ACTIVITY

Let's turn this activity into an obstacle-detection beeping device! First, import **simpleio** library so that we can initialize the buzzer's ***GP21*** pin.

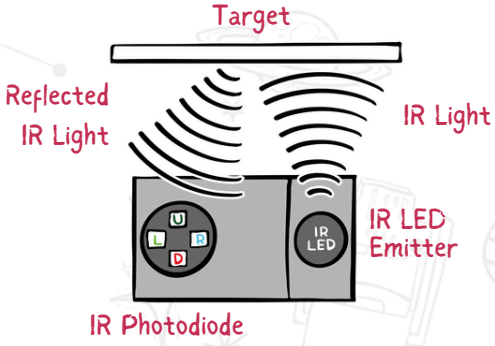
QUICK TIPS

```
while True:
    proximity = apds.proximity
    duration = 1 - proximity / 255
    print(f"Proximity: {proximity}, duration: {duration} sec")
    simpleio.tone(buzzer, 440, 0.1)
    time.sleep(duration)
```

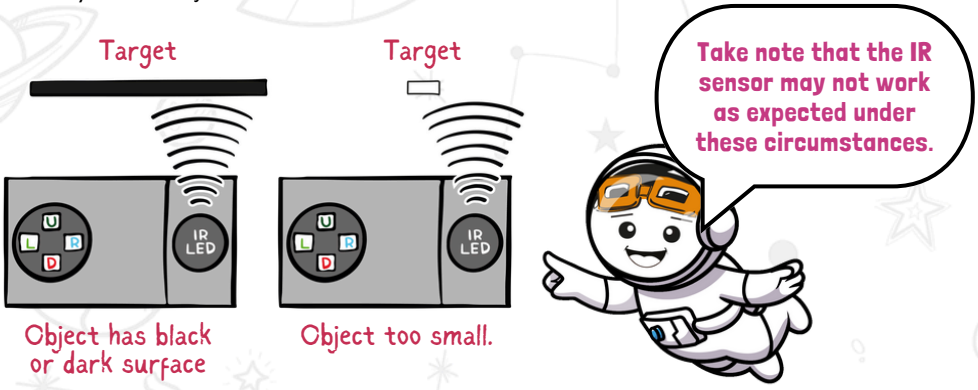
THE MORE YOU KNOW

APDS9960 - Proximity Sensor

The APDS9960 proximity sensor works by emitting and detecting infrared light to determine the distance of an object or obstacle. The sensor includes an IR photodiode that is placed adjacent to the IR LED.

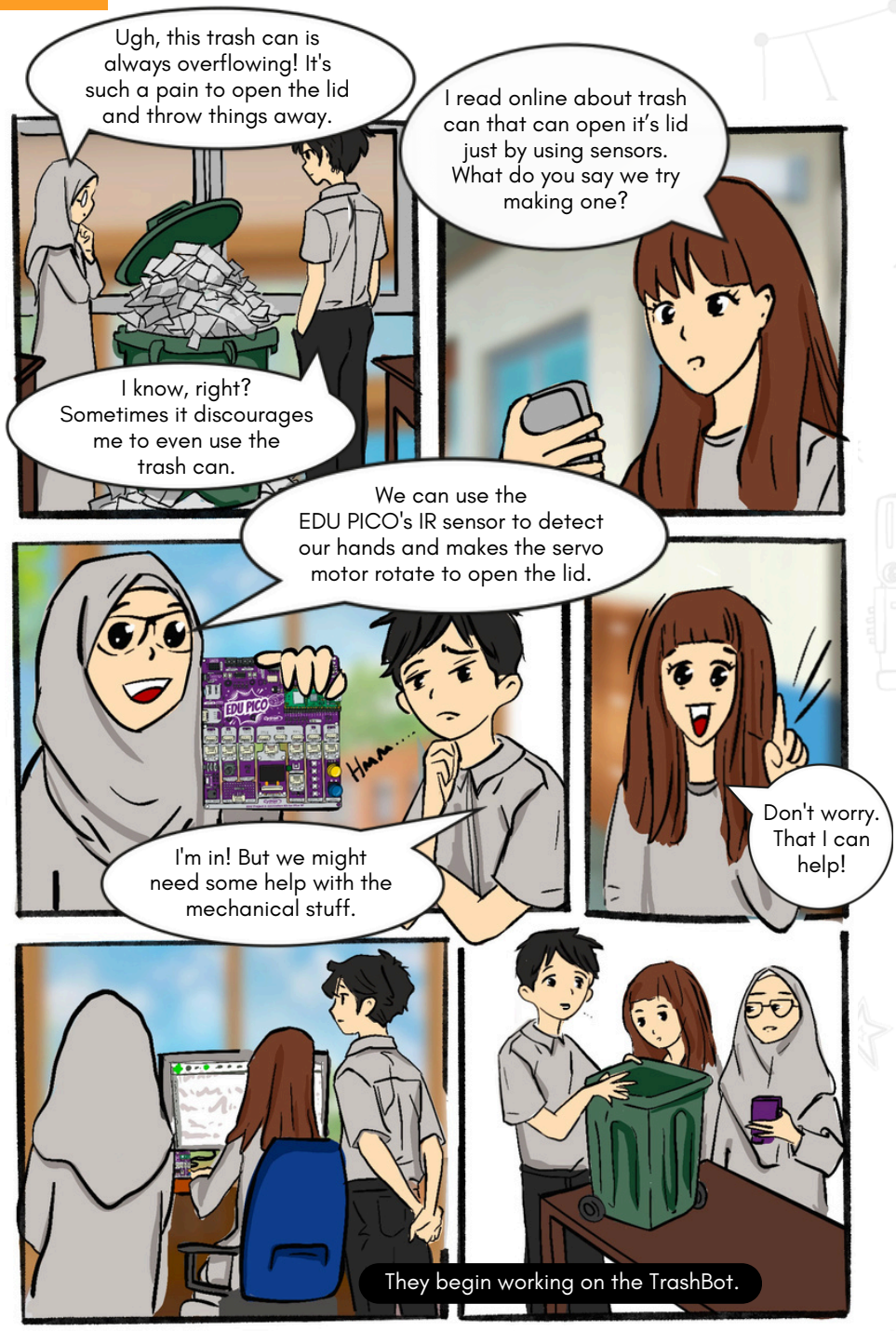


- **IR LED (Emitter):** Active IR sensors consist of an IR LED (Infrared Light Emitting Diode) that emits infrared light (around 950 nanometers) when powered on.
- **IR Photodiode (Receiver):** The IR photodetector (usually known as a photodiode or a phototransistor) is placed adjacent to the IR LED. 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 proximity of the object!



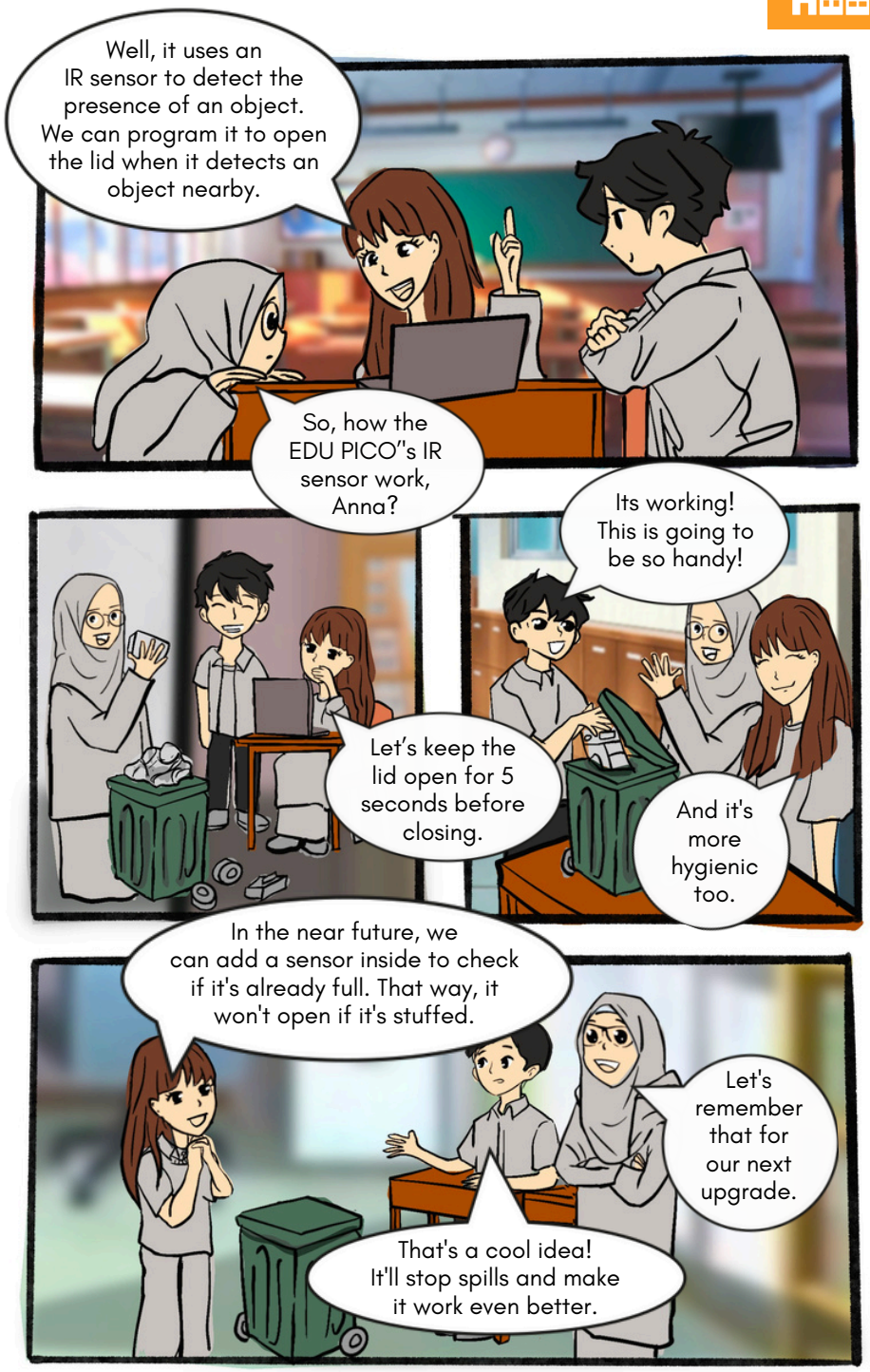
11 SUSTAINABLE CITIES AND COMMUNITIES

PROLOGUE: AUTOMATED WASTE BIN



PROLOGUE: AUTOMATED WASTE BIN

11 SUSTAINABLE CITIES AND COMMUNITIES



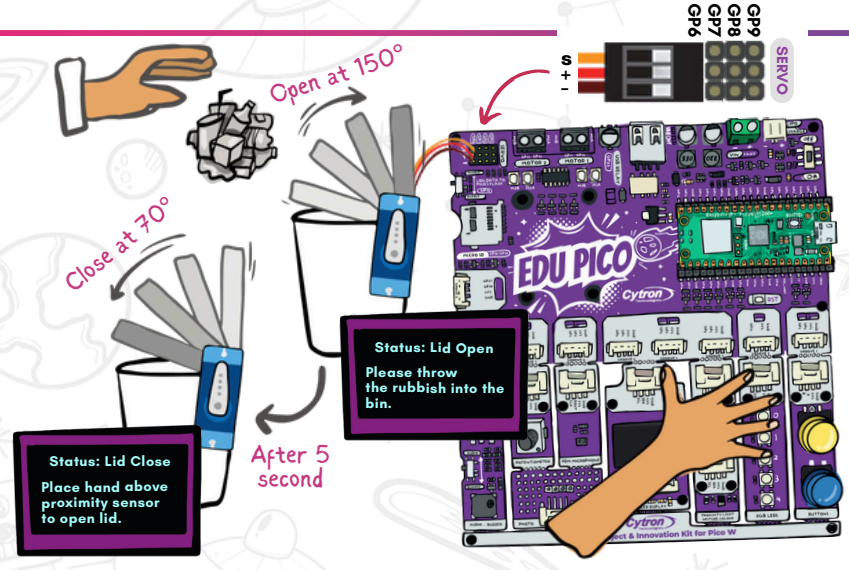
Automated Waste Bin

TrashBot Smart Bin

Time to build your very own TrashBot! Equipped with an IR 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, adafruit_apds9960, 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 **70** to **150** 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
from adafruit_apds9960.apds9960 import APDS9960
from adafruit_motor import servo
from pwmio import PWMOut

i2c = busio.I2C(board.GP5, board.GP4)
apds = APDS9960(i2c)
apds.enable_proximity = True

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 = 70

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()

show_lid_close()

while True:
    proximity = apds.proximity

    if proximity > 20:
        servo.angle = 150
        show_lid_open()
        time.sleep(5)

        servo.angle = 70
        show_lid_close()
```

What the Code Does </>

Import Necessary Libraries

Libraries

```
1 import board, time, busio, adafruit_ssd1306
2 from adafruit_apds9960.apds9960 import APDS9960
3 from adafruit_motor import servo
4 from pwmio import PWMOut
```

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

Line 2: **adafruit_apds9960** is used for controlling the APDS9960 proximity sensor.

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

Initialize Hardware Components

Hardware Initialization

```
6  i2c = busio.I2C(board.GP5, board.GP4)
7  apds = APDS9960(i2c)
8  apds.enable_proximity = True
9
10 oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
11
12 PWM_Servo = PWMOut(board.GP6, frequency=50)
13 servo = servo.Servo(PWM_Servo, min_pulse=500, max_pulse=2500)
14 servo.angle = 70
```

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

Line 7 - 8: Create an instance of the APDS9960 proximity sensor and enable proximity sensing.

Line 10: Initialize SSD1306 OLED display with a resolution of **128**x**64** pixels.

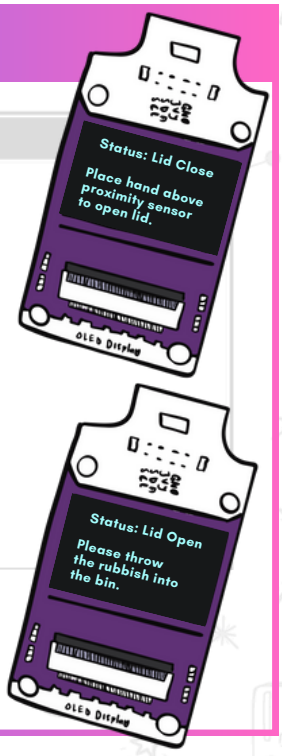
Line 12: Create a PWMOut instance for controlling a servo on GPIO pin ***GP6*** with a PWM frequency of **50 Hz**.

Line 13: Create an instance (**servo**) using the **servo.Servo** class, specifying the **PWMOut** instance and pulse width limits.

Line 14: Initialize the servo motor by rotating it to **70** degrees.

Define a Custom Function

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



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

Enter a Continuous Loop

Main Loop

```
34 while True:
35     proximity = apds.proximity
36
37     if proximity > 20:
38         servo.angle = 150
39         show_lid_open()
40         time.sleep(5)
41
42         servo.angle = 70
43         show_lid_close()
```

Line 35: Read proximity data from the APDS9960 sensor using **apds.proximity** and store it in the **proximity** variable.

Line 37: A proximity value greater than **20** indicates that an object is close to the proximity sensor.

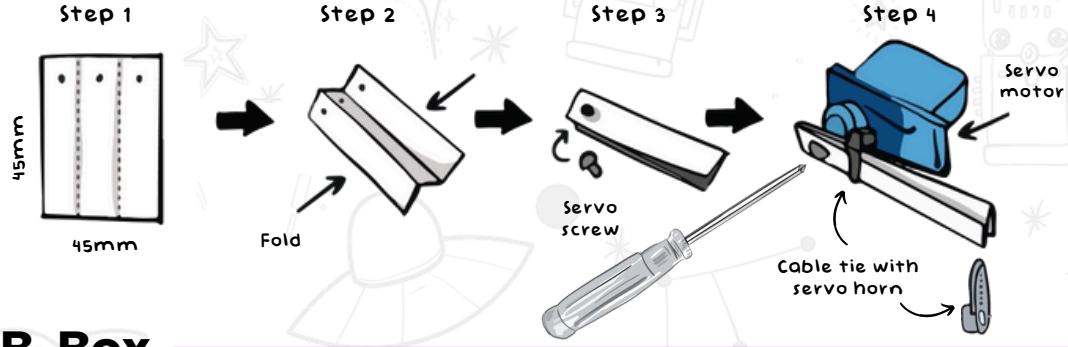
Line 38 - 43: When the proximity sensor detects an object, the servo motor will rotate and remain at **150** degrees for **5** second. After **5** second, the servo motor returns to its original state at **70** degrees, closing the lid of the trash bin.

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 set.

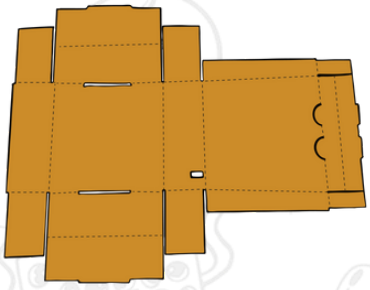
A. Servo Stick



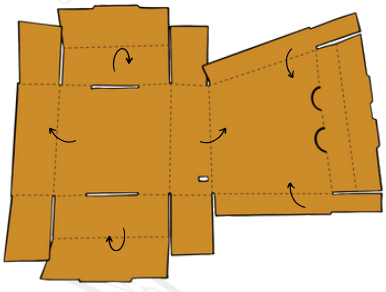
B. Box

Fold the Trashbot box accessory as shown below.

Step 1

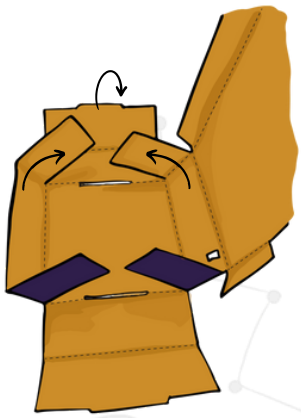


Step 2

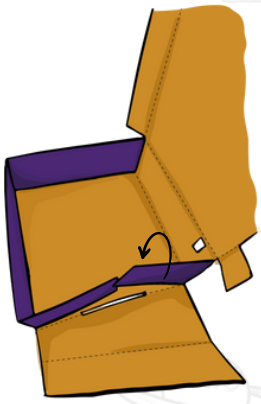


Fold the Trashbot's body.

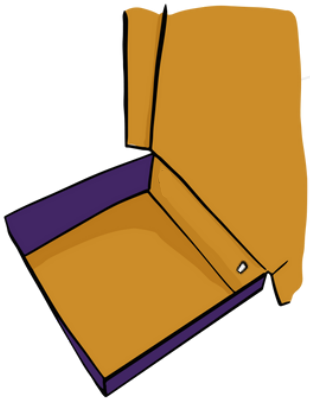
Step 3:



Step 4:

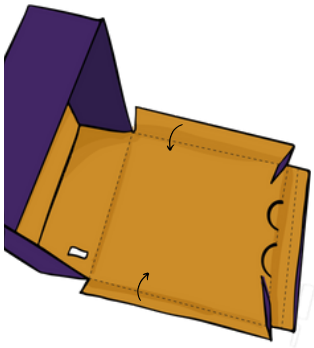


Step 5:

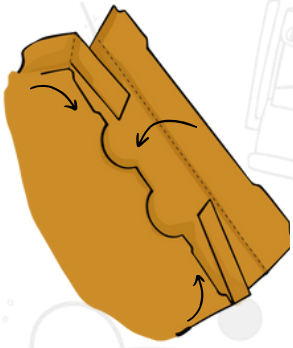


Fold the Trashbot's head.

Step 6:



Step 7:

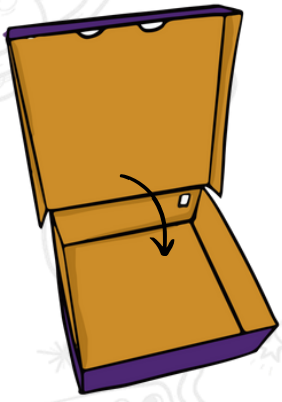


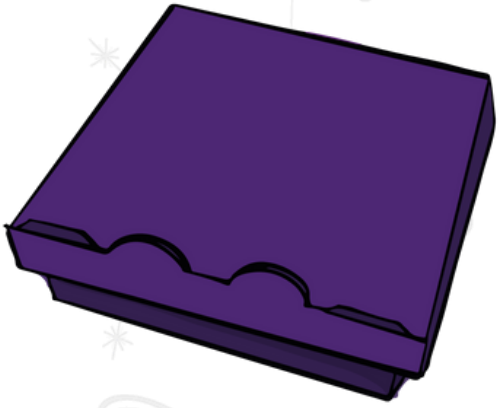
Step 8:

Step 9:

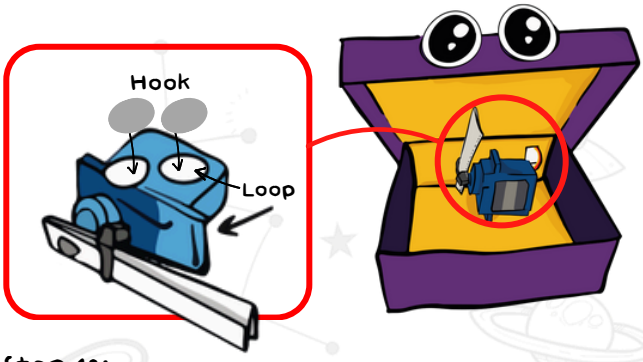
Step 10:

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



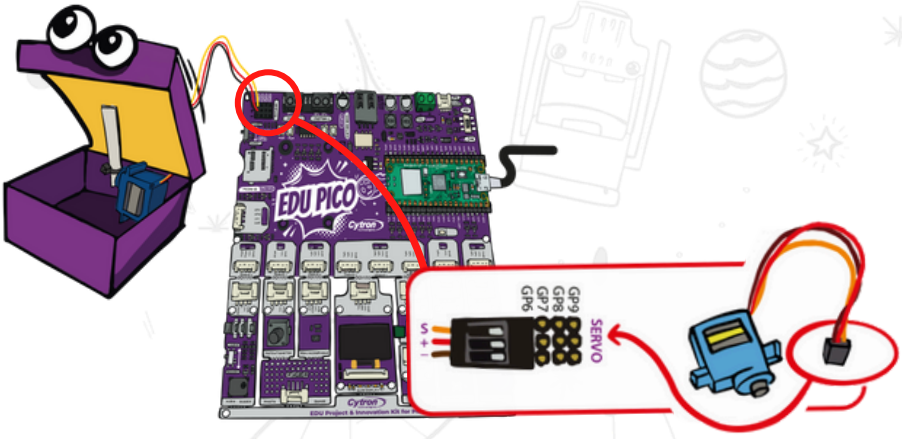


Step 11:
Attach the velcro's hook above the loop and attach the servo motor to the inner back of the Trashbot.

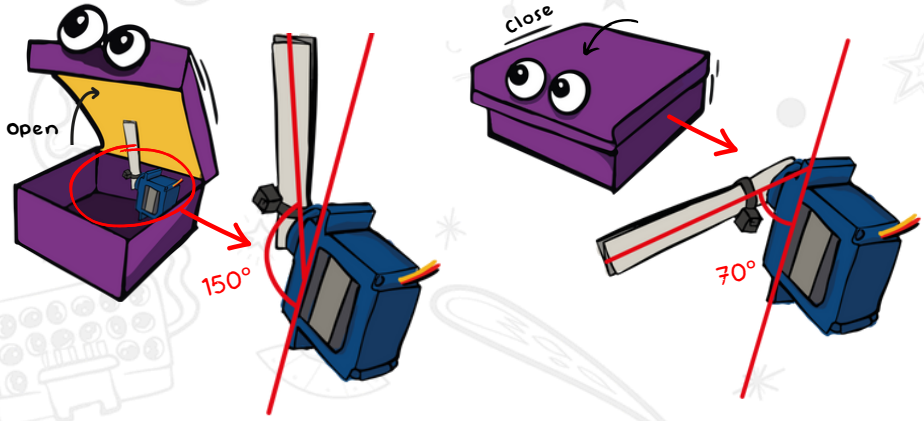


Step 12:
Slide the cable through the back of the box.

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



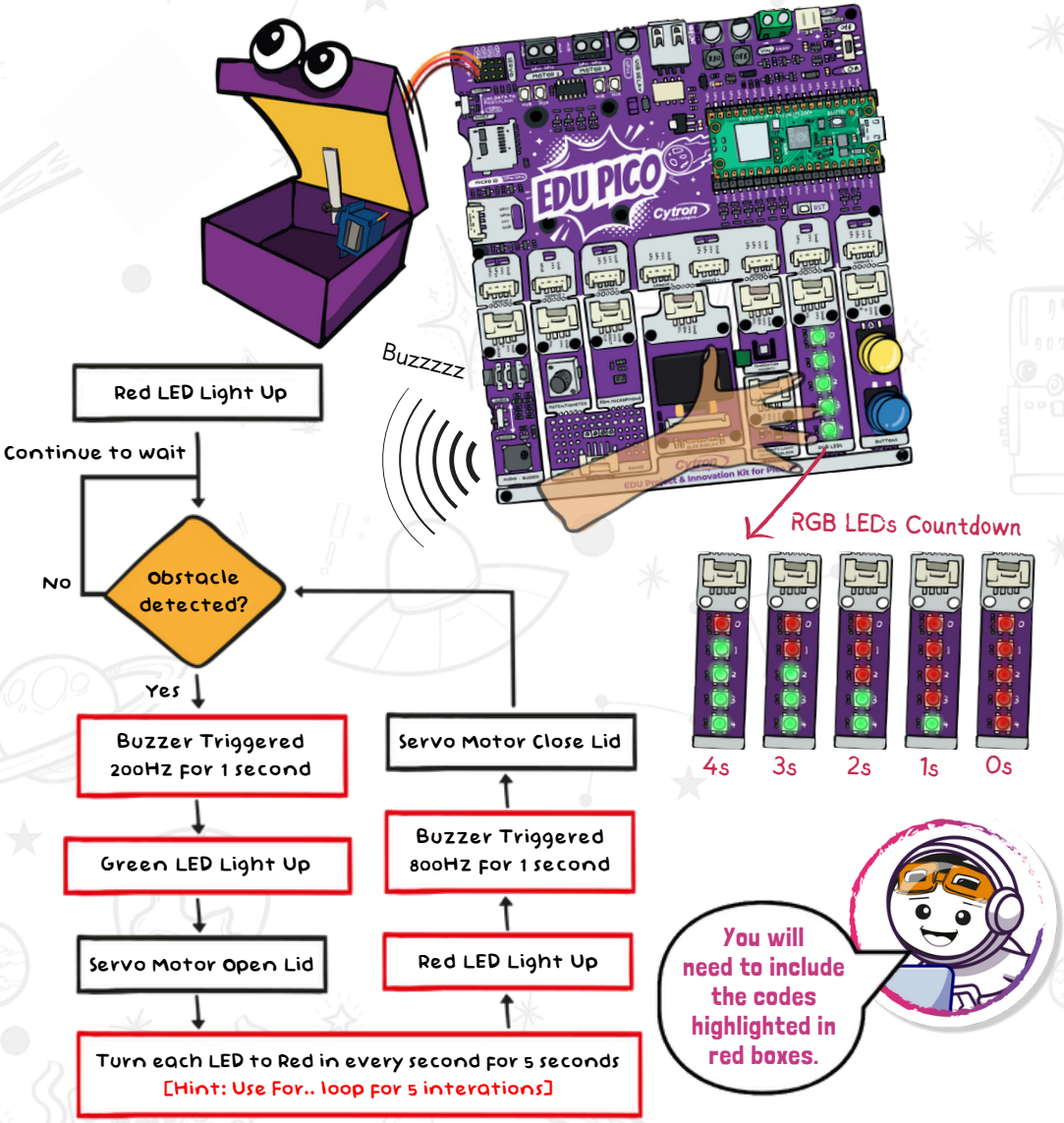
Step 14:
Test the Trashbot's servo motor's angle.



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!



CHAPTER 6

Noise Pollution Monitoring System

Potentiometer & PDM Sound Sensor

Hello Makers! In this chapter, we will cover both Potentiometer and the PDM Sound Sensor. These may not sound as futuristic as gesture sensors or OLED displays, but they hold the power to bring your projects to life in amazing ways.

Imagine being able to control the brightness of an LED or the speed of a motor with a simple twist on the potentiometer, or by reacting to the sound around you. Well, that's exactly what the Potentiometer and PDM Sound Sensor are here for!

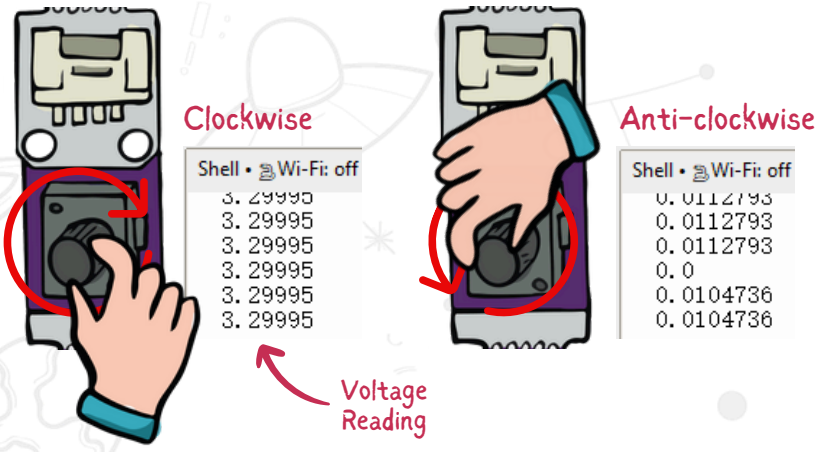
So, grab your EDU PICO, and let's embark on another exciting journey as we uncover the potential of the potentiometer and PDM Sound Sensor. Let's go!

Introduction to Potentiometer

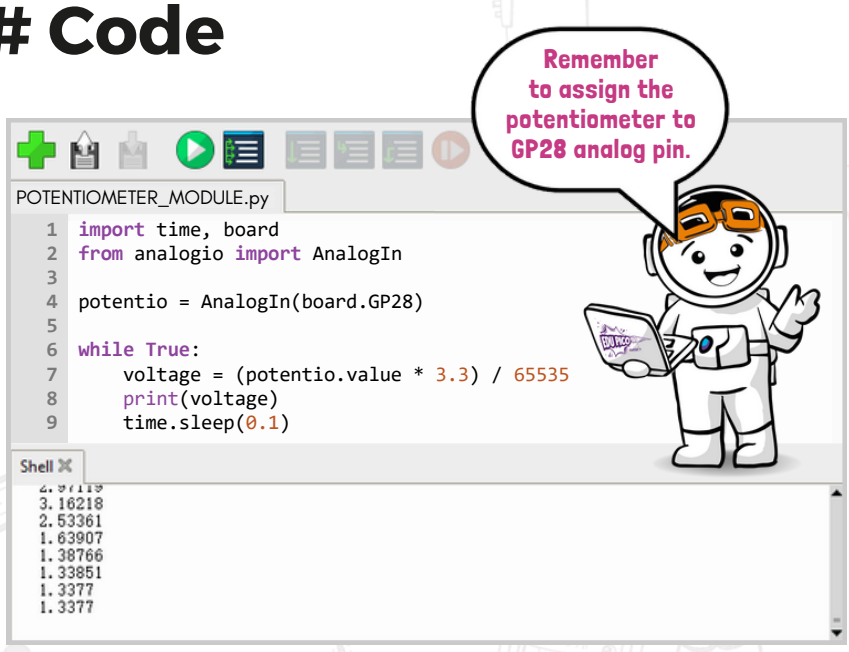
In this lesson, you will learn to interpret analog signal voltage while manipulating a variable resistor or a potentiometer (pot). This is also one of the more popular projects created for beginners to learn how to control electrical output by using a simple analog input device.

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 anti-clockwise will reduce the voltage.
- **Output:**
  - The code continuously reads 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 output voltage values 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)
```

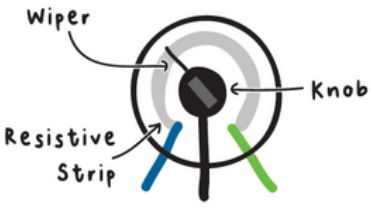
Line 7: Read the analog voltage from the potentiometer using potentio.value. This value is an integer ranging from **0** to **65535 (2^16 - 1)**, which represents the voltage level across the potentiometer. The raw analog value is then converted to voltage by scaling it from the range [0, 65535] to [0,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

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

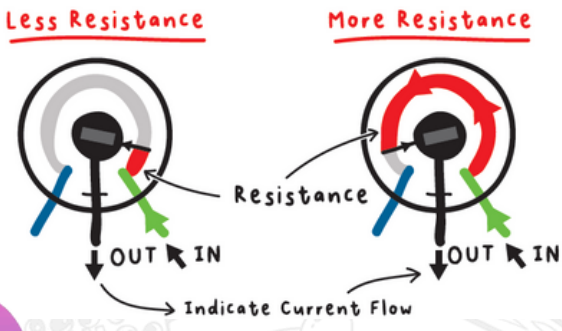
Line 9: Add a small delay of **0.1** second using **time.sleep(0.1)** to limit the rate of voltage readings.

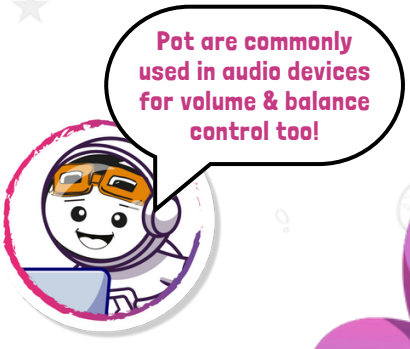
THE MORE YOU KNOW

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.



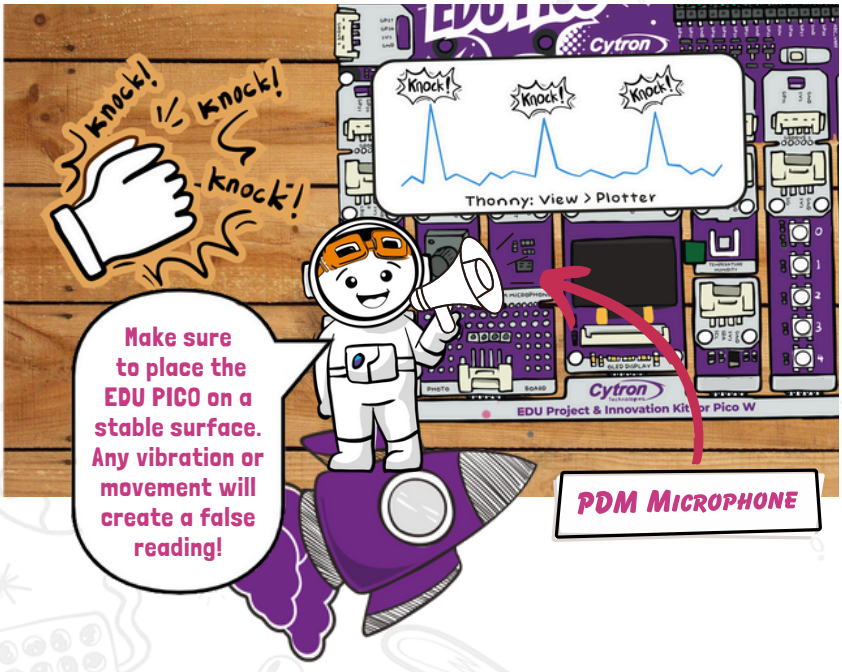


Introduction to Sound Sensor

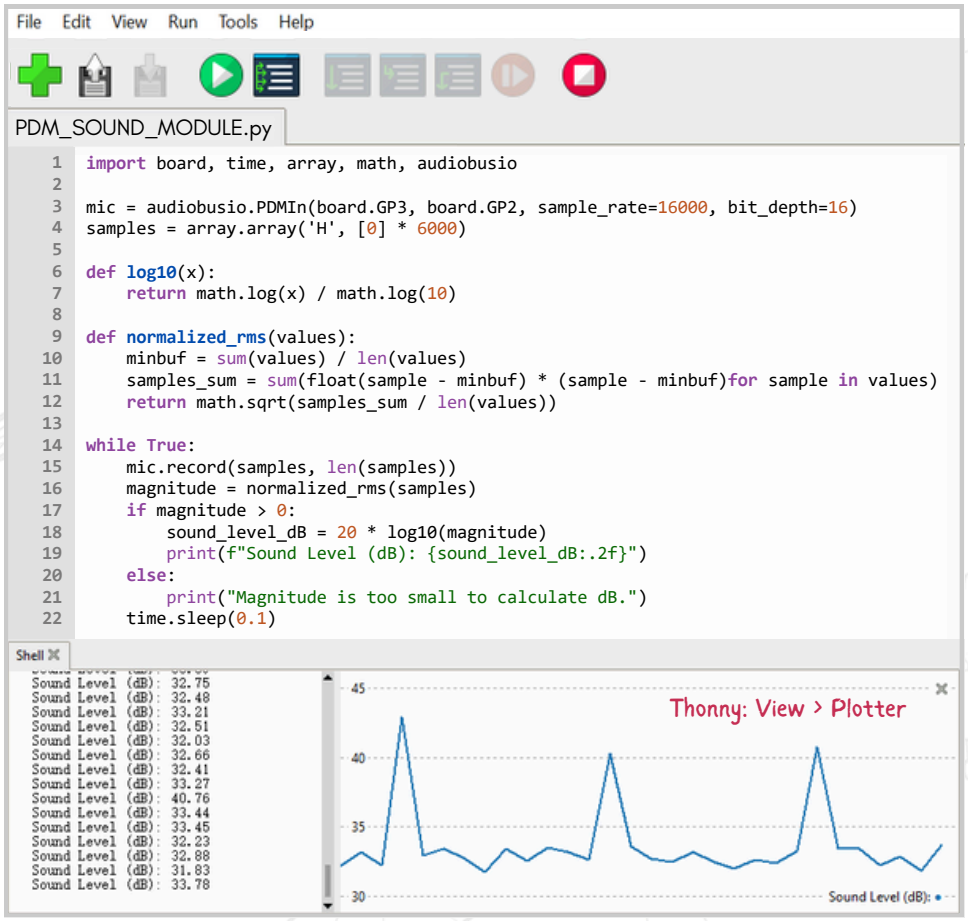
A PDM microphone, or Pulse-Densitiy Modulation microphone, is a type of microphone that converts sound waves into a digital signal. Unlike our traditional microphones that use analog signals, the digital signals produced by PDM microphones are more resistant to noisy environments, making them ideal for 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



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=16000, bit_depth=16)
4 samples = array.array('H', [0] * 6000)
```

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

Line 4: An array named **samples** is created to collect and store **6,000** 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**.

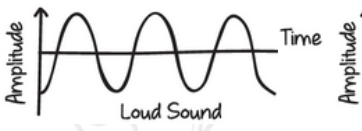
Line 16: Calculate the magnitude of the audio signal using the **normalized_rms** function.

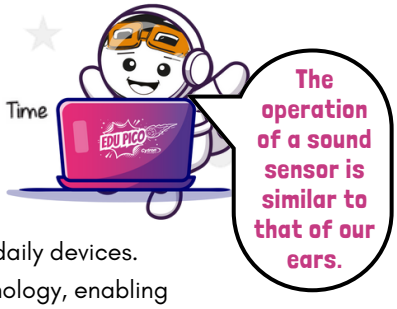
Line 17 - 19: If the magnitude is greater than **0** (indicating that there is some sound), calculate the sound level in decibels (dB) using the **log10** function and print it to the shell console.

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

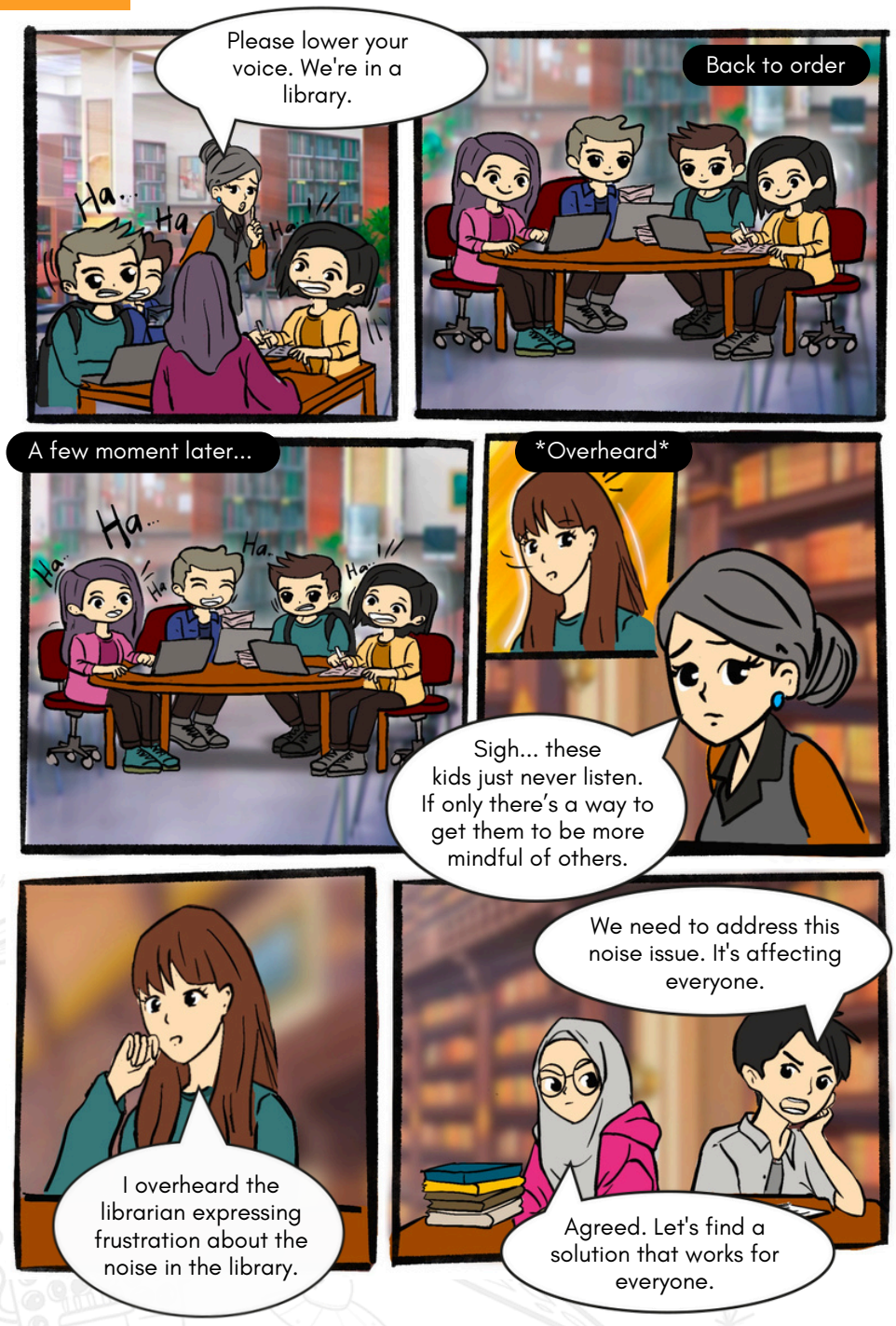
Sound is generated through the vibrations of objects, such as when a drum is struck. These vibrations set the surrounding air molecules (the medium) into motion, resulting in the formation of sound waves.





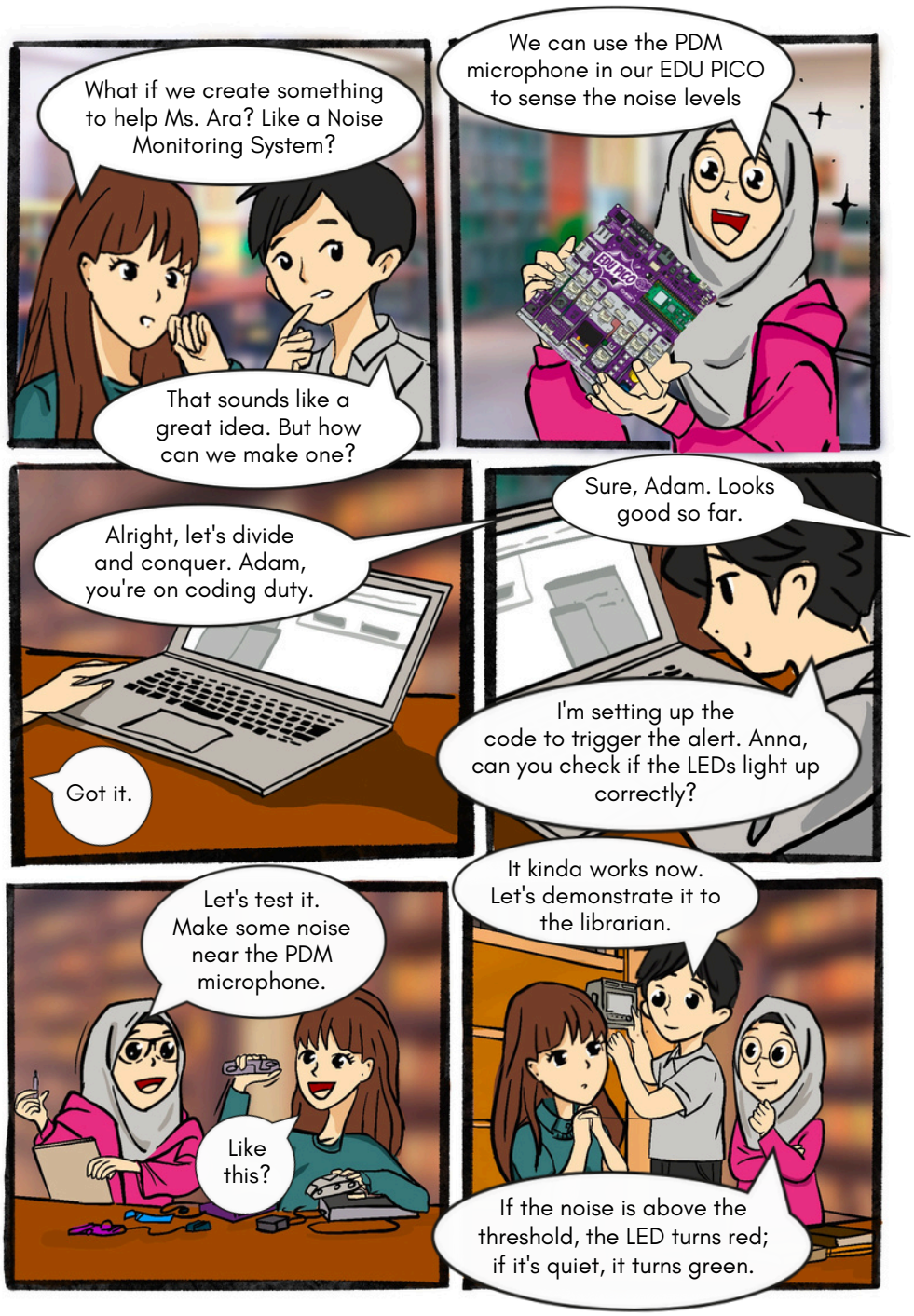
Sound sensors are now integrated into almost all of our daily devices. For example, smartphones utilize voice recognition technology, enabling users to interact with virtual assistants simply by speaking commands or queries.

PROLOGUE: NOISE POLLUTION MONITORING



PROLOGUE: NOISE POLLUTION MONITORING

11 SUSTAINABLE CITIES AND COMMUNITIES



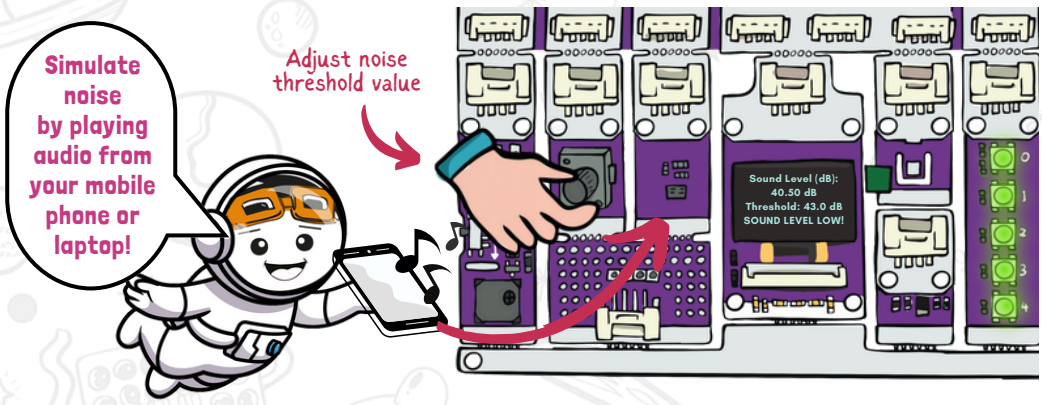
PROLOGUE: NOISE POLLUTION MONITORING

Noise Pollution Monitoring System

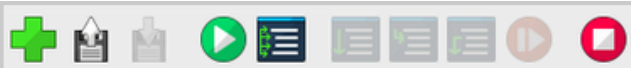
Noise pollution is a critical issue that often goes unnoticed in our ever-expanding urban society, ultimately affecting our overall well-being and quality of life. Fortunately, we can address this problem with the help of PDM microphone and EDU PICO. By learning how to use these tools, we can effectively 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.
- **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=16000, bit_depth=16)
samples = array.array('H', [0] * 6000)

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((sample - minbuf) ** 2 for sample in values)
    return math.sqrt(samples_sum / len(values))

def calculate_sound_level_dB(samples):
    magnitude = normalized_rms(samples)
    sound_level_dB = 20 * log10(magnitude)
    return sound_level_dB

while True:
    oled.fill(0)
    mic.record(samples, len(samples))
    sound_level_dB = calculate_sound_level_dB(samples)
    pot_value = potentiometer.value / 65535 * (sound_max - sound_min) + sound_min

    oled.text("Sound Level (dB):", 15, 5, 1)
    oled.text(f"{sound_level_dB:.2f} dB", 40, 20, 1)
    oled.text(f"Threshold: {pot_value:.1f} dB", 10, 35, 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)
    time.sleep(0.1)
    oled.show()
```

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: Imports **AnalogIn** from **analogio** module for reading analog input.

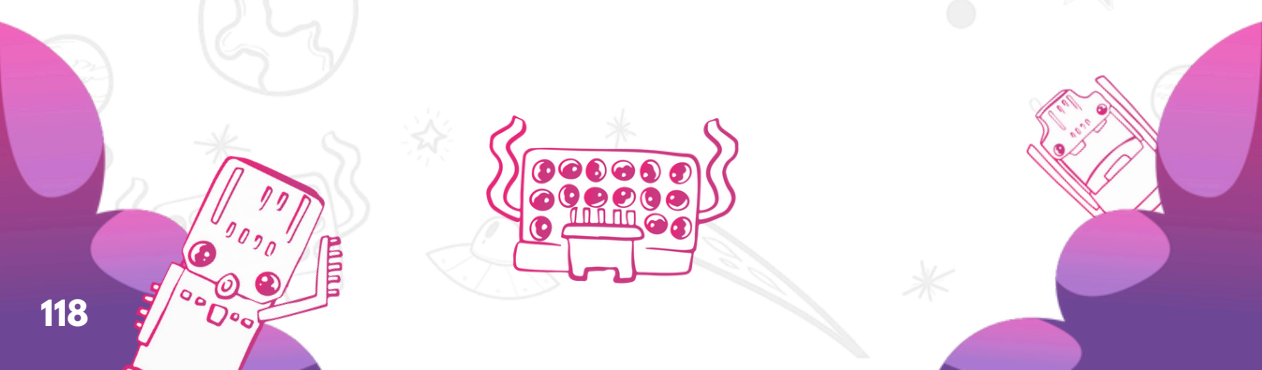
Initialize Hardware Components

Hardware Initialization

```
11 mic = audiobusio.PDMIn(board.GP3, board.GP2, sample_rate=16000, bit_depth=16)
12 samples = array.array('H', [0] * 6000)
13
14 sound_min = 30
15 sound_max = 80
```

Line 11 - 12: Initializes audio input using the PDM (Pulse-Density Modulation) method from the microphone connected to pins ***GP3*** and ***GP2***. It sets the sample rate to **16,000** 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))
24
25 def calculate_sound_level_dB(samples):
26     magnitude = normalized_rms(samples)
27     sound_level_dB = 20 * log10(magnitude)
28     return sound_level_dB
```

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.

Line 25 - 28: The **calculate_sound_level_dB** function calculates the sound level in dB by first finding the RMS magnitude of the audio signal and then converting it to a logarithmic scale with a scaling factor, making it a common method for expressing audio levels in a more human-readable form.

Enter a Continuous Loop

Record & Process Audio Sample

```
30 while True:
31     oled.fill(0)
32     mic.record(samples, len(samples))
33     sound_level_dB = calculate_sound_level_dB(samples)
```

Line 32: 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 33: Calls the **calculate_sound_level_dB** function to calculate and return the sound level in decibels (dB) using the audio samples from the microphone.

Reads Potentiometer Value & Setup OLED

```
34      pot_value = potentiometer.value / 65535 * (sound_max - sound_min) + sound_min
35
36      oled.text("Sound Level (dB):", 15, 5, 1)
37      oled.text(f"{sound_level_dB:.2f} dB", 40, 20, 1)
38      oled.text(f"Threshold: {pot_value:.1f} dB", 10, 35, 1)
```

Line 34: Reads the value of the potentiometer, maps it to a range between **sound_min** and **sound_max**, and stores the result in the **pot_value** variable. This value represents the threshold for determining whether the sound level is high or low.

Lines 36 - 38: Update the OLED display with text information. They display 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

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

Line 40 - 42: 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 screen.

Line 43 - 45: 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 screen.

CHALLENGE - SERVO SOUND METER

Having an OLED to display noise decibels 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.

The code below should help get you started. Fill in the missing code with the appropriate information based on the description given:

A: Include necessary libraries here to run a servo motor.

B: Initialize the servo motor pin and perform a servo test by sweeping the motor through its range from 0 to 180 degrees.

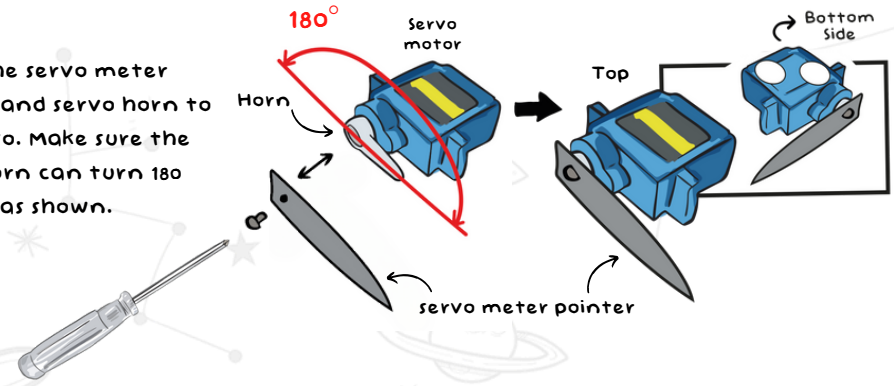
C: Calculate the angle for the servo motor based on sound level in decibels (dB). Check the angle to make sure it's within 0 to 180 degrees before rotating the servo motor.

```
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=16000, bit_depth=16)
samples = array.array('H', [0] * 6000)
                                    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))
    sound_level_dB = calculate_sound_level_dB(samples)
    pot_value = potentiometer.value / 65535 * (sound_max - sound_min) + sound_min

    oled.text("Sound Level (dB):", 15, 5, 1)
    oled.text(f"{sound_level_dB:.2f} dB", 40, 20, 1)
    oled.text(f"Threshold: {pot_value:.1f} dB", 10, 35, 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()
```

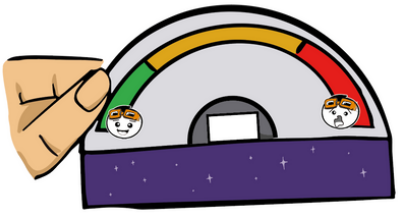
CHALLENGES

SERVO SOUND METER ACCESSORY

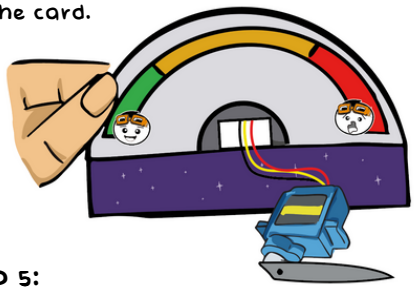
Step 1:
Screw the servo meter pointer and servo horn to the servo. Make sure the servo horn can turn 180 degrees as shown.



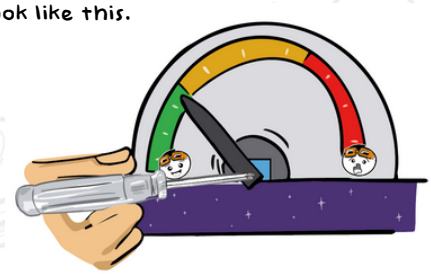
Step 2:
Push through the middle to make a hole on the card accessory.



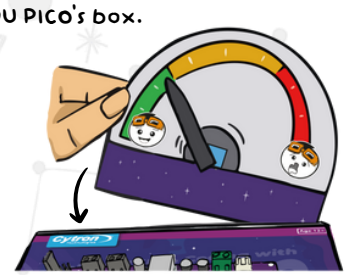
Step 3:
Plug in the servo motor from the front of the card.



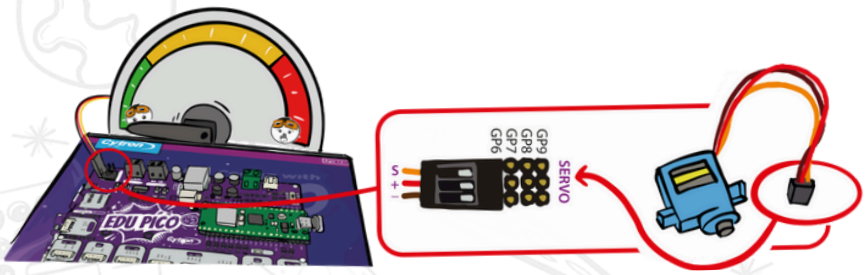
Step 4:
The assembled piece will look like this.



Step 5:
Slot the card into the slit on the EDU PICO's box.

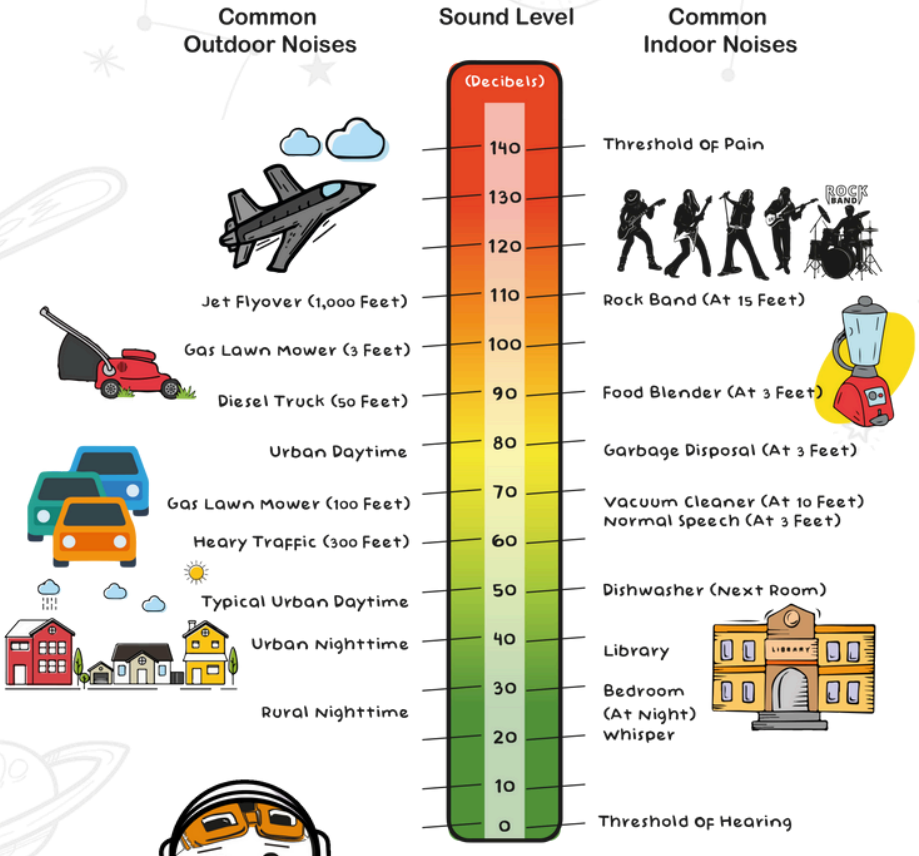


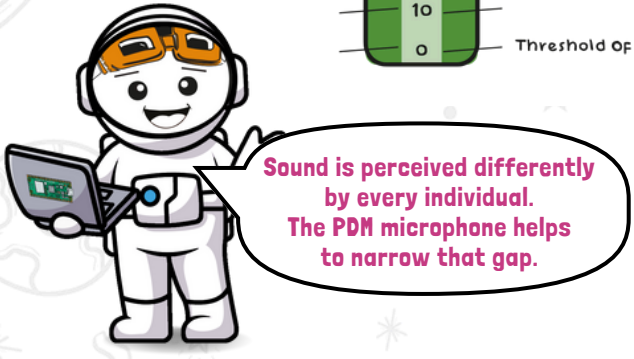
Step 6:
Connect Servo Motor Cable to EDU PICO board at GP6 and now you're ready!



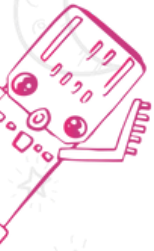
WHAT'S NEXT?

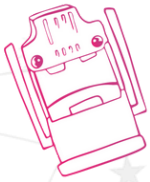
Once you're done with the program, here's what you can do to make this project more exciting. Take a walk with your EDU PICO with the noise monitoring program still running. Try to identify the noise level in each location indicated below. Is the measured noise level within the safe range as shown in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





CHAPTER 7





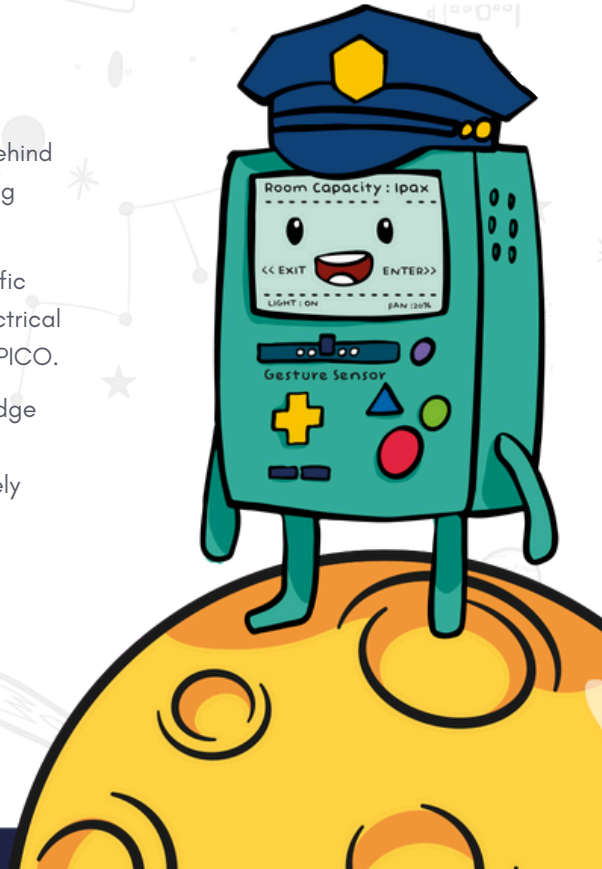
Smart Classroom

DC Motor & Relay

Imagine having the power to move things at your command. DC motors are here to do just that. It is the main component that acts 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.

By the end of this chapter, you'll have the knowledge and skills needed to create an environment that responds to your needs and preferences, ultimately transforming an ordinary classroom into a smart classroom.

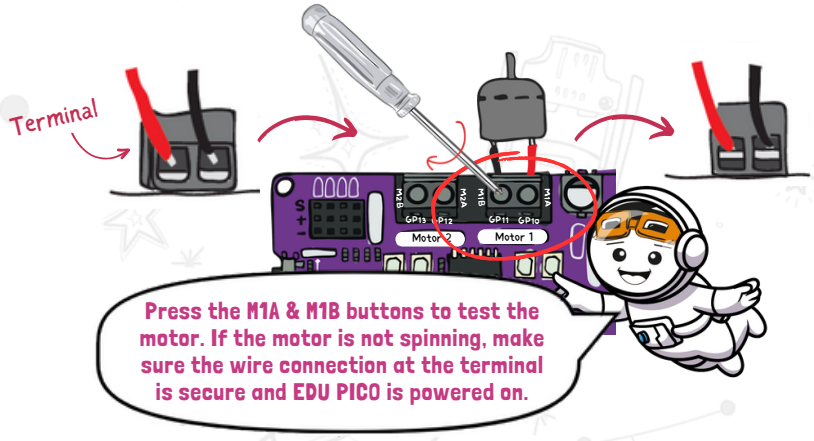


Introduction to DC Motor

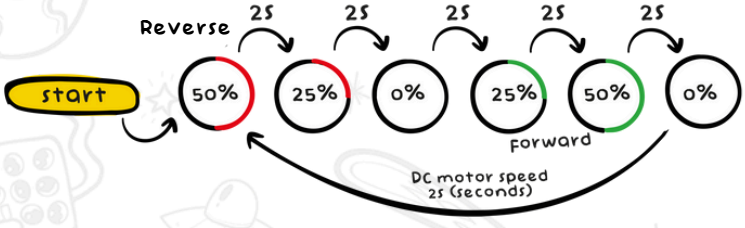
In this lesson, we will finally get our hands on a DC motor. This particular invention serves as the cornerstone of automation, converting electric energy into motion. Its a versatile component that can power an array of mechanisms, from spinning wheels to robotic arms. Consider it as a tool that can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

How Does This Activity Work?

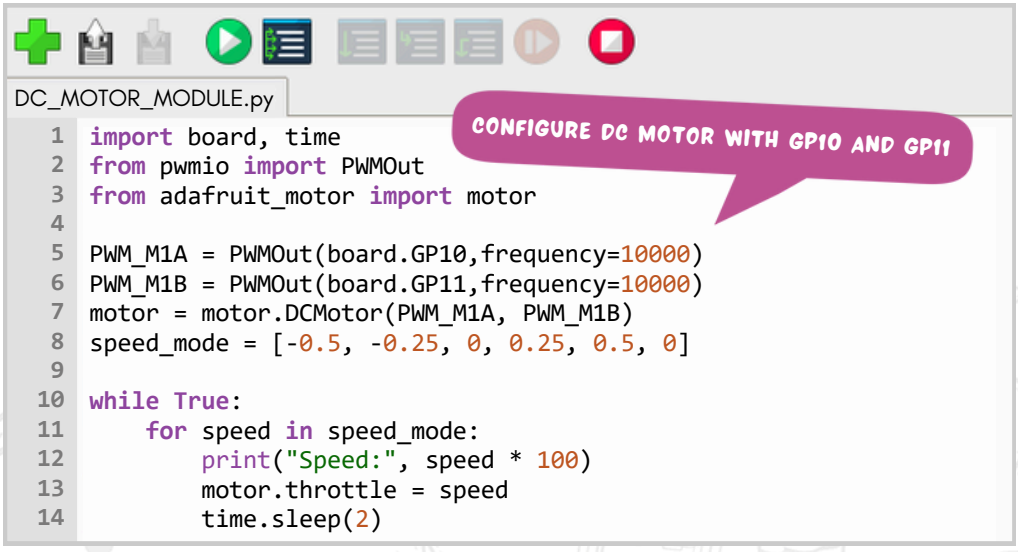
- **Libraries:** board, time, busio, pwmio, adafruit_motor.
- **DC Motor Configuration:**
  - PWM_M1A to ***GP10***, PWM_M1B to ***GP11***.
  - Connect DC Motor to M1A & M1B terminals as shown below:



- **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

CONFIGURE DC MOTOR WITH GP10 AND GP11

```
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)
```

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.

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: Prints the current speed percentile to the 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**.

MINI ACTIVITY

From the previous chapter, we have learned 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!

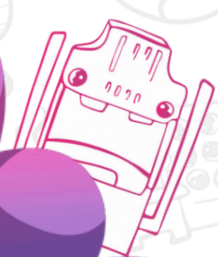
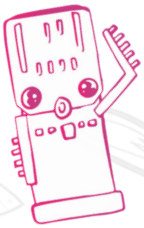
Fill In The Blank

```
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.

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.

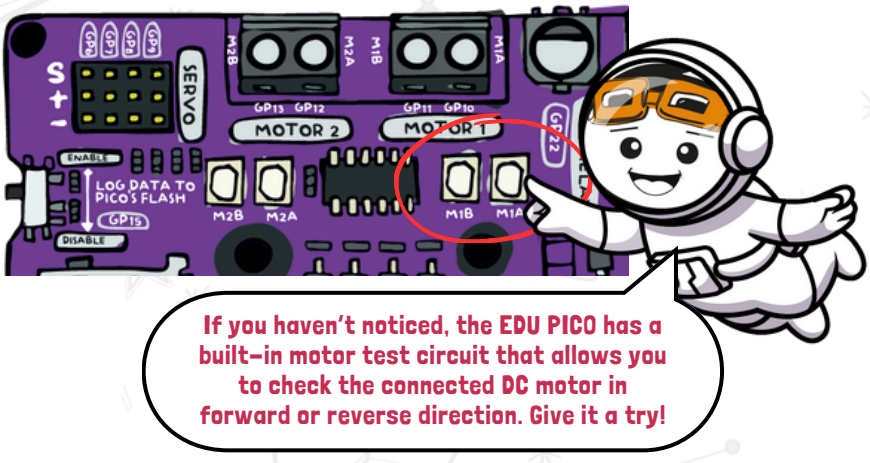




THE MORE YOU KNOW

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.

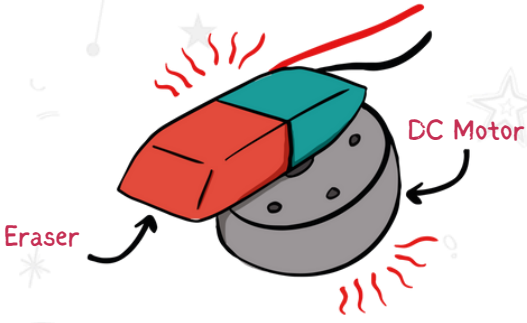
DC Motor

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!



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.

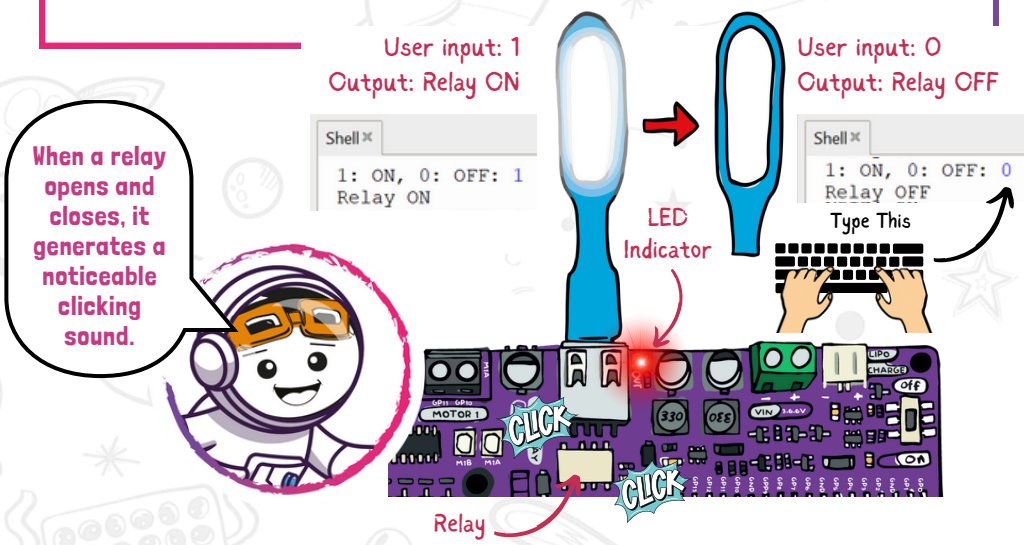


Introduction to Relay

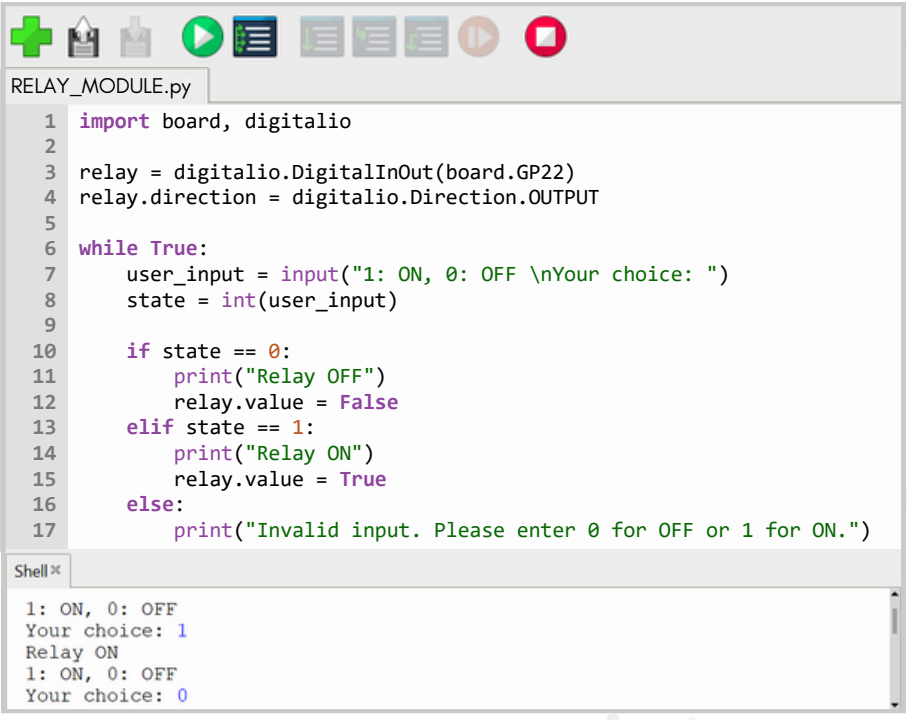
In this lesson, we will learn how to operate the USB relay on the EDU PICO, but more importantly, to understand how a relay works. A relay is an electromechanical device that acts as a switch controlled by an electrical signal. It allows a low-power circuit to control a high-power circuit, making it 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

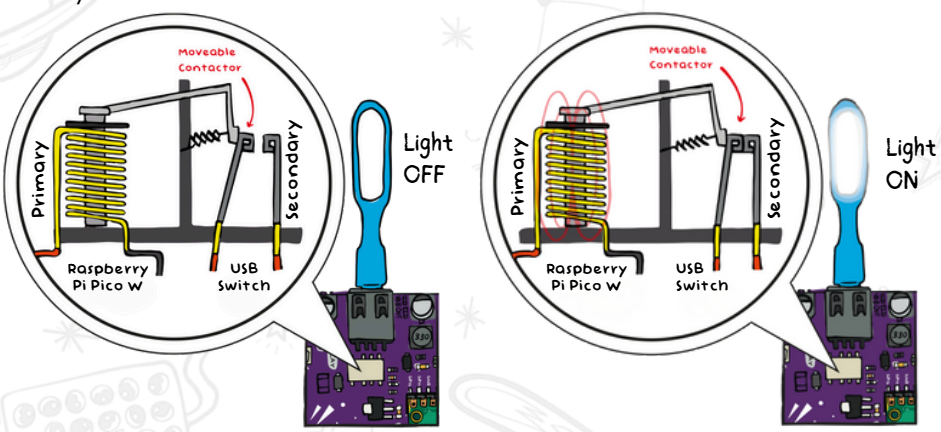
Relay

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

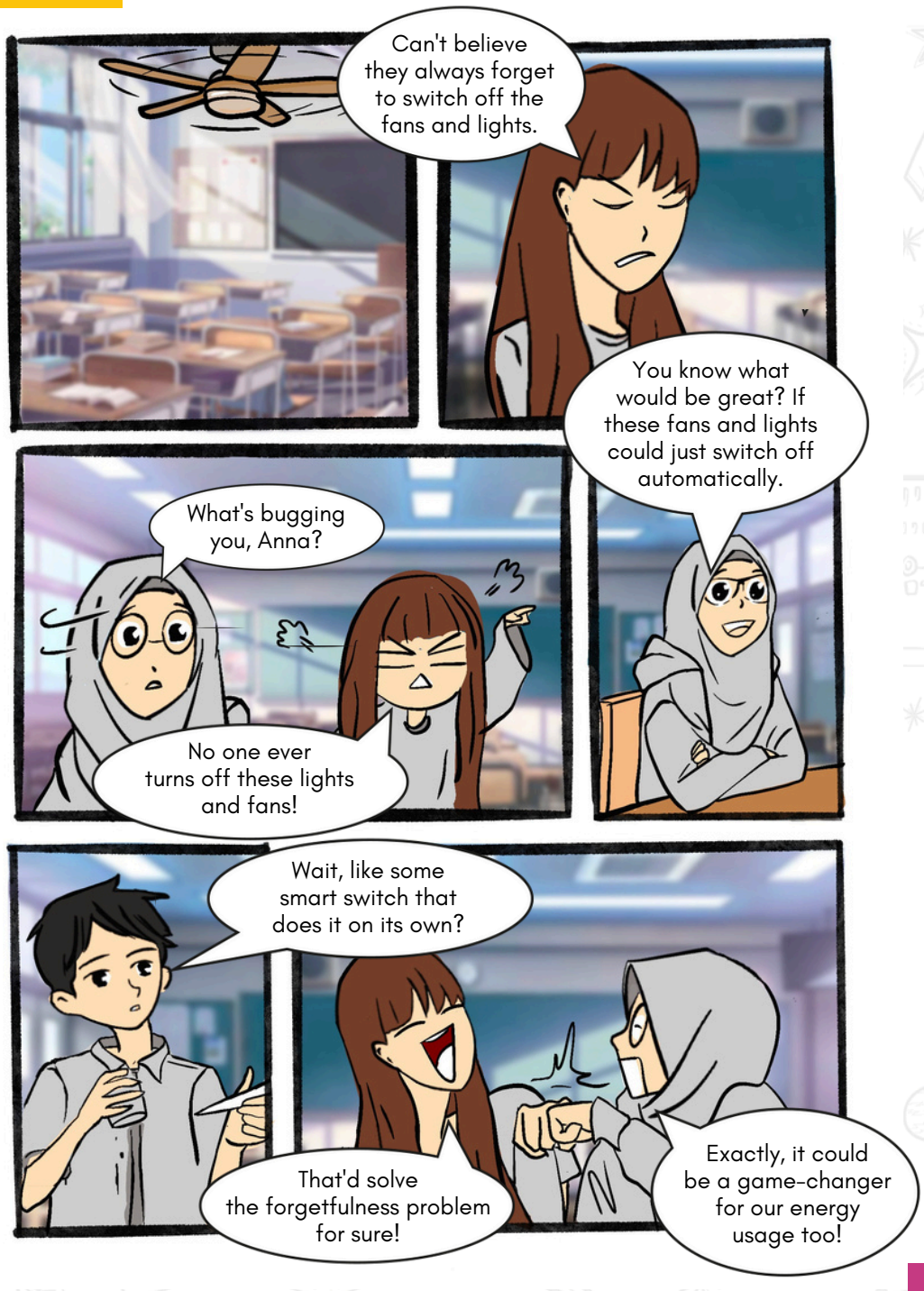
The **primary circuit** mainly receives an external signal that controls the ON / OFF operation of the relay; in this case, it's connected to EDU PICO ***GP22***.

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.

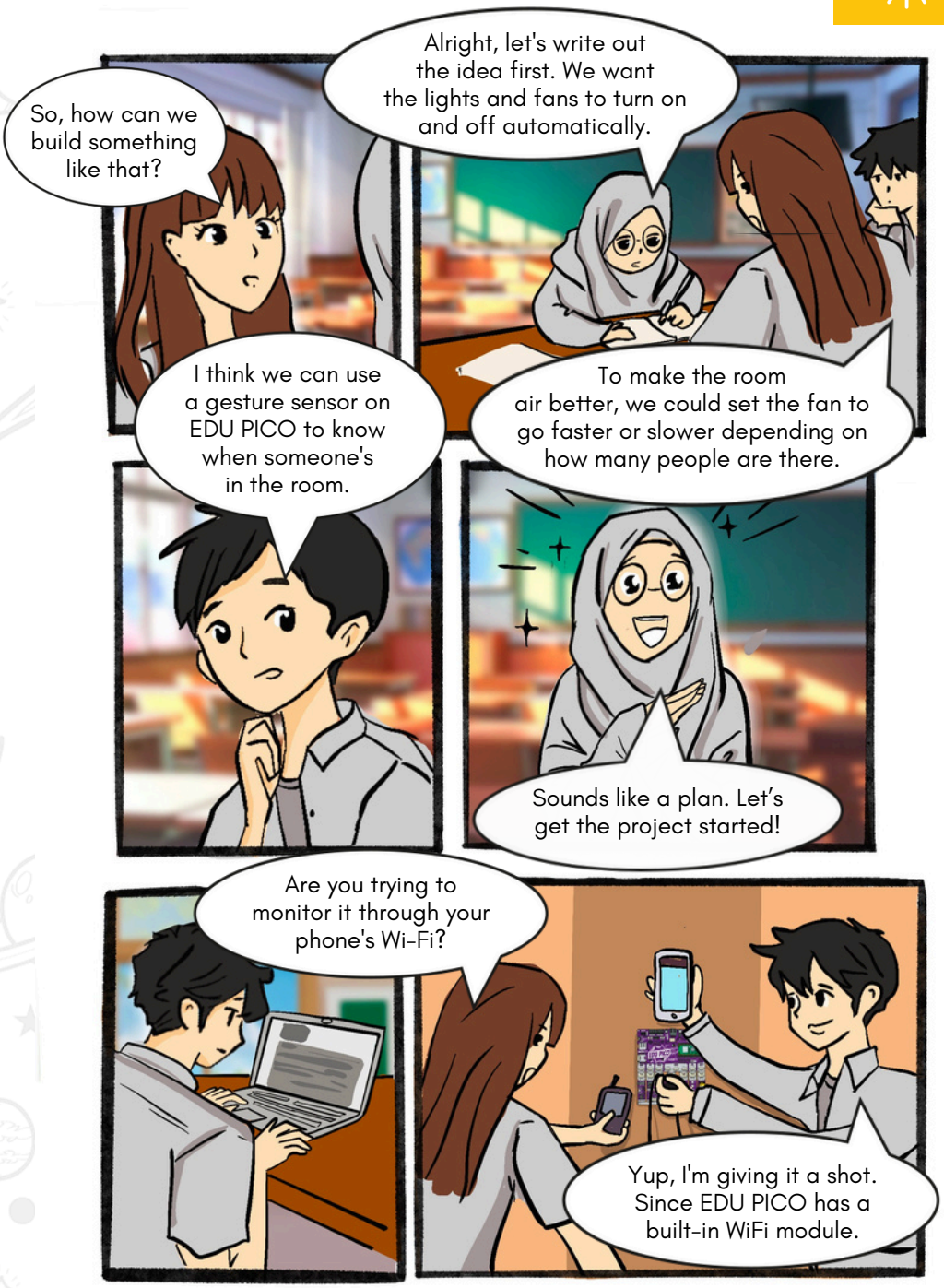


PROLOGUE: SMART CLASSROOM

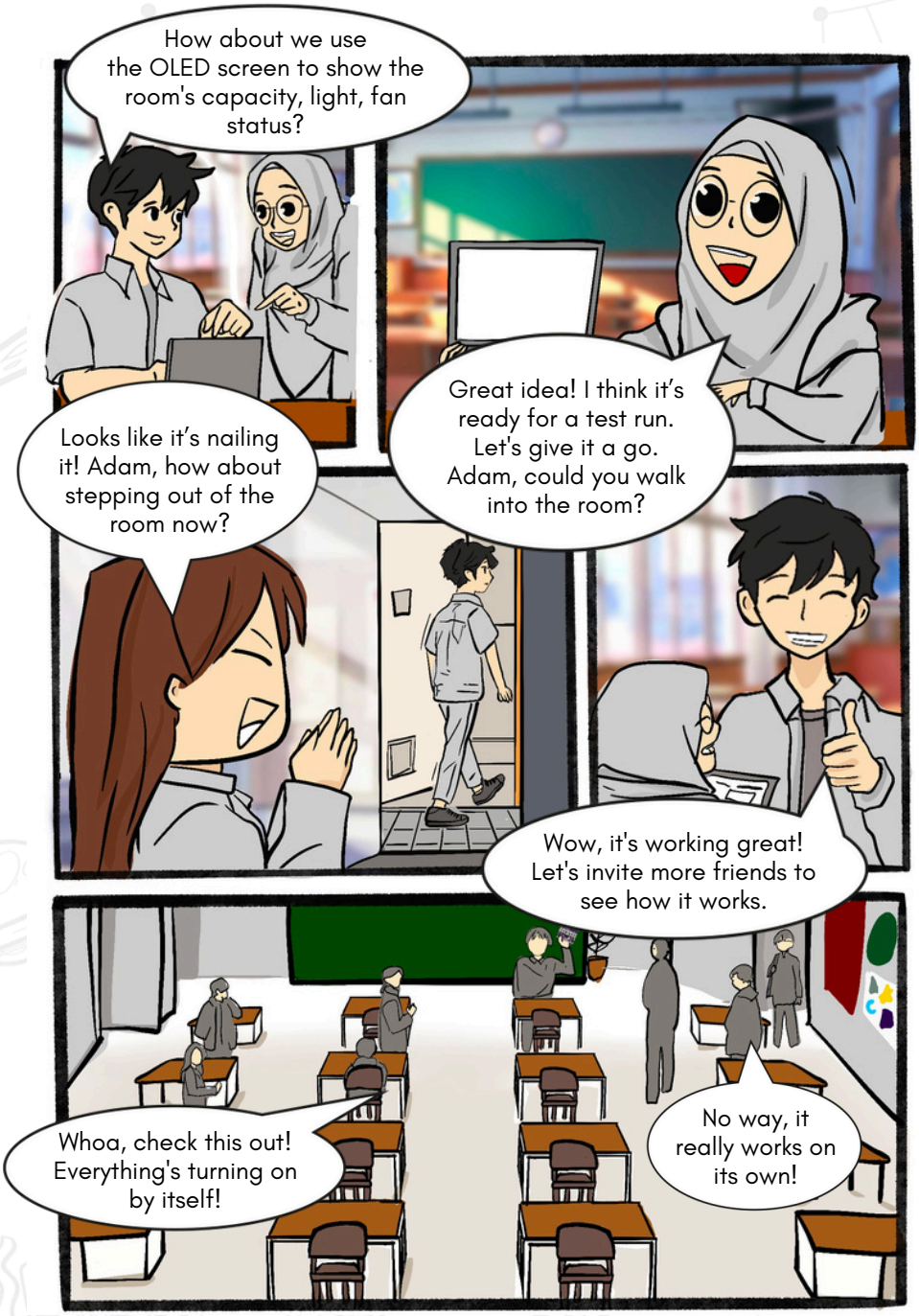


PROLOGUE: SMART CLASSROOM

7 AFFORDABLE AND CLEAN ENERGY

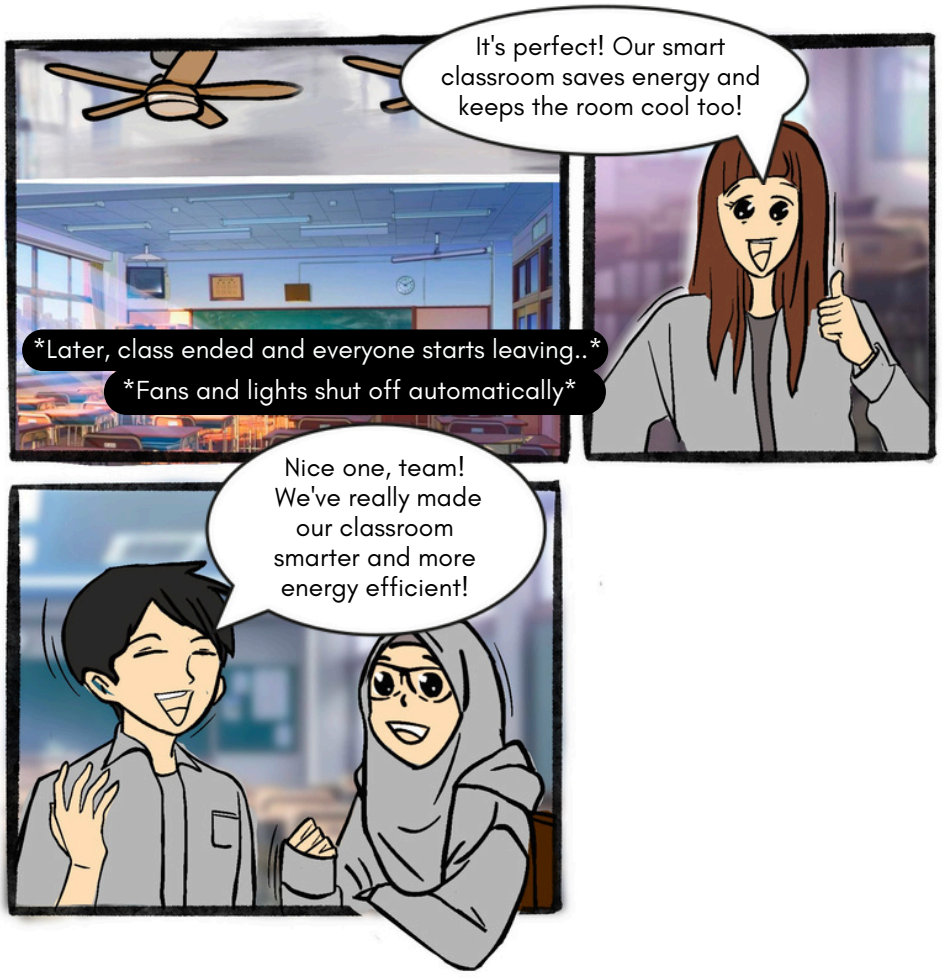


PROLOGUE: SMART CLASSROOM

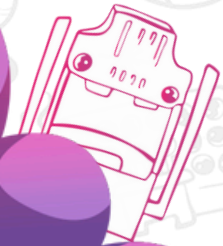


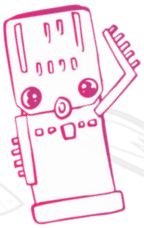
PROLOGUE: SMART CLASSROOM

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.]



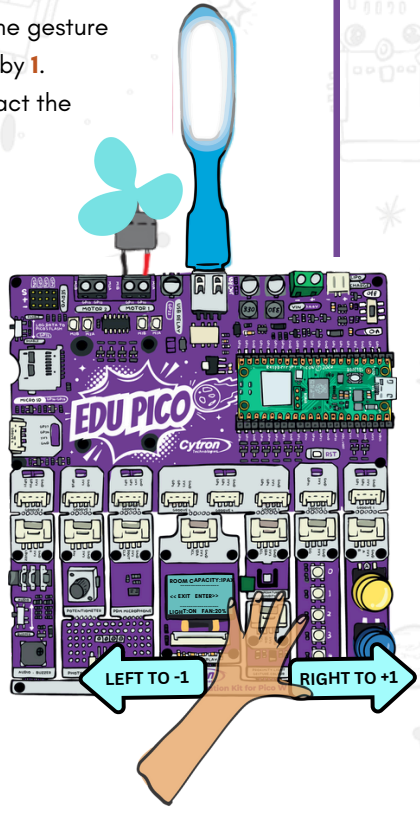


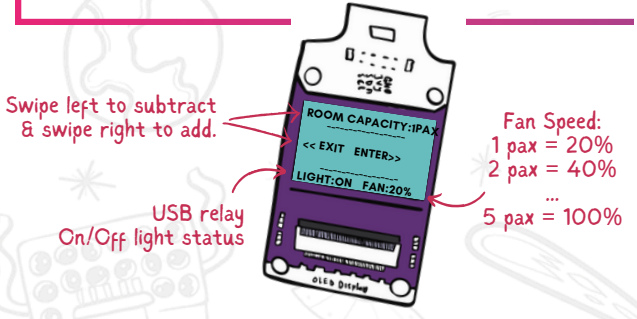
Smart Classroom

In this project, we will learn to integrate APDS9960 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, APDS9960, pwmio, adafruit_motor, adafruit_ssd1306.
- **DC Motor Configuration:** PWM_M1A to ***GP10***, PWM_M1B to ***GP11***.
- **USB Relay Configuration:** ***GP22***.
- **OLED and APDS9960 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
from adafruit_apds9960.apds9960 import APDS9960

i2c = busio.I2C(board.GP5, board.GP4)
apds = APDS9960(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

apds.enable_gesture = True
apds.enable_proximity = True

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)

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 = apds.gesture()

    if gesture == 3:
        class_num = max(0, class_num - 1)
    elif gesture == 4:
        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 adafruit_apds9960.apds9960 import APDS9960
```

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 APDS9960 gesture sensor.

Initialize Hardware Components

Initialize I2C, APDS9960 Sensor, PWM for Motor, OLED Display, and Relay

```
6  i2c = busio.I2C(board.GP5, board.GP4)
7  apds = APDS9960(i2c)
8  PWM_M1A = PWMOut(board.GP10, frequency=10000)
9  PWM_M1B = PWMOut(board.GP11, frequency=10000)
10 motor_instance = motor.DCMotor(PWM_M1A, PWM_M1B)
11 oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
12 relay = digitalio.DigitalInOut(board.GP22)
13 relay.direction = digitalio.Direction.OUTPUT
```

These lines initialize the I2C interface, the APDS9960 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 APDS9960 gesture sensor.

Line 8 - 10: PWM outputs for controlling the DC motor.

Line 11: Configure the OLED display with the assigned I2C pins.

Line 12 - 13: Configure ***GP22*** as digital output for controlling the relay.

Enable Gestures and Proximity on the APDS9960 Sensor

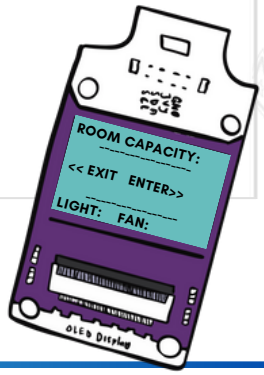
```
15 apds.enable_gesture = True
16 apds.enable_proximity = True
17
18 class_num = 0
19 current_speed = 0.0
```

Line 15 - 16: Enable gesture and proximity detection on the APDS9960 sensor.

Line 18 - 19: Initialize variables for class capacity and current fan speed.

Set Up the OLED Display

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



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

```
29 def handle_gesture():
30     global class_num, current_speed
31
32     oled.fill_rect(90, 7, 35, 7, 0)
33     oled.fill_rect(40, 50, 20, 7, 0)
34     oled.fill_rect(90, 50, 35, 7, 0)
35     gesture = apds.gesture()
36
37     if gesture == 3:
38         class_num = max(0, class_num - 1)
39     elif gesture == 4:
40         class_num = min(5, class_num + 1)
```

Line 30: 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 32 - 34: 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 35: Retrieves gesture value from the APDS9960 sensor where **3** represents a left-to-right gesture, and **4** represents a right-to-left gesture.

Line 37 - 40: If gesture detected is equal to **3**, the **class_num** variable will reduce by **1** (but not below **0**), and if gesture **4** is detected, it increases the **class_num** by **1** (but not above **5**).

Adjust Fan Speed

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

Line 42 - 44: 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

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

Line 46: 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 47: Updates the OLED display to show whether the relay is ON or OFF. It uses a conditional (ternary) expression where if **class_num** is greater than **0**, the text is set to "ON"; otherwise, it is set to "OFF".

Line 49: 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 50 - 51: Updates the OLED display to show the room capacity status.

Enter a Continuous Loop

Main Loop, try...finally

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

Line 53 - 55: The **try** block contains the main loop of the program. The **while True** loop ensures that the **handle_gesture** function is continuously called.

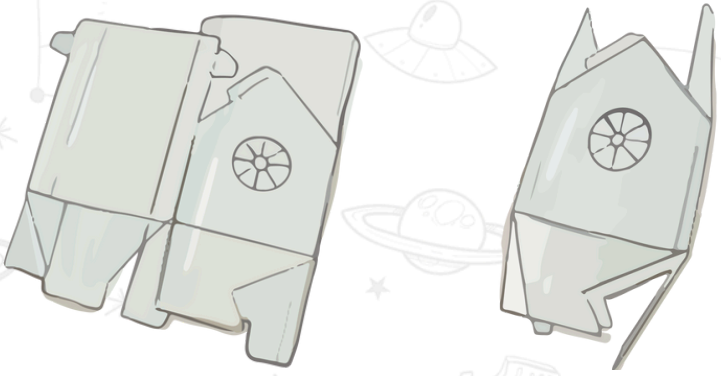
Line 57 - 59: The **finally** block is used for cleanup operations which include resetting the OLED screen to its original state.

Line 61: Deinitializes the I2C interface. *[Note: This is important for releasing the resources used by I2C before the program terminates.]*

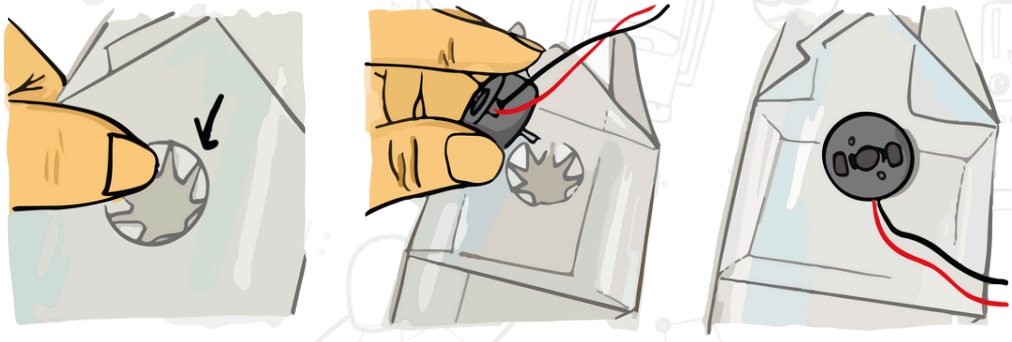
WHAT'S NEXT?

HOUSE ACCESSORY - DC MOTOR FAN

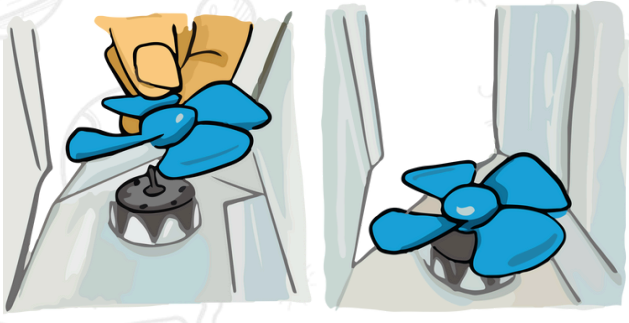
Step 1:
Prepare the house accessory by opening it into a shape of a box.



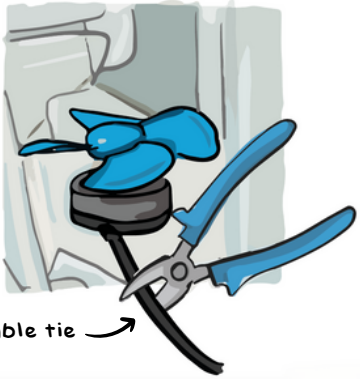
Step 2:
Push the DC motor hole inwards and push the DC motor through the hole carefully.



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

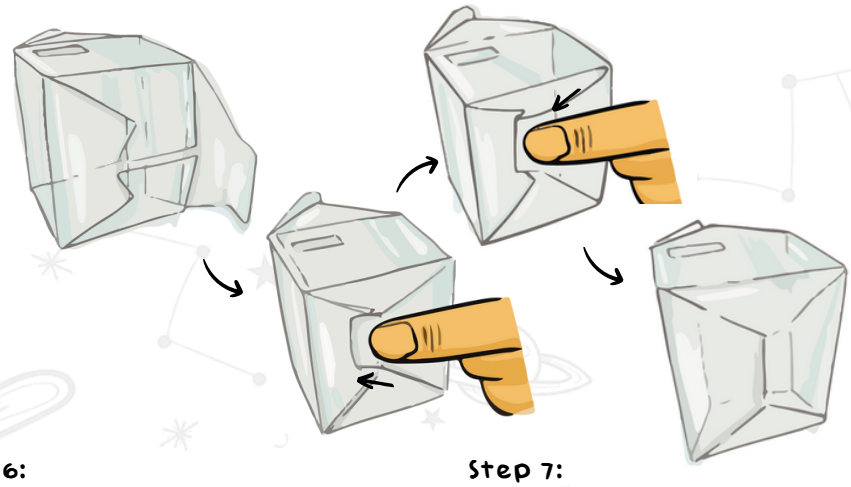


Step 4:
Secure a cable tie around the DC motor.



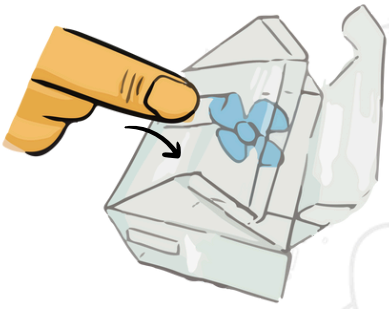
Step 5:

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



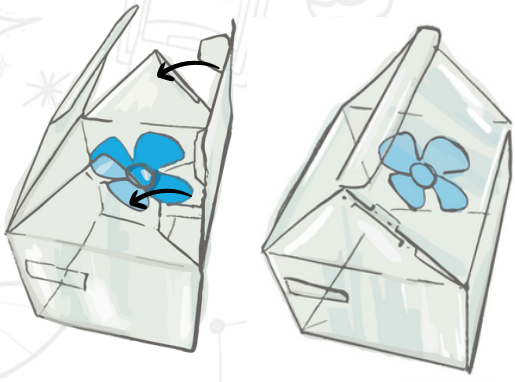
Step 6:

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



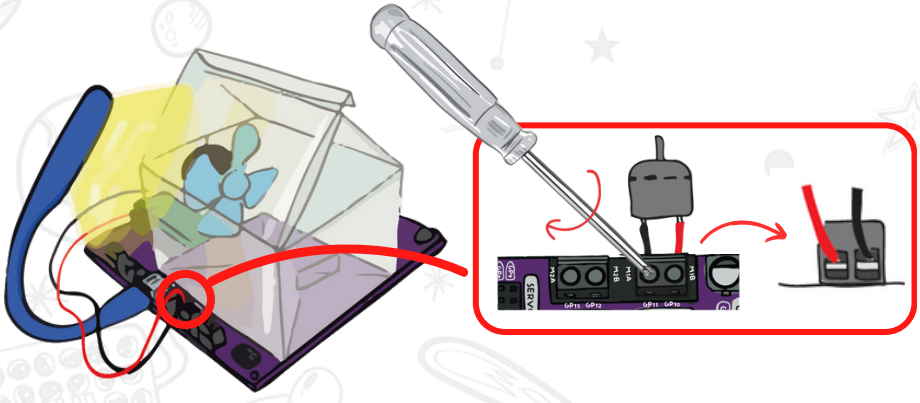
Step 7:

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.



Bonus: Wireless Network (AP-Mode)

By the end of this guide, you'll be able to build 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 USB port.

Connecting with Computer

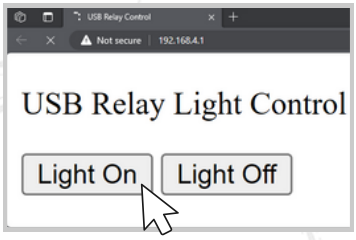
- **Output:**
  - The program starts the server and prints "Starting server..." followed by "Listening on http://[IP_ADDRESS]" where [IP_ADDRESS] is the IP address of the WiFi access point.

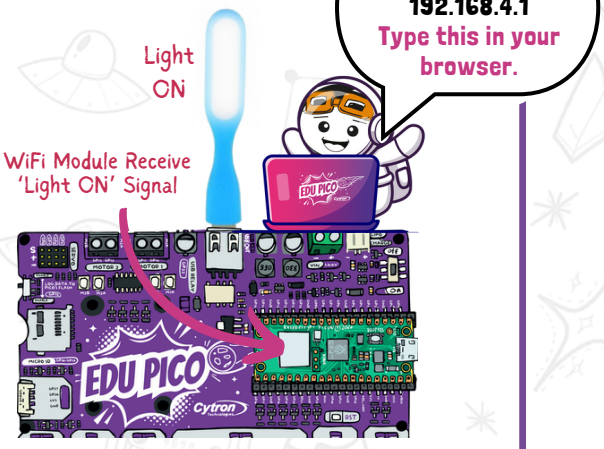
```
Shell
>>> %Run -c $EDITOR_CONTENT
 Starting server...
 Listening on http://192.168.4.1
```

  - 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.

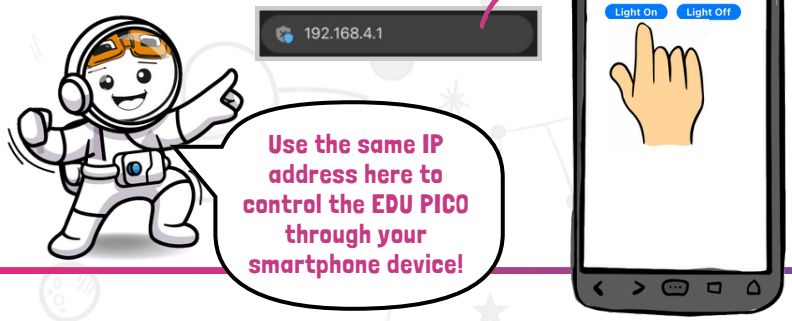
- Once you're on the webpage, clicking these buttons will send a signal to the server, and the relay state will change accordingly. Messages like "Light ON" and "Light OFF" will be printed on the shell console when the buttons are clicked.





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.



Code

BONUS_AP_RELAY.py

```
import board, digitalio
import wifi, socketpool
from adafruit_httpserver import Server, Request, Response, POST

def setup_wifi_ap():
    ap_ssid = "EDUPICO_AP"
    ap_password = "12345678"
    wifi.radio.start_ap(ssid=ap_ssid, password=ap_password)
    pool = socketpool.SocketPool(wifi.radio)
    return pool

def setup_relay():
    relay = digitalio.DigitalInOut(board.GP22)
    relay.direction = digitalio.Direction.OUTPUT
    return relay

def light_on(relay):
    print("Light ON")
    relay.value = True

def light_off(relay):
    print("Light OFF")
    relay.value = False

def webpage():
    html = """
    <!DOCTYPE html>
    <html>
    <head>
    <meta http-equiv="refresh" content="5">
    <title>USB Relay Control</title>
    </head>
    <body>
    <p>USB Relay Light Control</p>
    <form accept-charset="utf-8" method="POST">
    <button class="button" name="Light On"
    value="light_on" type="submit">Light On</button></a>
    <button class="button" name="Light Off"
    value="light_off" type="submit">Light Off</button></a>
    </form>
    </body>
    </html>
    """
    return html
```

```
46  def setup_server(pool, relay):
47      server = Server(pool, "/static")
48
49      @server.route("/")
50      def base(request: Request):
51          return Response(request, f"{webpage()}", content_type='text/html')
52
53      @server.route("/", POST)
54      def buttonpress(request: Request):
55          if request.method == POST:
56              raw_text = request.raw_request.decode("utf8")
57              if "light_on" in raw_text:
58                  light_on(relay)
59              if "light_off" in raw_text:
60                  light_off(relay)
61              return Response(request, f"{webpage()}", content_type='text/html')
62
63      print("Starting server...")
64      server.start(str(wifi.radio.ipv4_address_ap))
65      print("Listening on http://%s" % wifi.radio.ipv4_address_ap)
66      return server
67
68  pool = setup_wifi_ap()
69  relay = setup_relay()
70  server = setup_server(pool, relay)
71  while True:
72      server.poll()
```

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
2  import wifi, socketpool
3  from adafruit_httpserver import Server, Request, Response, POST
```

Line 2: 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 3: Libraries for setting up a simple HTTP server.

Define Custom Functions

Set Up WiFi AP

```
5  def setup_wifi_ap():
6      ap_ssid = "EDUPICO_AP"
7      ap_password = "12345678"
8      wifi.radio.start_ap(ssid=ap_ssid, password=ap_password)
9      pool = socketpool.SocketPool(wifi.radio)
10     return pool
```

Line 5 - 10: This function configures the Raspberry Pi Pico 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.]

Initialize USB Relay

```
12 def setup_relay():
13     relay = digitalio.DigitalInOut(board.GP22)
14     relay.direction = digitalio.Direction.OUTPUT
15     return relay
16
17 def light_on(relay):
18     print("Light ON")
19     relay.value = True
20
21 def light_off(relay):
22     print("Light OFF")
23     relay.value = False
```

Line 12 - 15: Configure ***GP22*** as digital output for controlling the relay. The relay is used to control the ON and OFF of the USB light stick.

Line 17 - 23: These functions update the state of the relay and print a message to the shell console.

Webpage HTML

```
25 def webpage():
26     html = """
27     <!DOCTYPE html>
28     <html>
29     <head>
30     <meta http-equiv="refresh" content="5">
31     <title>USB Relay Control</title>
32     </head>
33     <body>
34     <p>USB Relay Light Control</p>
35     <form accept-charset="utf-8" method="POST">
36     <button class="button" name="Light On"
37     value="light_on" type="submit">Light On</button></a>
38     <button class="button" name="Light Off"
39     value="light_off" type="submit">Light Off</button></a>
40     </form>
41     </body>
42     </html>
43     """
44     return html
```

Line 25 - 44: 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.

Setting Up the HTTP Server

```
46 def setup_server(pool, relay):
47     server = Server(pool, "/static")
48
49     @server.route("/")
50     def base(request: Request):
51         return Response(request, f"{webpage()}", content_type='text/html')
52
53     @server.route("/", POST)
54     def buttonpress(request: Request):
55         if request.method == POST:
56             raw_text = request.raw_request.decode("utf8")
57             if "light_on" in raw_text:
58                 light_on(relay)
59             if "light_off" in raw_text:
60                 light_off(relay)
61             return Response(request, f"{webpage()}", content_type='text/html')
62
63     print("Starting server...")
64     server.start(str(wifi.radio.ipv4_address_ap))
65     print("Listening on http://%s" % wifi.radio.ipv4_address_ap)
66     return server
```

Line 46: Creates an instance of the Server class with the provided pool (socket pool) and a static route ("/static"). The static route may be used for serving static files like stylesheets or images.

Line 49 - 51: Generates an HTML response with the HTML content using the **webpage()** function and sends it back to the client.

Line 53 - 61: The **buttonpress** function is called when a POST request is received. It checks if the request contains data related to turning the light on or off and calls the corresponding functions. It then returns an updated HTML response.

Line 64: This line starts the server, and it specifies the IPv4 address of the WiFi connection. The server will listen for incoming requests from this address.

Enter a Continuous Loop

Main Loop

```
68 pool = setup_wifi_ap()
69 relay = setup_relay()
70 server = setup_server(pool, relay)
71 while True:
72     server.poll()
```

Line 68: Calls the **setup_wifi_ap** function to set up the WiFi Access Point (AP) while configuring a specific SSID and password.

Line 69: Calls the **setup_relay** function to set up a digital output pin ***GP22*** to control the relay. The relay is used to control the ON / OFF of the USB light stick in response to user interactions with the web interface.

Line 70: Calls the **setup_server** function to set up the HTTP server. It creates an instance of the HTTPServer class, configures routes, and starts listening for incoming HTTP requests.

Line 71 - 72: The main loop continuously polls the HTTP server for incoming requests. It keeps the server running and responsive to client interactions.

CHAPTER 8

Climate Control Greenhouse

Light Sensor & Temperature Humidity Sensor

In this chapter, we will embark on a journey to explore two fascinating sensors - the APDS9960 light sensor and the AHT20 temperature and humidity sensor - as we work towards creating a climate-controlled greenhouse using the EDU PICO.

Imagine having the power to monitor and adjust the conditions within a greenhouse to create an ideal environment for plants to thrive. With the APDS9960 and AHT20 sensors, we will 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.

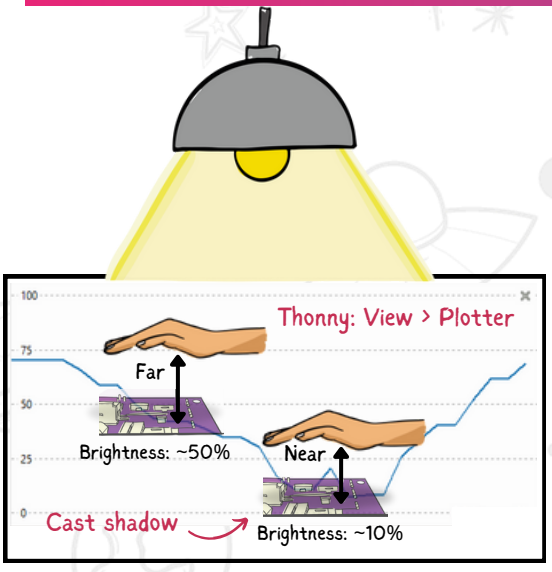
Lastly, we are going to top it off with two bonus activities on how you can establish a simple dashboard through IoT and logging data for analytic purpose too!

Introduction to Light Sensor

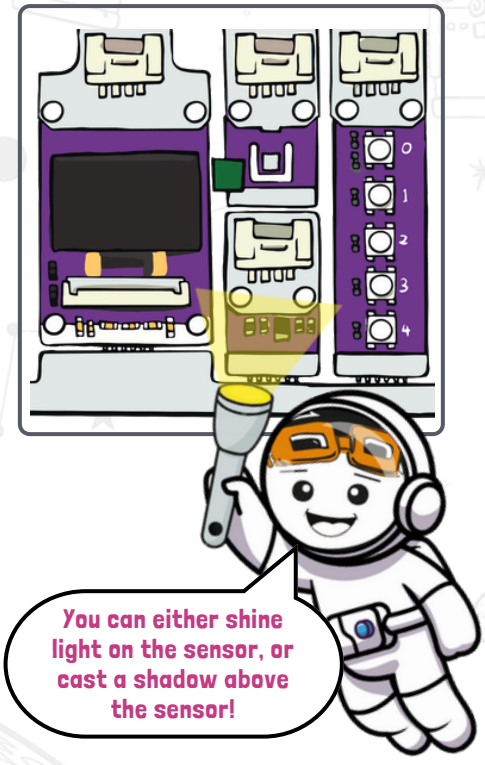
By now you have probably noticed the APDS9960 sensor on the EDU PICO offers several advantages for your project. Here's one last feature we have yet to explore from this powerful sensor, it is none other than the commonly used, light sensor. In this activity, we will learn how to measure different levels 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_apds9960.
- **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.



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

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

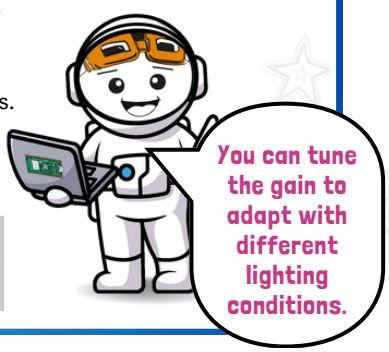
What the Code Does </>

Initialize Hardware Components

```
6 apds.enable_color = True
7 apds.color_gain = 2
```

Line 6 - 7: These lines enable colour sensing on the APDS9960 sensor and set the **color_gain** to **2**.
The colour gain affects the sensitivity of the colour sensors. A higher gain value makes the sensor more sensitive to changes in colour or brightness.

color_gain	0	1	2	3
Gain Multiplier	1x	4x	16x	64x



Enter a Continuous Loop

```
10      r, g, b, c = apds.color_data
11      brightness_percentage = (c / 65535) * 100
12      print(f"Brightness: {brightness_percentage:.1f}%")
```

Line 10: Read the colour data from the APDS9960 sensor, which includes the red (r), green (g), blue (b), and clear (c) values. The clear value represents the amount of ambient light, which can be used to estimate brightness.

Line 11: This line calculates the brightness level as a percentage by dividing the clear value (c) by the maximum possible clear value (**65535**) and then multiplying it by **100**. This conversion is used to represent the brightness in a human-readable form.

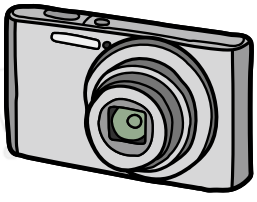
Line 12: Print the calculated brightness level to the console with one decimal point of precision.

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:

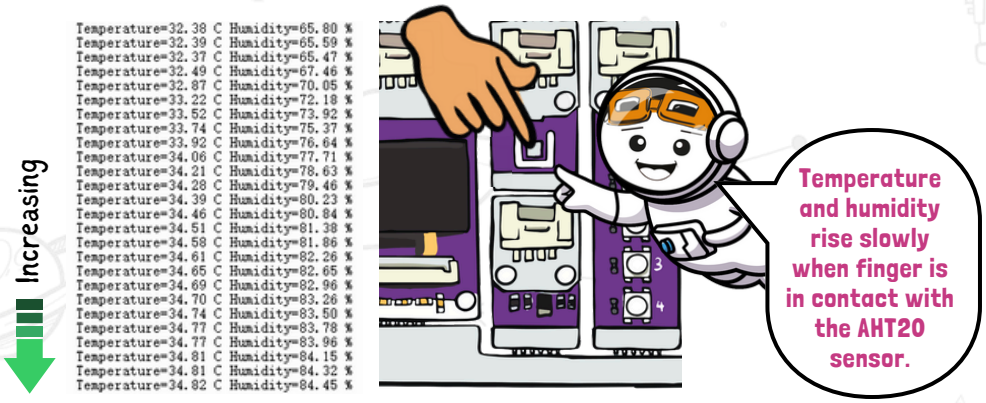
- Solar panels use light sensors to track the sun's movement, ensuring that they are always pointed at the sun to maximize energy generation. This process is called solar tracking.

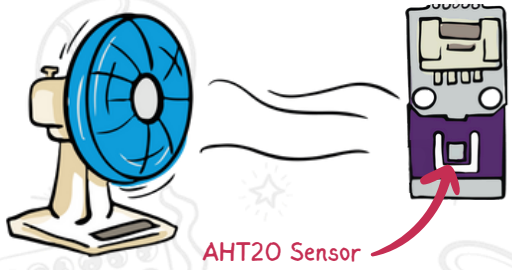
Introduction to Temperature & Humidity Sensor (AHT20)

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?

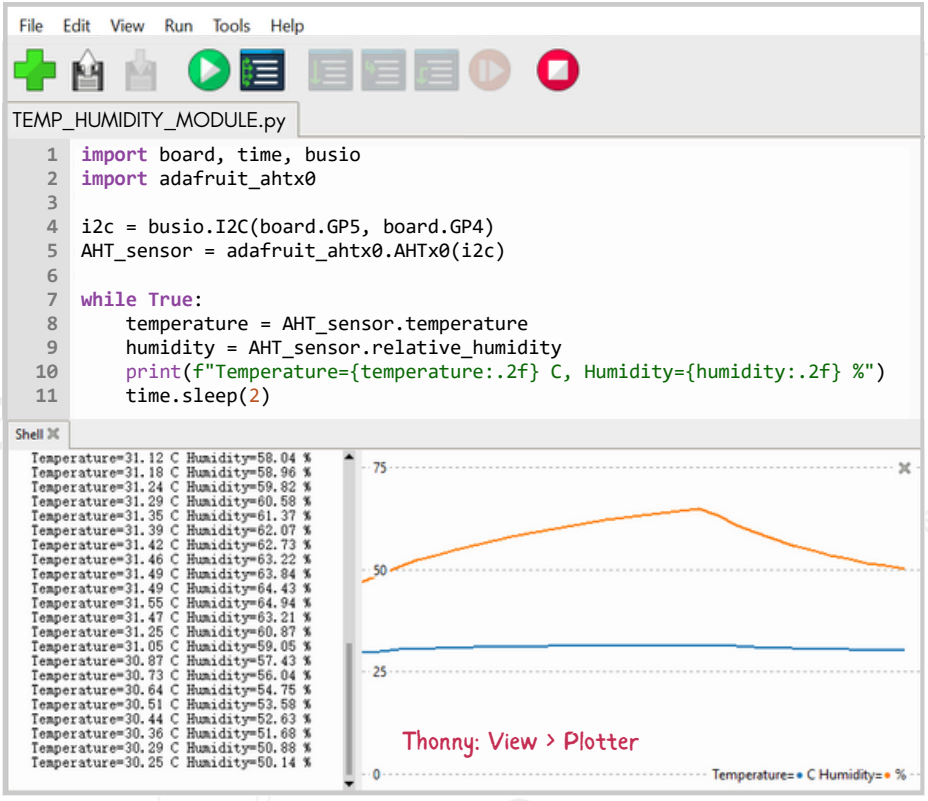
- **Libraries:** board, time, digitalio, adafruit_ahtx0.
- **AHT20 I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
- **Output:**
  - Print temperature in Celsius and relative humidity percentile in the shell console.
  - Enabling **view > plotter** in Thonny IDE will allow you to visualize 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: Provide support for the AHT20 temperature and humidity 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.

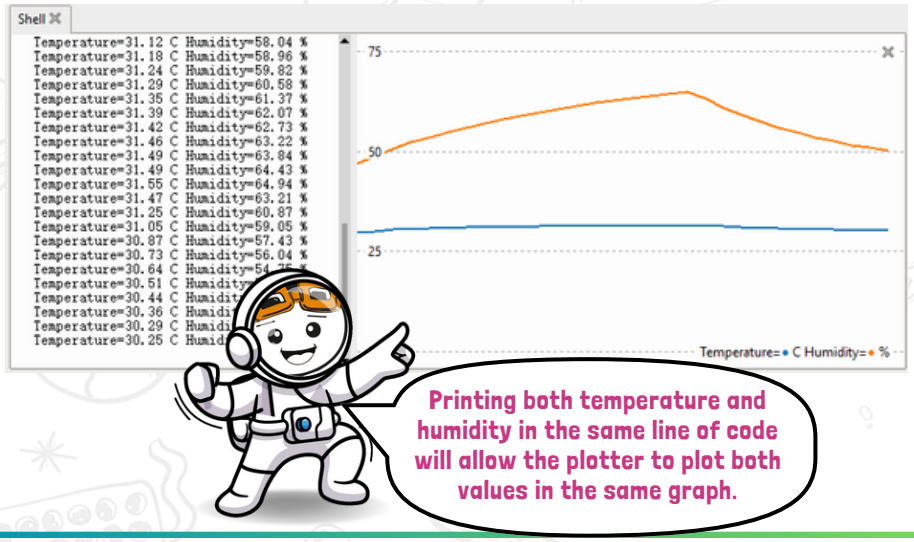
Enter a Continuous Loop

```
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: Reads the temperature data from the AHT sensor and stores it in the **temperature** variable.

Line 9: Reads the relative humidity data from the AHT sensor and stores 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.



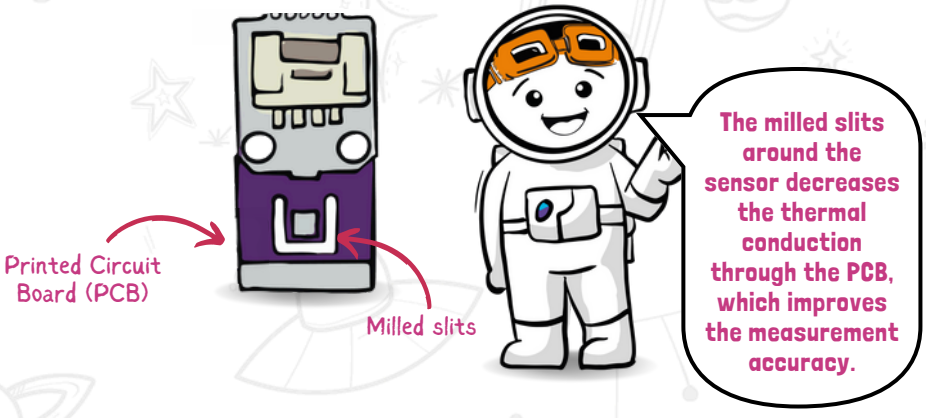
THE MORE YOU KNOW

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 white lines 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.

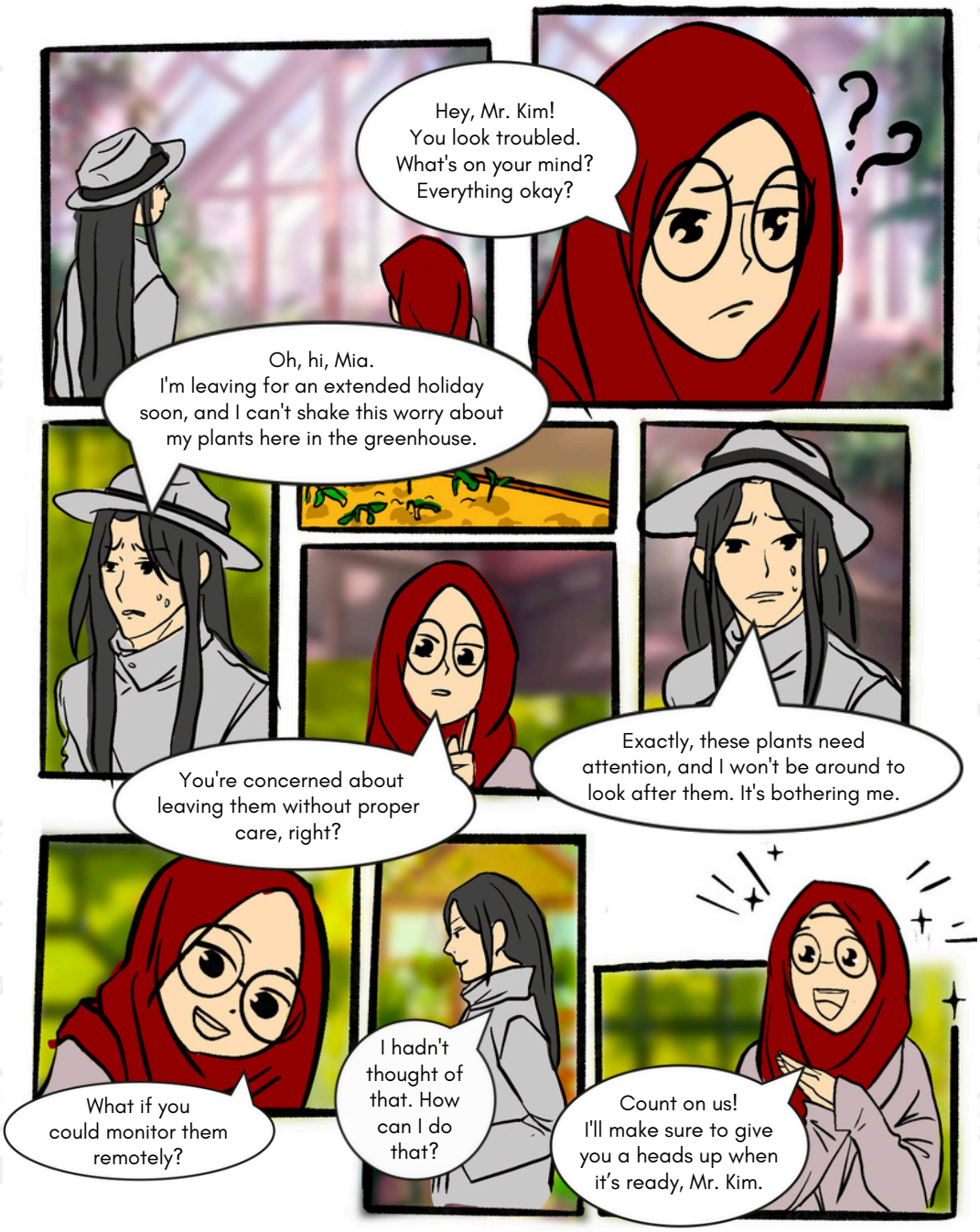


When the measurement frequency is too high, the temperature of the sensor module will heat up, which may affect the measurement accuracy. To keep the temperature from rising, it is recommended to include a 2 second interval for the data collection cycle.

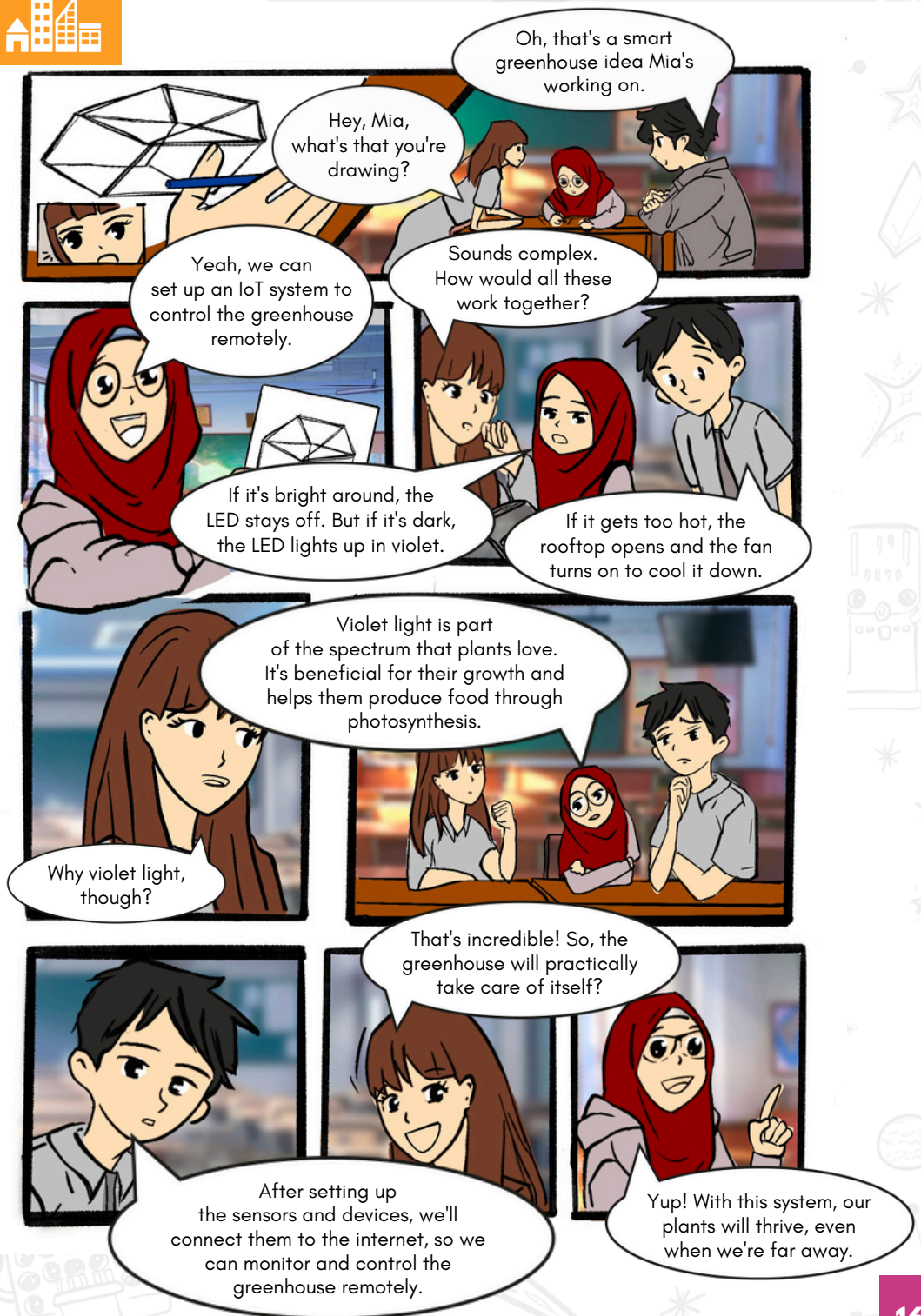
2 second interval

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

11 SUSTAINABLE CITIES AND COMMUNITIES

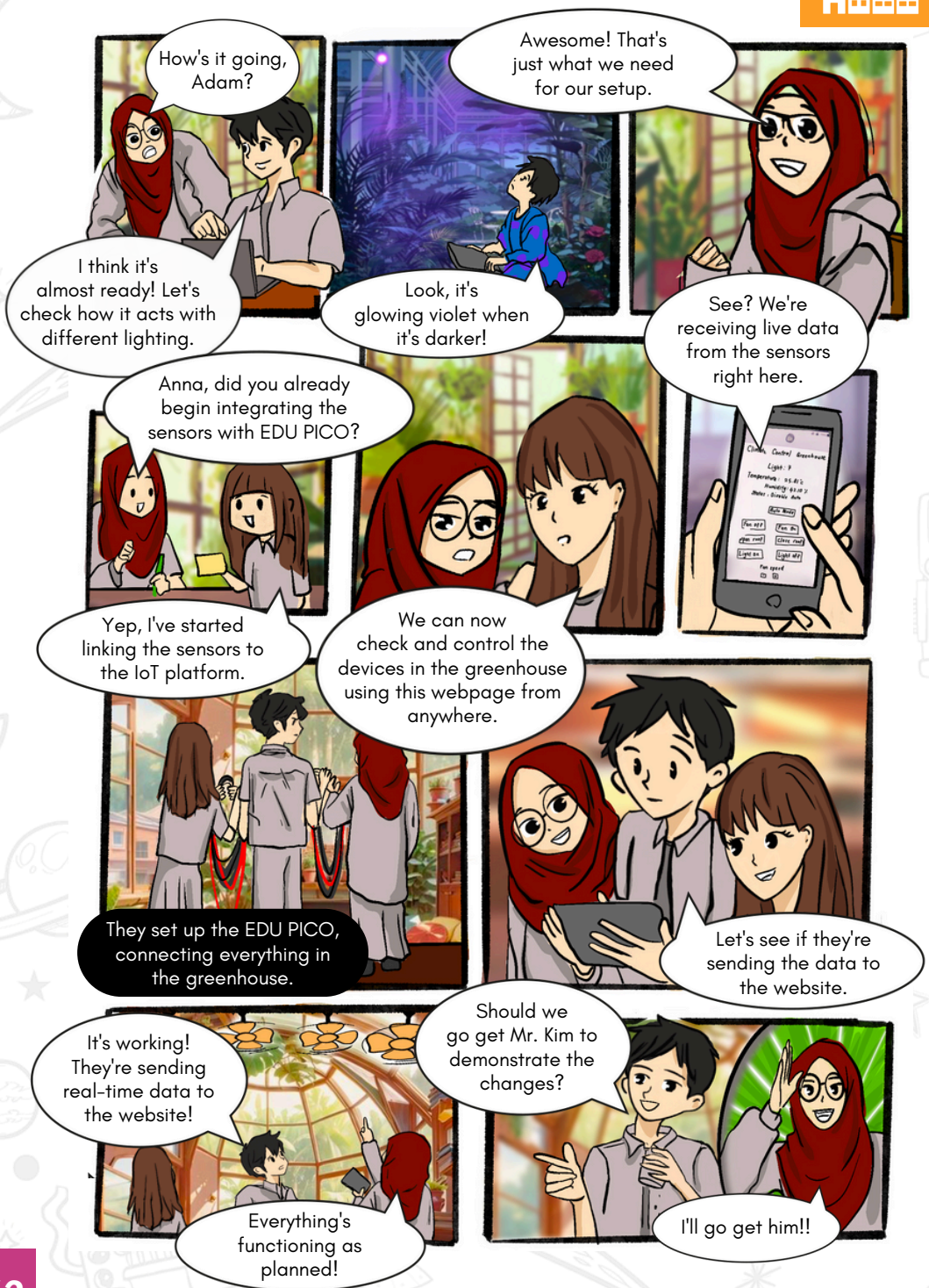


PROLOGUE: CLIMATE CONTROL GREENHOUSE

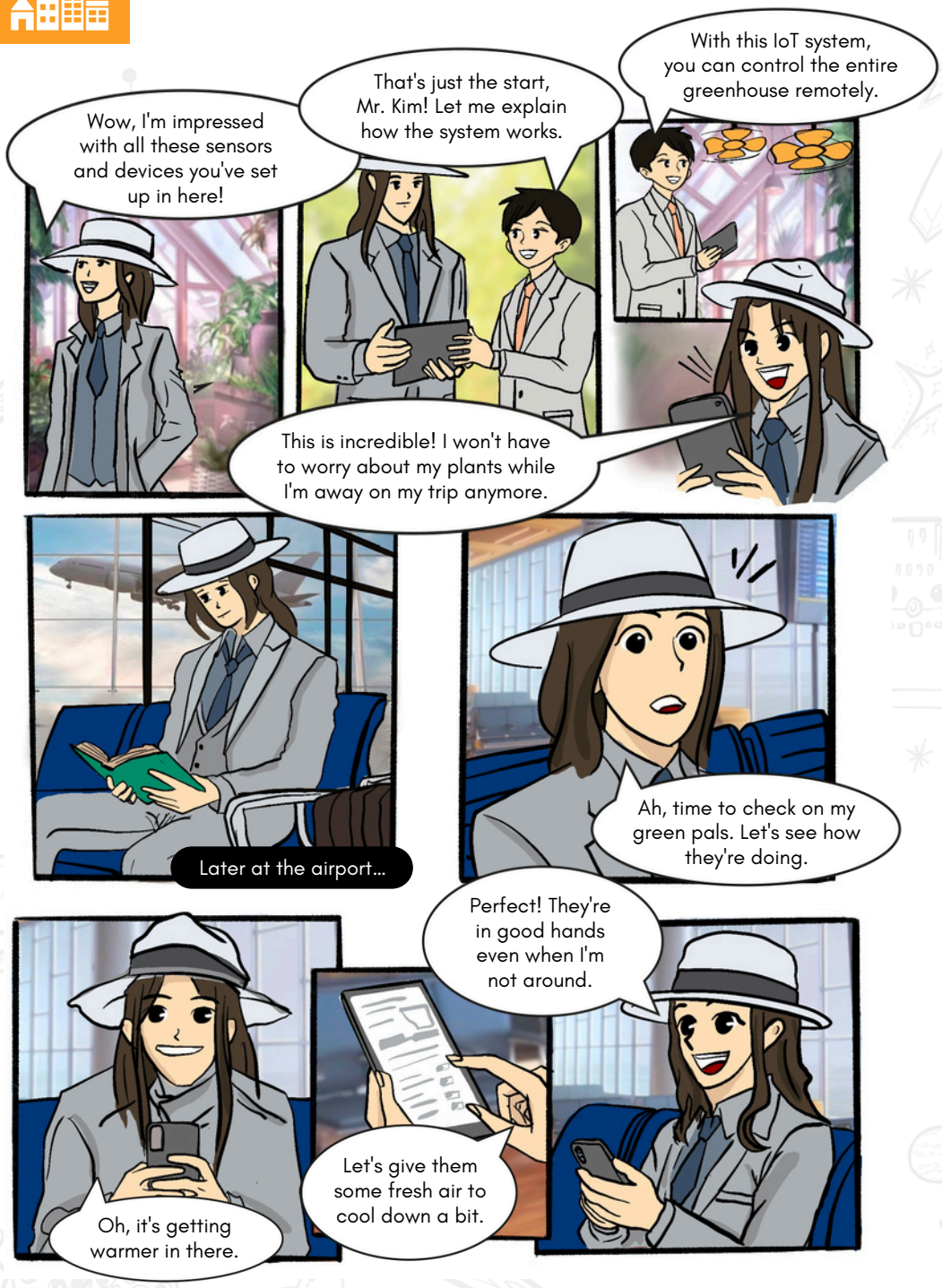


PROLOGUE: CLIMATE CONTROL GREENHOUSE

11 SUSTAINABLE CITIES AND COMMUNITIES



PROLOGUE: CLIMATE CONTROL GREENHOUSE

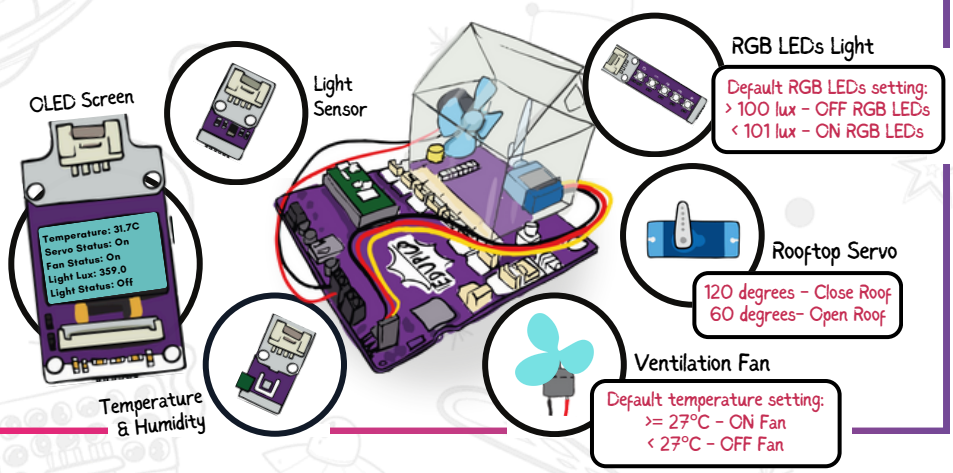


Climate Control Greenhouse

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 APDS9960 light sensor as input devices; RGB LEDs (for light), servo motor (for rooftop control), DC motor (for the fan), and OLED screen for printing the greenhouse status.

How Does This Activity Work?

- **Libraries:** board, busio, time, adafruit_ssd1306, neopixel, PWMOut adafruit_ahtx0, , adafruit_motor, adafruit_ssd1306, adafruit_apds9960.
- **I2C Pins Configuration:** SCL = ***GP5*** and SDA = ***GP4***.
  - APDS9960 light sensor, AHT20 temperature humidity sensor, and SSD1306 OLED share similar I2C pins configuration.
- **Input:**
  - Shine a light on the APDS9960 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.

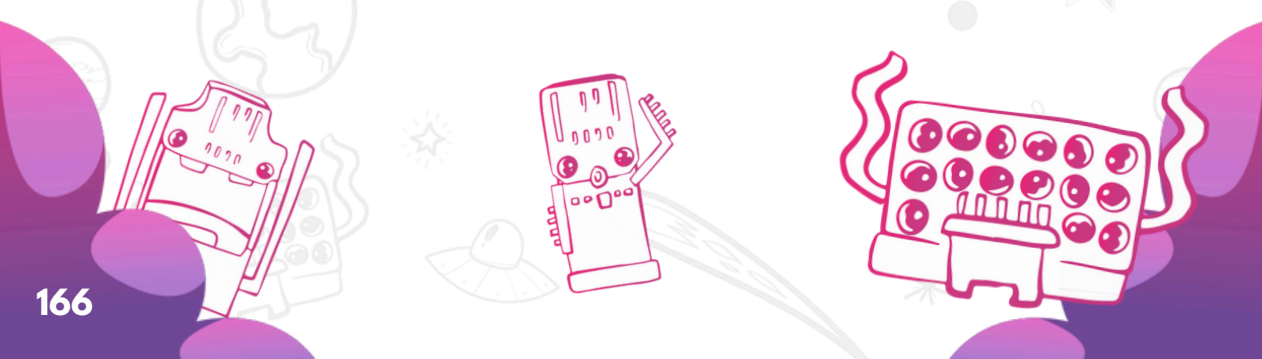


Code

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_apds9960.apds9960 import APDS9960
from adafruit_apds9960 import colorutility

i2c = busio.I2C(board.GP5, board.GP4)
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
oled.invert(True)
aht20 = adafruit_ahtx0.AHTx0(i2c)
apds = APDS9960(i2c)
apds.enable_color = True

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):
    while not apds.color_data_ready:
        time.sleep(0.005)
    r, g, b, c = apds.color_data
    light_lux = colorutility.calculate_lux(r, g, b)
    if light_lux < light_threshold:
        light_on()
    else:
        light_off()
    oled.text("Light Lux: {:.1f} ".format(light_lux), 5, 39, 1)

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()
```



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_apds9960.apds9960 import APDS9960
6  from adafruit_apds9960 import colorutility
```

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 (APDS9960), servo motor, DC motor, and RGB LEDs.

Initialize Hardware Components

Hardware Initialization

```
8  i2c = busio.I2C(board.GP5, board.GP4)
9  oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
10 oled.invert(True)
11 aht20 = adafruit_ahtx0.AHTx0(i2c)
12 apds = APDS9960(i2c)
13 apds.enable_color = True
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 8: Initializes I2C communication with ***GP5*** for SCL and ***GP4*** for SDA.

Line 9 - 13: Configure the OLED, temperature, humidity, and colour sensor using the same I2C bus and pins. Line 10 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 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

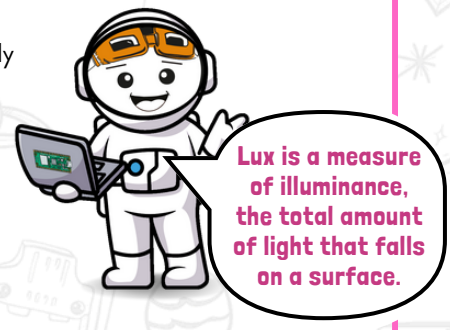
```
40 def temp_control(temp_threshold):
41     temperature = aht20.temperature
42     oled.text("Temperature: {:.1f} C".format(temperature), 5, 3, 1)
43     if temperature >= temp_threshold:
44         servo_on()
45         motor.throttle = 0.25
46         oled.text("Fan Status: On", 5, 27, 1)
47     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 screen.

Line 43 - 50: Activate the DC motor fan and servo motor when the temperature is above the **temp_threshold** set. The temperature threshold value is set at line 66.

Light Control Function

```
52 def light_control(light_threshold):
53     while not apds.color_data_ready:
54         time.sleep(0.005)
55     r, g, b, c = apds.color_data
56     light_lux = colorutility.calculate_lux(r, g, b)
57     if light_lux < light_threshold:
58         light_on()
59     else:
60         light_off()
61     oled.text("Light Lux: {:.1f} ".format(light_lux), 5, 39, 1)
```



Line 53 - 54: This loop waits until colour data is ready to be read from the APDS9960 sensor. It checks the **color_data_ready** property of the sensor. The loop pauses for **0.005** second in each iteration to avoid unnecessary CPU load while waiting.

Line 55 - 56: Read the values r, g, b, and c colour data and calculate the illuminance in lux.

Line 57 - 60: Calls **light_on()** function when the calculated lux value is below a preset threshold, and **light_off()** function if the lux value is above the threshold.

Line 61: 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 **light_lux**.

Enter a Continuous Loop

Main Loop

```
63 try:
64     while True:
65         oled.fill(0)
66         temp_control(temp_threshold=27)
67         light_control(light_threshold=100)
68         oled.show()
69         time.sleep(2)
```

Adjust this temperature value to control the output sensitivity.

Line 64 - 69: Continuously calls the **temp_control** function while passing a temperature threshold of **27** degrees Celsius, the code then calls the **light_control** function, passing a light threshold of **100** lux. These functions update the OLED display with the latest data every **2** seconds and control the fan and servo motor depending on the latest temperature and lux data measured.

Cleanup Operations

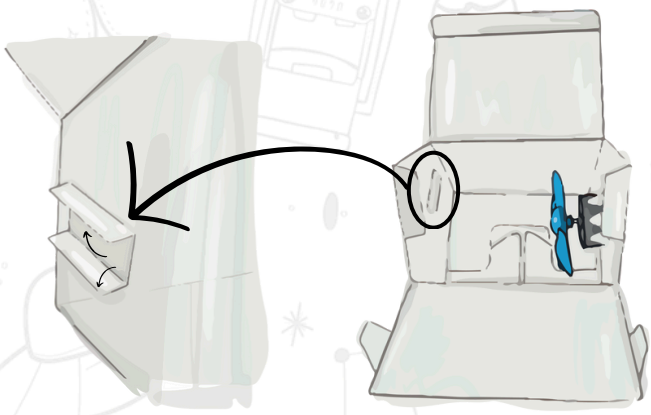
```
71 finally:
72     oled.fill(1)
73     oled.show()
74     print("deinit I2C")
75     i2c.deinit()
```

Line 71 - 75: The **finally** block ensures that cleanup operations are performed, regardless of whether an exception occurred or not during the main program execution. In this case, it clears the OLED display, prints a message, and deinitializes the I2C bus. These operations are essential for maintaining the integrity of the hardware and ensuring a clean exit of the program.

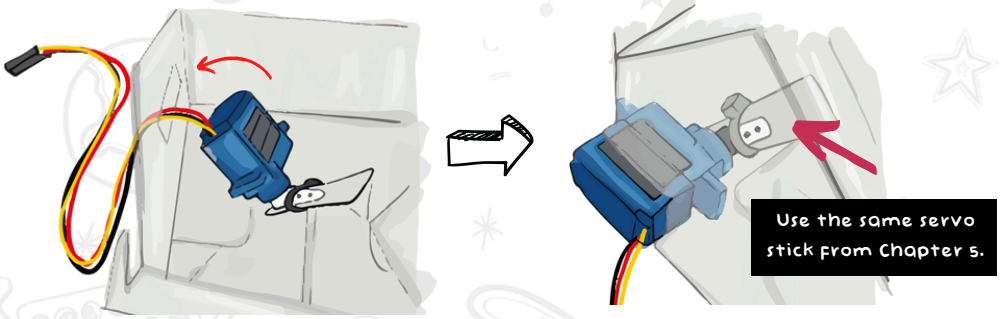
WHAT'S NEXT?

HOUSE ACCESSORY - SERVO ROOF

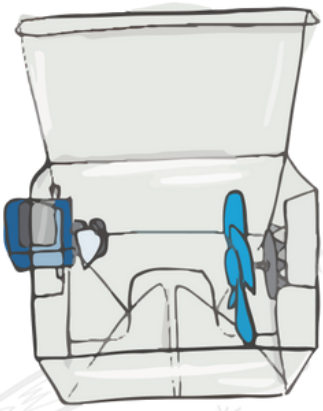
Step 1:
Locate the servo motor attachment flaps and flip it outwards as shown.



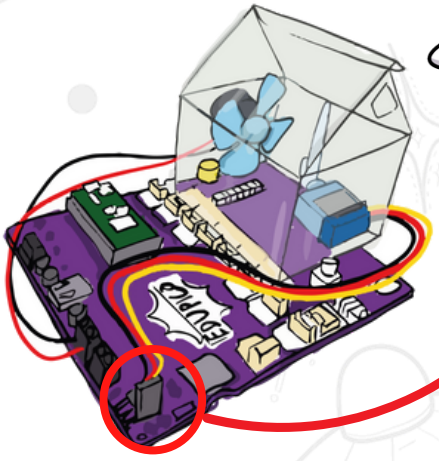
Step 2:
Attach the servo motor from inside out with the cable going through the box first.



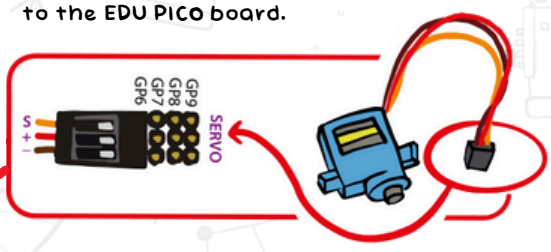
Step 3:
Close the front side of the roof back to it's original position.



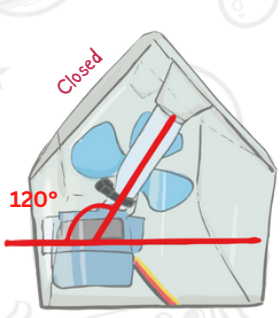
Step 4:
Give the back roof a gentle push downwards and inwards to the box.

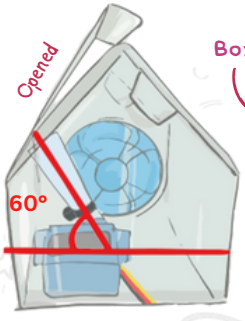


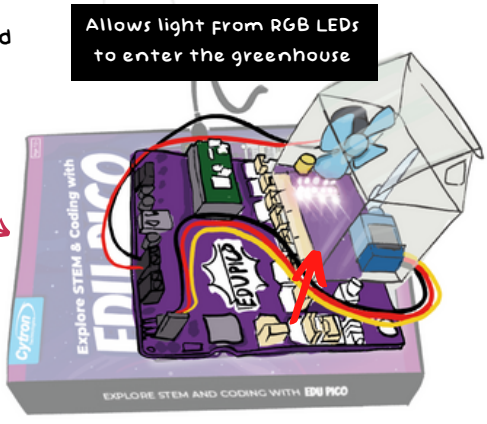
Step 5:
Connect the servo motor & DC motor to the EDU PICO board.



Step 6:
Test the servo motor horn position to open and close the roof. Open roof = 60°, close roof = 120°.



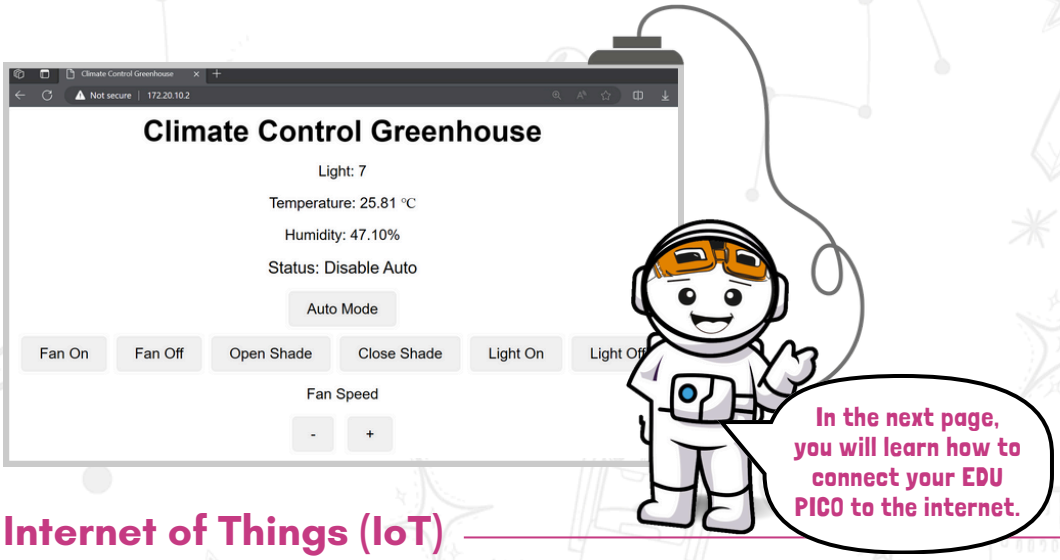




Final Placement

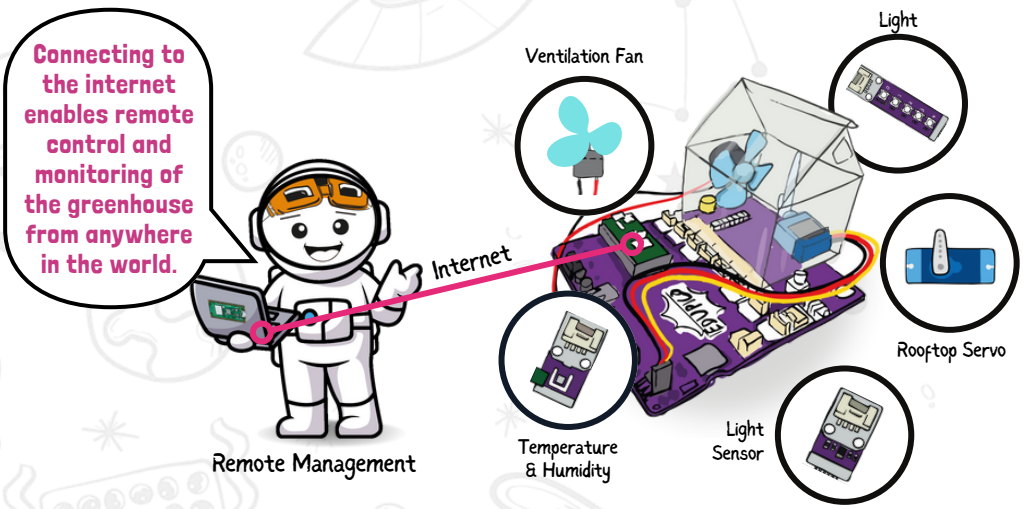
CHALLENGE

In this final chapter, we will test your programming skills, in both Python and HTML.
Set up the dashboard shown below. Your code should be able to enable **Auto Mode** and **Manual Mode**, allowing users to take control of the system when needed.



Internet of Things (IoT)

The Internet of Things (IoT) is made up of a vast network of physical devices, vehicles, appliances, and other objects that are connected via sensors, software, and internet connectivity. Imagine you have everyday things like your fridge or lights connected to the internet where they can communicate and share information. It's like giving regular devices a way to be smart and work together to make your life easier!

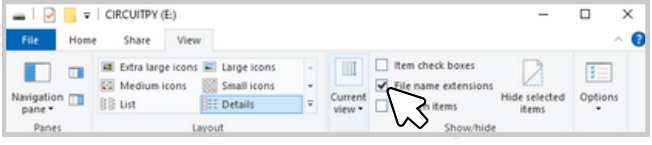


Bonus: Introduction to Internet of Things (IoT)

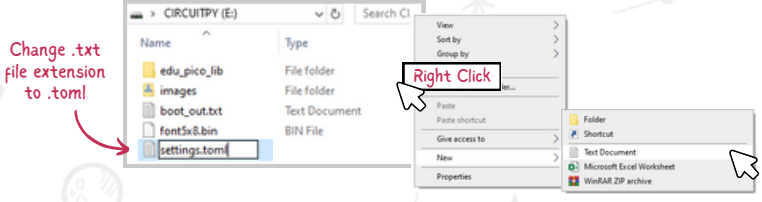
IoT stands for the Internet of Things. Imagine if everyday objects could connect to the internet and communicate with each other. IoT is all about linking things together via the internet to make them smarter and more useful. In this activity, we will turn the EDU PICO into an IoT-enabled device by reading the Raspberry Pi Pico W onboard temperature and controlling the EDU PICO's USB relay output through a webpage.

How Does This Activity Work?

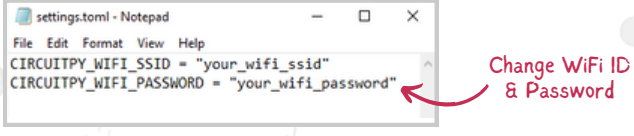
- **Libraries:** board, digitalio, wifi, socketpool, os, adafruit_httpserver, microcontroller.
- **USB Relay Configuration**:
  - Assign to ***GP22***, and connect USB light stick to the USB port.
- **Create WiFi configuration file "settings.toml":**
  - Enable View > **File name extensions** to show the file extension.



  - Create a new Text Document in the CircuitPython drive and name the Text Document as **"settings.toml"**.



  - Type the text below and replace **"your_wifi_ssid"** with your WiFi ID and **"your_wifi_password"** with your WiFi password.



  - Save the file, you're all set!

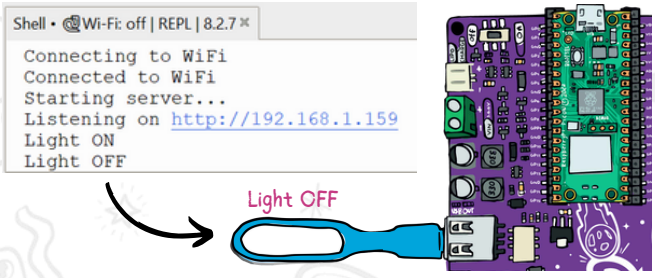
[Note: If you want to use AP-Mode (Chapter 7 Bonus) on the Raspberry Pi Pico W, make sure to delete the settings.toml file.]

- **Connect your PC to WiFi:**
  - Connect your PC to the same WiFi network as connected by the Raspberry Pi Pico W.
  - In this example, it's connected to **"My_WiFi_Network"**, which is also a mobile WiFi hotspot.

- **Output**:
  - The program begins by connecting to the WiFi with the SSID and Password preset in the **settings.toml** file.
  - Once the Raspberry Pi Pico W is connected to the WiFi, it will start the server and print "Starting server..." followed by "Listening on http://[IP_ADDRESS]" where **[IP_ADDRESS]** is the IP address of the WiFi network.

  - Type your IP address into your browser. The IP address is 192.168.1.159 as shown in the image below:

  - Clicking these buttons will send a signal to either turn ON or OFF the USB relay. Messages like "Light ON" and "Light OFF" will be printed on the shell console when the buttons are clicked.



Code

BONUS_IOT.py

```
import board, digitalio
import wifi, socketpool, os, microcontroller
from adafruit_httpserver import Server, Request, Response, POST

def setup_wifi():
    print("Connecting to WiFi")
    wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
                       os.getenv('CIRCUITPY_WIFI_PASSWORD'))
    print("Connected to WiFi")
    pool = socketpool.SocketPool(wifi.radio)
    return pool

def setup_relay():
    relay = digitalio.DigitalInOut(board.GP22)
    relay.direction = digitalio.Direction.OUTPUT
    return relay

def light_on(relay):
    print("Light ON")
    relay.value = True

def light_off(relay):
    print("Light OFF")
    relay.value = False

def pico_temp():
    return microcontroller.cpu.temperature

def webpage():
    Pico_Temp = pico_temp()
    html = f"""
    <!DOCTYPE html>
    <html>
    <head>
    <meta http-equiv="refresh" content="5">
    <title>USB Relay Control</title>
    </head>
    <body>
    <p>CPU-Temperature: {Pico_Temp:.2f} &#8451;</p>
    <p>USB Relay Light Control</p>
    <form accept-charset="utf-8" method="POST">
    <button class="button" name="Light On"
    value="light_on" type="submit">Light On</button></a>
    <button class="button" name="Light Off"
    value="light_off" type="submit">Light Off</button></a>
    </form>
    </body>
    </html>
    """
    return html
```

```
51  def setup_server(pool, relay):
52      server = Server(pool, "/static")
53
54      @server.route("/")
55      def base(request: HTTPRequest):
56          return Response(request, f"{webpage()}", content_type='text/html')
57
58      @server.route("/", POST)
59      def buttonpress(request: Request):
60          if request.method == POST:
61              raw_text = request.raw_request.decode("utf8")
62              if "light_on" in raw_text:
63                  light_on(relay)
64              if "light_off" in raw_text:
65                  light_off(relay)
66              return Response(request, f"{webpage()}", content_type='text/html')
67
68      print("Starting server...")
69      server.start(str(wifi.radio.ipv4_address))
70      print("Listening on http://%s" % wifi.radio.ipv4_address)
71      return server
72
73  pool = setup_wifi()
74  relay = setup_relay()
75  server = setup_server(pool, relay)
76  while True:
77      server.poll()
```

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
2  import wifi, socketpool, os, microcontroller
3  from adafruit_httpserver import Server, Request, Response, POST
```

Line 2: The **microcontroller** library provides access to Raspberry Pi Pico W on-board features. In this case, it provides the temperature data from the microcontroller's CPU. The OS library is used to retrieve the values of environment variables. In this example, the program allows retrieval of the value from the 'CIRCUITPY_WIFI_SSID' and 'CIRCUITPY_WIFI_PASSWORD' environment variables.

Define Custom Functions

WiFi Setup and Connection

```
5  def setup_wifi():
6      print("Connecting to WiFi")
7      wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
                          os.getenv('CIRCUITPY_WIFI_PASSWORD'))
8      print("Connected to WiFi")
9      pool = socketpool.SocketPool(wifi.radio)
10     return pool
```

Line 7: Connect the Raspberry Pi Pico W to a WiFi network using the SSID and password specified in the **settings.toml** file.

Line 9: Creates and returns a socket pool for communication.

Initialize USB Relay & Pico Onboard Temperature Sensor

```
12 def setup_relay():
13     relay = digitalio.DigitalInOut(board.GP22)
14     relay.direction = digitalio.Direction.OUTPUT
15     return relay
16
17 def light_on(relay):
18     print("Light ON")
19     relay.value = True
20
21 def light_off(relay):
22     print("Light OFF")
23     relay.value = False
24
25 def pico_temp():
26     return microcontroller.cpu.temperature
```

Line 12 - 15: Initializes a digital pin ***GP22*** for the relay and sets it as an output.

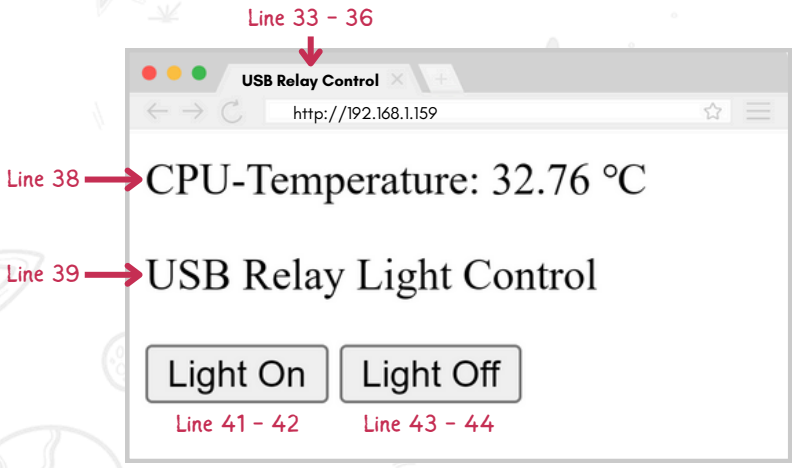
Line 17 - 19: Controls the ON / OFF of the USB light stick by setting the value of the digital pin connected to the relay.

Line 25 - 26: Retrieve the Raspberry Pi Pico W's CPU temperature.

Webpage HTML

```
28 def webpage():
29     Pico_Temp = pico_temp()
30     html = f"""
31     <!DOCTYPE html>
32     <html>
33     <head>
34     <meta http-equiv="refresh" content="5">
35     <title>USB Relay Control</title>
36     </head>
37     <body>
38     <p>CPU-Temperature: {Pico_Temp:.2f} &#8451;</p>
39     <p>USB Relay Light Control</p>
40     <form accept-charset="utf-8" method="POST">
41     <button class="button" name="Light On"
42     value="light_on" type="submit">Light On</button></a>
43     <button class="button" name="Light Off"
44     value="light_off" type="submit">Light Off</button></a>
45     </form>
46     </body>
47     </html>
48     """
49     return html
```

Line 28 - 49: This function generates an HTML webpage that displays the Raspberry Pi Pico W's CPU temperature and ON / OFF buttons to control the USB relay.



Web Server Setup

```
51 def setup_server(pool, relay):
52     server = Server(pool, "/static")
53
54     @server.route("/")
55     def base(request: Request):
56         retrurn Response(request, f"{webpage()}", content_type='text/html')
57
58     @server.route("/", POST)
59     def buttonpress(request: Request):
60         if request.method == POST:
61             raw_text = request.raw_request.decode("utf8")
62             if "light_on" in raw_text:
63                 light_on(relay)
64             if "light_off" in raw_text:
65                 light_off(relay)
66             return Response(request, f"{webpage()}", content_type='text/html')
67
68     print("Starting server...")
69     server.start(str(wifi.radio.ipv4_address))
70     print("Listening on http://%s" % wifi.radio.ipv4_address)
71     return server
```

Line 51 - 52: Creates an instance of the Server class with the provided pool (socket pool) and a static route ("/static"). The static route may be used for serving static files like stylesheets or images.

Line 54 - 56: Generates an HTML response with the HTML content using the **webpage()** function and sends it back to the client.

Line 58 - 66: The **buttonpress** function is called when a POST request is received. It checks if the request contains data related to turning the light on or off and calls the corresponding functions. It then returns an updated HTML response.

Line 69: This line starts the server, and it specifies the IPv4 address of the WiFi connection. The server will listen for incoming requests from this address.

Enter a Continuous Loop

Main Loop

```
73 pool = setup_wifi()
74 relay = setup_relay()
75 server = setup_server(pool, relay)
76 while True:
77     server.poll()
```

Line 73 - 75: Sets up the WiFi connection, relay, and HTTP server.

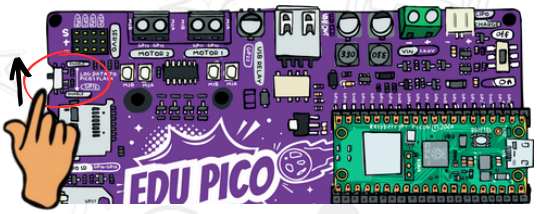
Line 76 - 77: Continuously polls the server to handle incoming requests. Keeping the server running and responsive to client interactions.

Bonus: Introduction to Data Logging

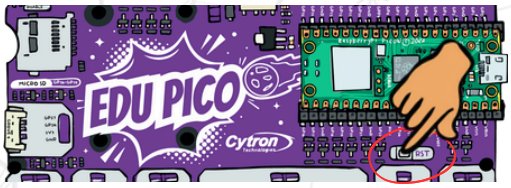
In this bonus, we will learn how to use CircuitPython to read the Raspberry Pi Pico W internal temperature data and write it to a file on the CircuitPython drive. This will enable you to create your own temperature data logger.

How Does This Activity Work?

- **Libraries:** boot.py, board, digitalio, time, microcontroller, os.
- **Data Logging Configuration File:**
  - Create a new Python script named **boot.py** and type in the code provided.
  - Save **boot.py** in the CircuitPython drive.
  - Flip the **'LOG DATA TO PICO'S FLASH'** switch to **'ENABLE'**.
    - NOTE: The CircuitPython drive will become non-writable when data logging mode is enabled. This means you won't be able to save, create a new file, or delete files in the CircuitPython drive.



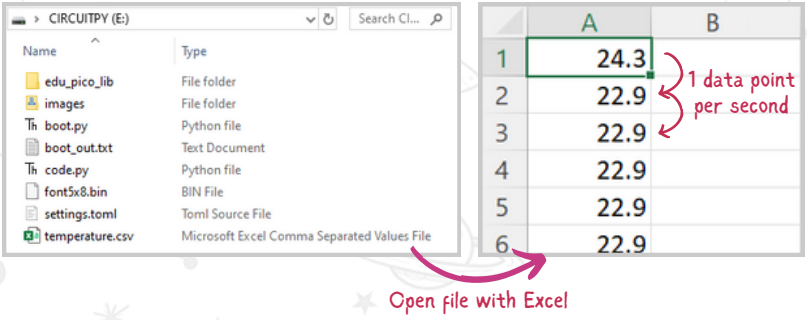
  - Restart your EDU PICO by pressing the Reset (RST) button.



  - The **boot_out.txt** file will appear in the CIRCUITPY drive, you will see **'boot.py output:'** inside the text file.

- Execute the **BONUS_DATALOGGING.py** code to start the data logger. The Pico will record 1 temperature data point every 1 second.

- **Output:** After resetting the EDU PICO, a file named **temperature.csv** will appear in the CircuitPython drive.



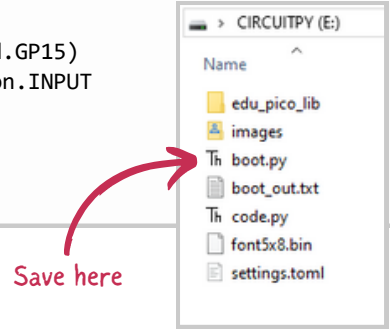
Code

First, you will need to remount the storage by saving the **boot.py** with the code shown below. Save the code in the CircuitPython root directory drive.

boot.py

boot.py

```
1  import board
2  import digitalio
3  import storage
4
5  write_pin = digitalio.DigitalInOut(board.GP15)
6  write_pin.direction = digitalio.Direction.INPUT
7  write_pin.pull = digitalio.Pull.UP
8
9  if not write_pin.value:
10     storage.remount("/", readonly=False)
```



Main Code

BONUS_DATALOGGER.py

```
1  import board, digitalio, time, microcontroller, os
2
3  led = digitalio.DigitalInOut(board.LED)
4  led.switch_to_output()
5
6  file_name = "temperature.csv"
7  max_file_size = 400000
8
9  with open(file_name, "a") as datalog:
10     while True:
11         file_size = os.stat(file_name)[6]
12         if file_size < max_file_size:
13             temp = microcontroller.cpu.temperature
14             datalog.write("{0:.1f}\n".format(temp))
15             datalog.flush()
16             led.value = not led.value
17             time.sleep(1)
18         else:
19             led.value = True
```

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

What the Code Does </>

Configure File and LED Pin

```
3  led = digitalio.DigitalInOut(board.LED)
4  led.switch_to_output()
5
6  file_name = "temperature.csv"
7  max_file_size = 400000
```

Line 3 - 4: Initialize a digital output pin connected to the onboard LED of the Raspberry Pi Pico W.

Line 6 - 7: Set up the file name for the temperature log ("temperature.csv") and define the maximum file size in bytes (**400kB**).

[Note: The Raspberry Pi Pico W has a limited onboard storage, hence setting the 400kB limit will ensure the data logging doesn't exceed 400kB.]

Main Loop

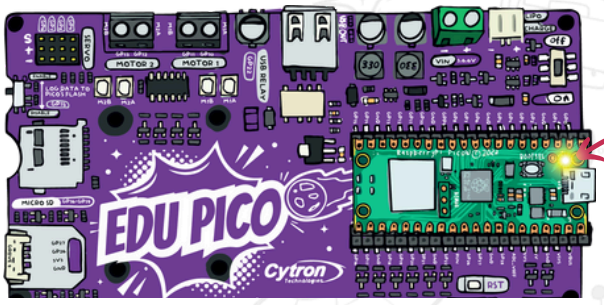
```
9  with open(file_name, "a") as datalog:
10     while True:
11         file_size = os.stat(file_name)[6]
12         if file_size < max_file_size:
13             temp = microcontroller.cpu.temperature
14             datalog.write("{0:.1f}\n".format(temp))
15             datalog.flush()
16             led.value = not led.value
17             time.sleep(1)
18         else:
19             led.value = True
```

Line 9 - 10: Open the file in append mode and enter a loop for continuous logging.

Line 11 - 12: Check if the current file size is below the specified maximum size.

Line 13 - 15: If the file size is within the limit, it reads the CPU temperature, writes it to the CSV file, and flushes the file to ensure data is written immediately.

Line 16: Toggles the state of the onboard LED, providing a visual indication that data is being logged.



Line 17: Introduces a 1 second delay between temperature readings to control the logging frequency.

Line 18 - 19: If the file size exceeds the limit, the LED is turned on continuously, indicating that logging is temporarily disabled due to the file size limit being reached.

Guidebook Summary

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.
- ☐ save, open, and edit Python .py files in EDU PICO.
- ☐ import CircuitPython libraries.

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.
- ☐ create and use variables.
- ☐ use while loop.
- ☐ use a conditional if statement.

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.
- ☐ program using If.. elif conditions.
- ☐ setup dictionaries and for loops.
- ☐ create and use functions.

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

In this chapter, we learn to:

- ☐ program EDU PICO to light up RGB LEDs in different colours.
- ☐ read colour data with colour sensor.
- ☐ setup lists to store multiple items.

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 data with Thonny's plotter.
- ☐ construct a Trashbot smart bin with card box accessories.

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

In this chapter, we learn to:

- ☐ program EDU PICO 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.

Chapter 7: Smart Classroom (DC Motor and Relay)

In this chapter, we learn to:

- ☐ program EDU PICO to control a DC motor – spinning direction and speed control.
- ☐ turn ON and OFF a USB switch relay.
- ☐ program EDU PICO's Raspberry Pi Pico W into a WiFi access point for IoT applications.

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.
- ☐ perform basic data logging on Raspberry Pi Pico W local storage.

*Tick the check box if you've completed the learning outcome; otherwise return to the chapter to revise.

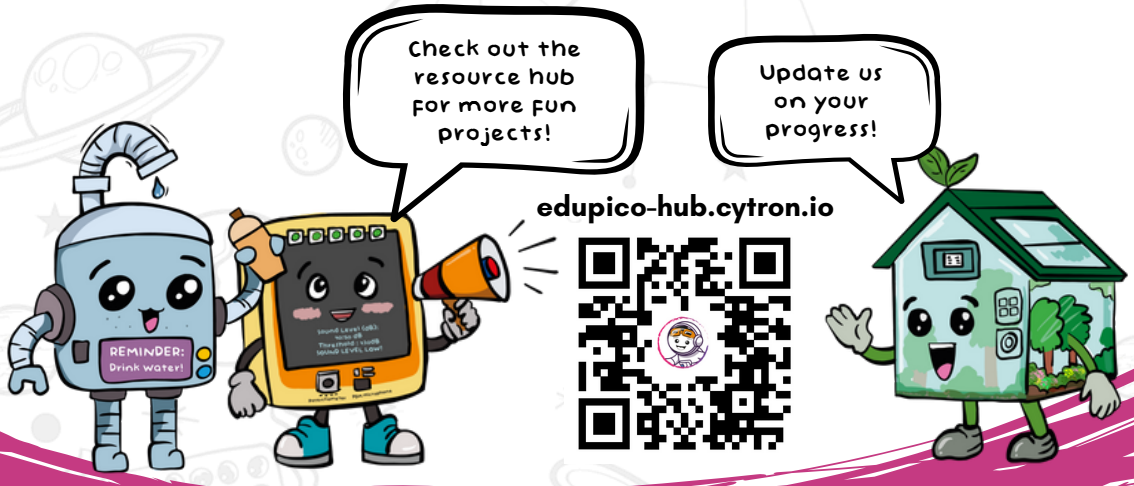
CONGRATULATIONS!

As you reach the end of this guidebook, the EDU PICO 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 stepping stone to a world filled with endless possibilities. As you continue to pursue your passion for learning, experimenting, and innovating, remember that the journey doesn't end here—it evolves into new opportunities and discoveries.

Once again, congratulations on completing this chapter of your educational journey and we can't wait to see what you will be building with the EDU PICO!

Best Wishes,
Adam the astronaut & EDU PICO team



EDU PICO