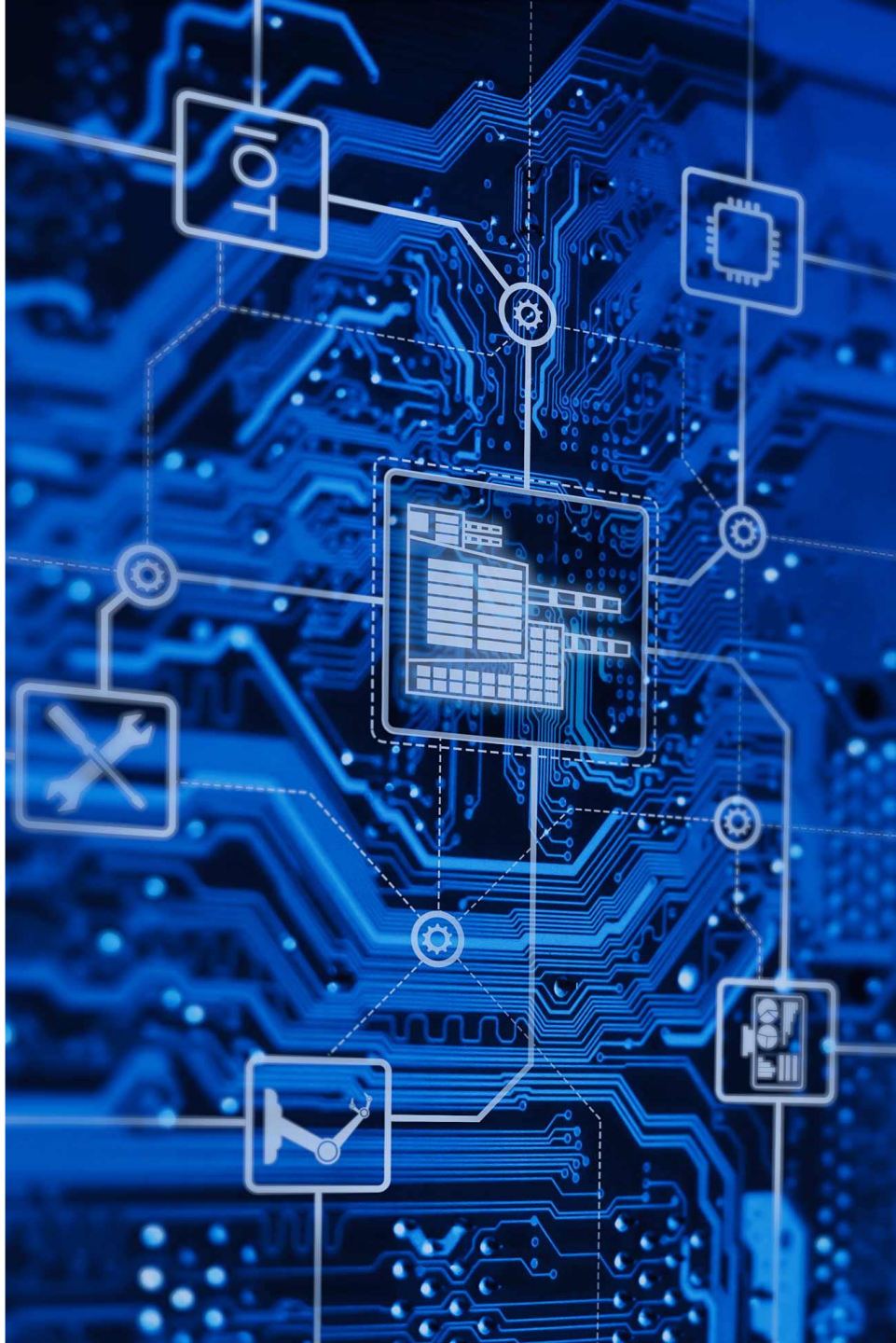
PROJECT NUMBER: 2023-1-PL01-KA220-SCH-000154043

Bringing the Internet of Things in school education as a tool to address 21st century challenges

---

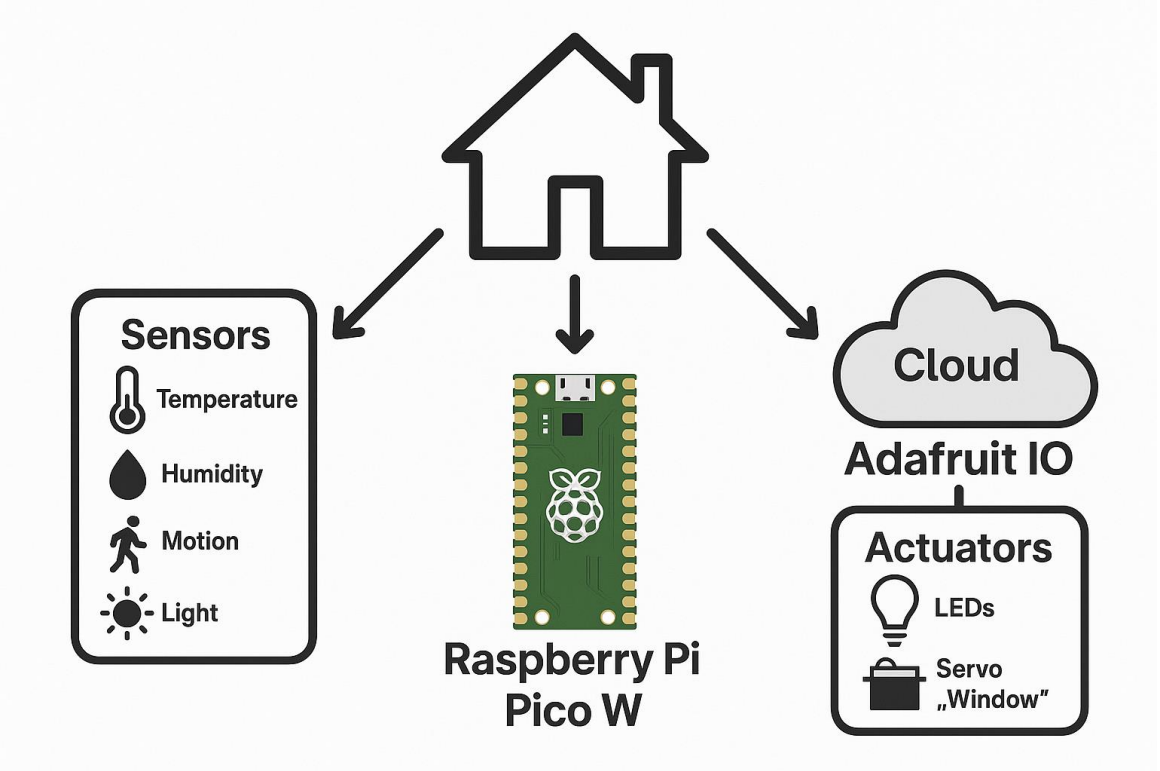
SmartHome project: Building a SmartHome ecosystem using IoT devices



Co-funded by the European Union

What is the SmartHome project?

1. Students build a mini Smart Home system
2. Use Raspberry Pi Pico W as the main microcontroller
3. Integrate sensors (DHT11, PIR, LDR) and actuators (LEDs, servo motor)
4. Program everything with MicroPython in Thonny
5. Connect to the Adafruit IO cloud platform to monitor & control the system online.



Key Learning Objectives

By the end of the project, students will:

1. Get familiar with programming in MicroPython
2. Learn how to connect electronics and sensors on a breadboard
3. Create an online application to control inputs/outputs
4. Understand Smart Home and IoT devices concepts
5. Interpret simple sensor readings: temperature, humidity, light, motion

Learning Pathway & Time Plan (1/2)

- Total duration: 7-8 class periods (≈6.5-7.5 hours)
- Implemented in 2 levels and 8 sessions
- Recommended pacing: 2-3 sessions per week over 3-4 weeks

Level 1

- Session 1: Intro to IoT & Smart Home concepts
- Session 2: Overview of Raspberry Pi Pico W & blink project
- Session 3: MicroPython basics & components (breadboard, resistors, sensors)
- Session 4: Connect & program DHT11, PIR, LDR
- Session 5: Connect & program LEDs & servo motor

Learning Pathway & Time Plan (2/2)

Level 2

- Session 6: Complete circuit & Adafruit IO integration
- Session 7: Testing, debugging, documentation
- Session 8: Student presentations

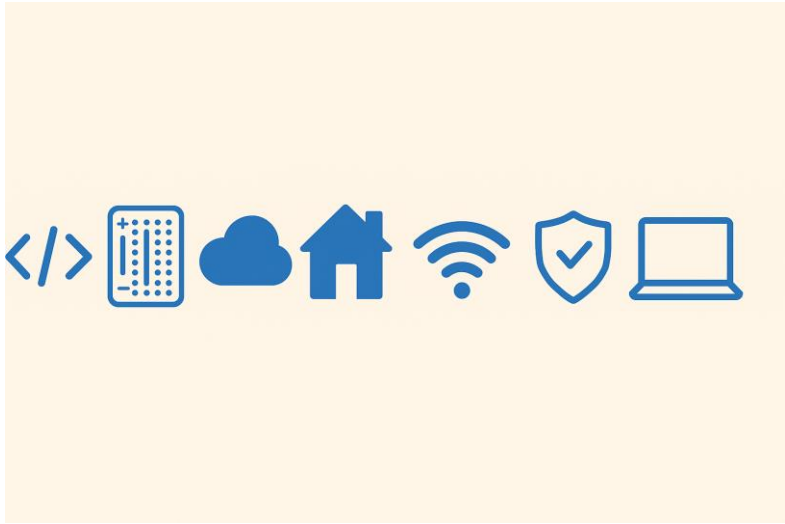
Pedagogical Approach

- Designed for **group work** (3–4 students per team)
- Roles can rotate: Coder, Builder, Tester, Documenter
- Combines **theory, guided practice, and open-ended problem-solving**
- Fosters **computational thinking**, **collaboration**, and **communication**
- Discussion prompts:
  - IoT benefits & risks
  - Energy efficiency & sustainability
  - Data privacy & security in smart homes

Student & Classroom Prerequisites

Student prerequisites:

- Basic programming concepts (variables, loops, conditionals, functions)
- Familiarity with IDE/editor, terminal/shell
- Basic understanding of circuits and safe breadboard use



Teacher & classroom prerequisites:

- Thonny installed on lab PCs/laptops
- MicroPython firmware flashed to Pico W
- School Wi-Fi that allows Pico W connection (2.4 GHz, WPA2)
- Classroom strategy for an Adafruit IO account & feeds
- Short safety briefing (voltages, current limits)

Discussion Points for Teachers

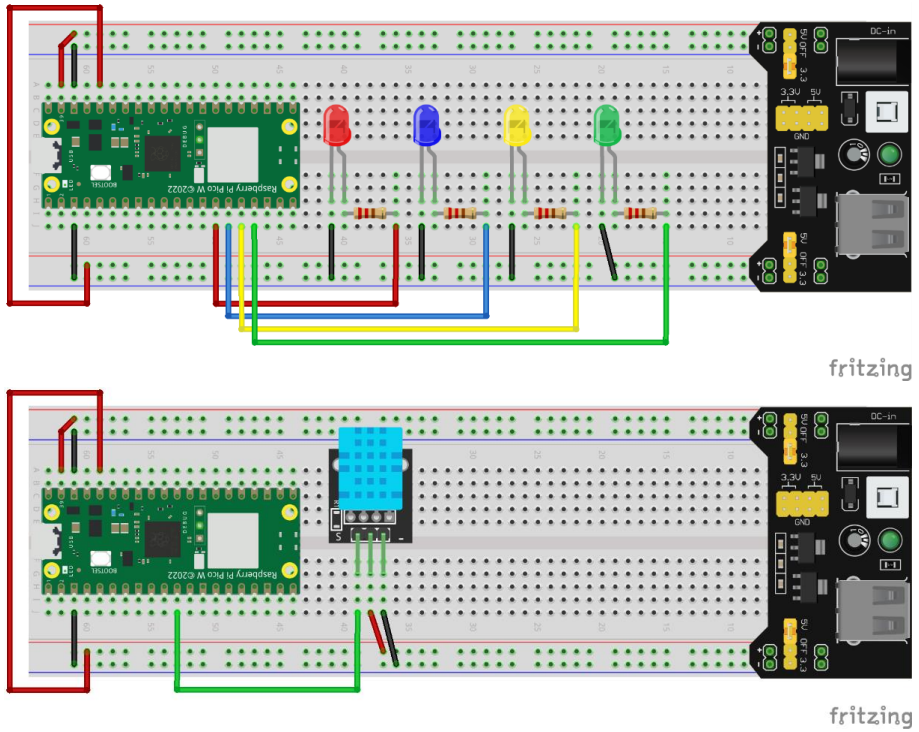
Teachers can use these points when they see fit during sessions with the students.

- How does IoT transform traditional homes?
- How can Smart Home systems improve convenience, efficiency, security?
- How can Smart Homes contribute to environmental sustainability?
- What are the risks around data privacy and security?
- What are good practices for securing IoT devices?

Level 1 – RPi Pico W Setup & Intro

Sessions 1–5 focus on:

- Introducing **IoT & Smart Home concepts**
- Setting up **Raspberry Pi Pico W** & running the first blink.py
- Learning **MicroPython basics** and core syntax
- Understanding key components: **breadboard, resistors, jumper wires, sensors, LEDs, servo**
- Reading data from **DHT11, PIR, LDR** and controlling **LEDs & servo**



Level 1 – Hardware & Software

Hardware

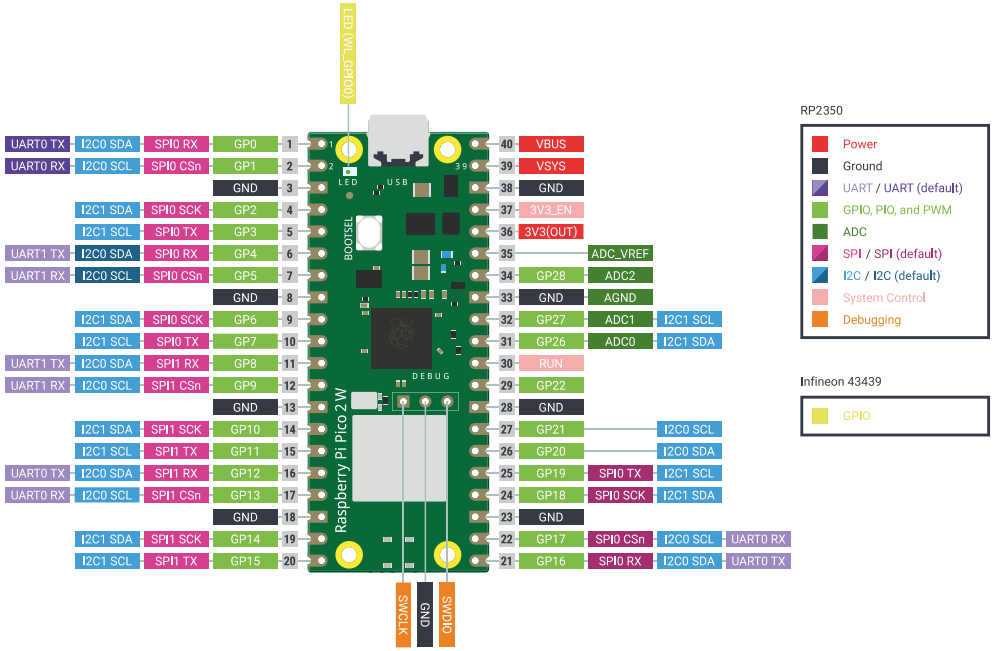
- 1 × Raspberry Pi Pico W
- 1 × Breadboard
- 1 × MB-102 power supply
- 6 × AA batteries
- Sensors: DHT11, PIR (HC-SR501), LDR photoresistor
- Actuators: 2 × LEDs, SG-90 servo motor
- Resistors, jumper cables

Software & Platforms

- Thonny IDE & MicroPython firmware
- MicroPython libraries (e.g., dht.py, servo.py, iot4schools.py software

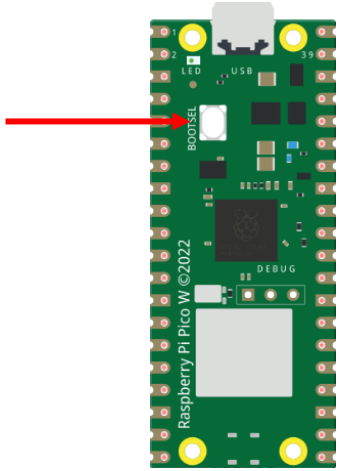
Level 1 – What is the Raspberry Pi Pico W

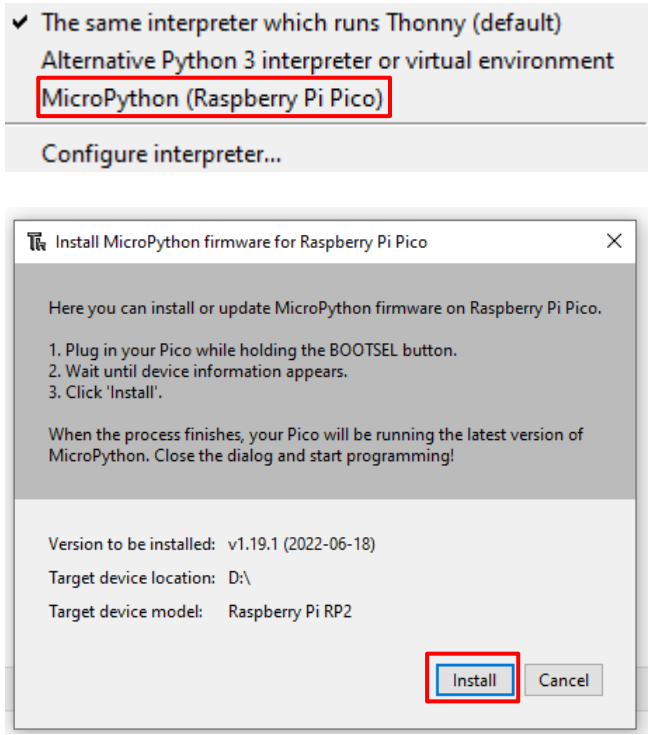
- Low-cost microcontroller with Wi-Fi
- Programmed using MicroPython
- Connects via micro-USB to computer
- BOOTSEL button used to flash firmware
- GPIO pins used for sensors and actuators



Level 1 – Installing MicroPython on Pico W

- Connect Pico W to PC while holding BOOTSEL
- It appears as a USB drive
- Copy the MicroPython UF2 file to the Pico drive
- Open Thonny, choose the MicroPython (Raspberry Pi Pico) interpreter
- Confirm the Pico responds in the Shell / REPL





**for detailed step-by-step instructions, refer to the teacher manual*

Level 1 – First program `blink.py`

The goal is to toggle the onboard LED.
Included concepts:

- Importing modules (`machine, time`)
- Configuring a Pin as output
- Using an infinite loop (`while True`)
- Using `sleep()` to create a delay

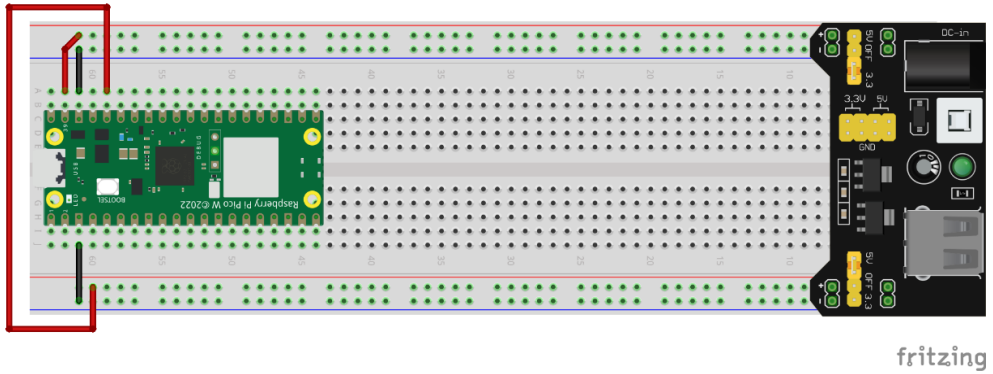
Write the following code and save your program as `blink.py`

```
from machine import Pin
from time import sleep
led = Pin("LED", Pin.OUT)
while True:
        led.toggle()
        sleep(1)
```

Level 1 – Power & Breadboard Setup

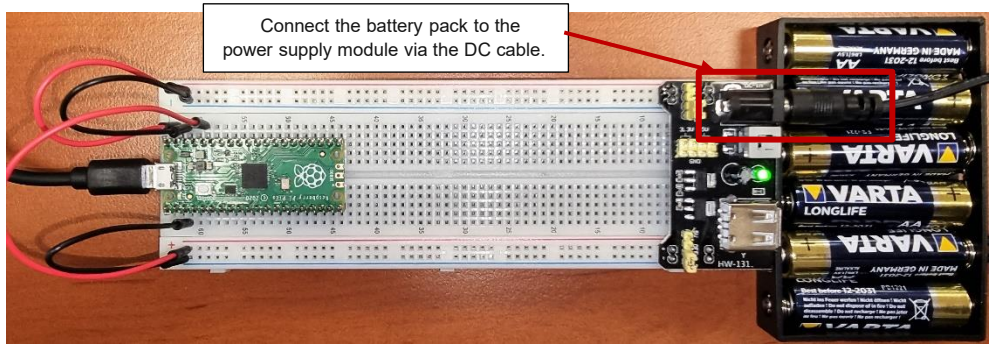
You will need:

- Raspberry Pi Pico W
- Breadboard
- MB-102 power module
- 6× AA battery pack
- Jumper wires (male-male, male-female)



Key ideas:

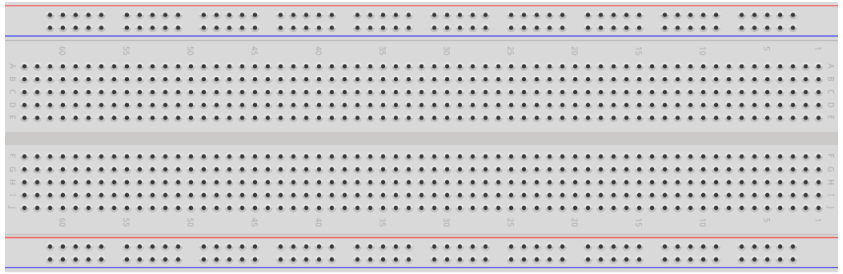
- Power rails on the breadboard (+ and –)
- Correct orientation of MB-102 power module
- Red = VCC, Black = GND to rails
- Never power Pico W from USB and batteries simultaneously



Level 1 – Components Explanation (1/3)

Breadboard:

- Plastic board with holes and internal metal clips
- Has numbered rows and power buses (+ and –)

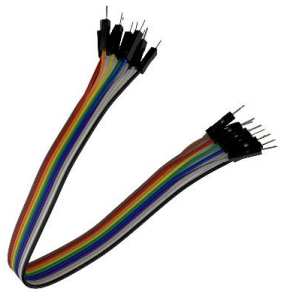


Resistors:

- Used to limit current
- Have colour codes indicating resistance value

Jumper cables:

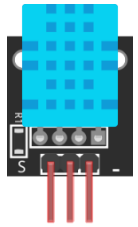
- Connect two points without soldering
- Colour variation used to differentiate between types of connections (e.g., black/brown for GND)



Level 1 – Components Explanation (2/3)

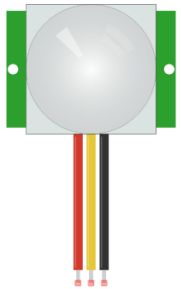
DHT11 – Temperature & Humidity Sensor:

- Measures temperature and humidity
- Has VCC, GND, DATA pins
- Connected to a GPIO pin for data



PIR Motion Sensor:

- Detects motion (infrared changes)
- Has VCC, GND, OUT pins
- Output is HIGH when motion is detected



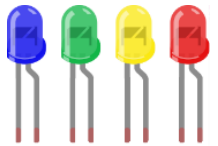
LDR Photoresistor:

- Light-dependent resistor
- Forms a voltage divider with a resistor
- Read with an analog pin (e.g., ADC0)

Level 1 – Components Explanation (3/3)

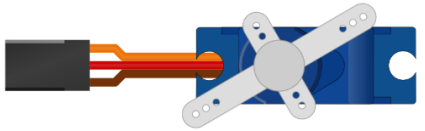
LEDs:

- Indicate states (e.g., lights, alarms)
- Must have a current-limiting resistor (e.g., 220Ω)
- Connected to a GPIO pin via resistor



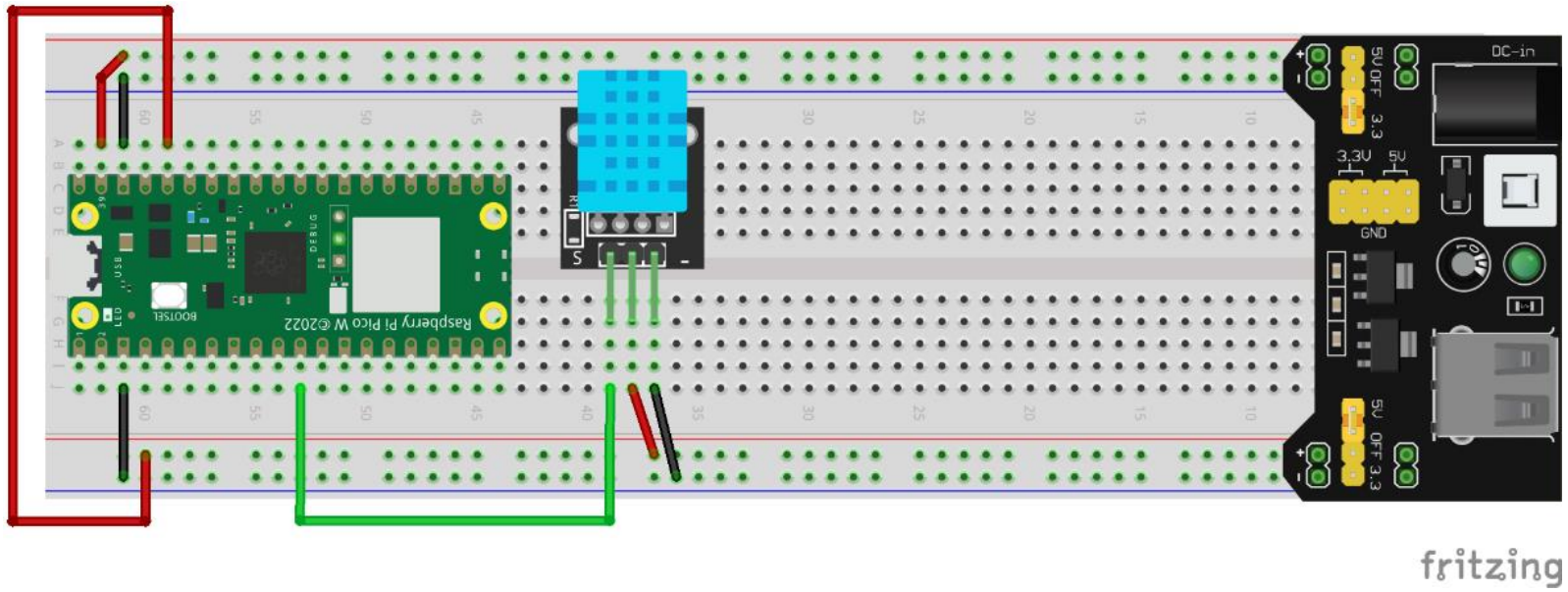
Servo Motor (SG-90):

- Controls position (e.g., “window” opening)
- Has VCC, GND, signal wires
- Controlled by PWM signal from a GPIO pin



Level 1 – Connecting the DHT11 Sensor

- VCC (red cable) is connected to 3v3 rail (+)
- GND (black cable) is connected to GND rail (-)
- S (green cable) is connected to GPIO8 pin
- Note that in this exercise we use the KY-015 module DHT11 sensor which has three legs.



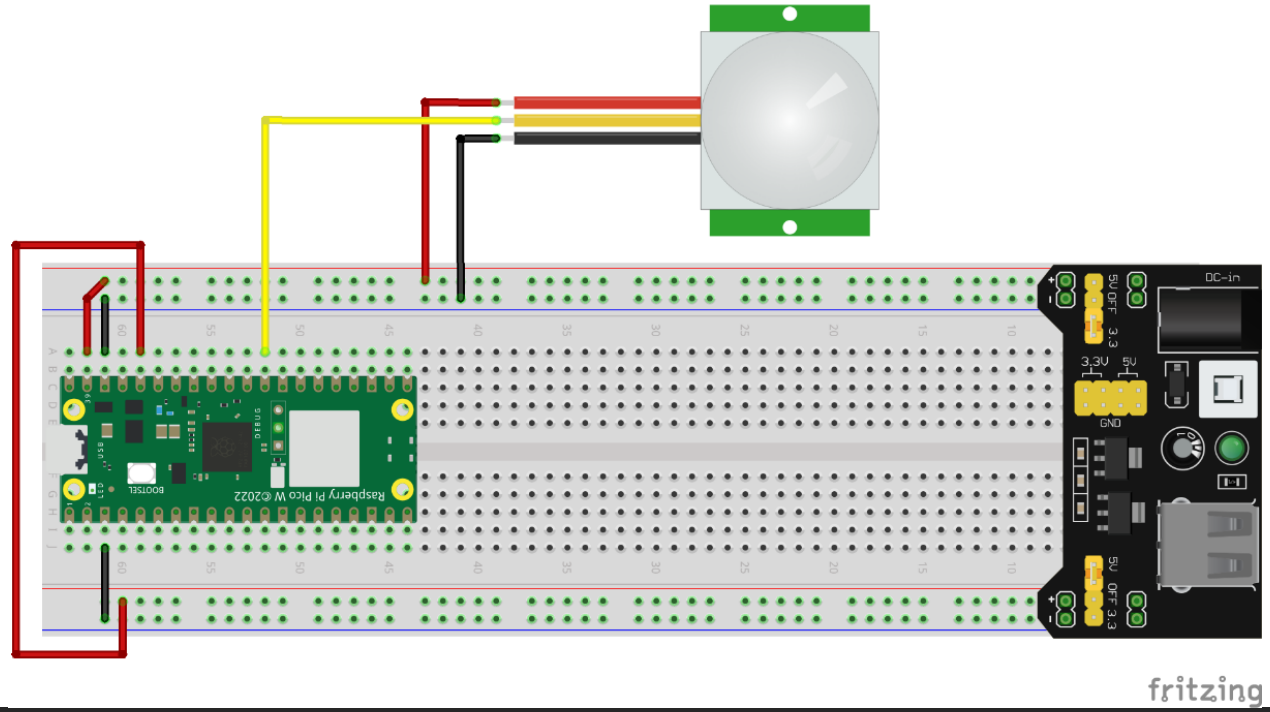
Level 1 – Programming the DHT11 Sensor

- The program imports `Pin`, `DHT11`, and `sleep`, sets `PIN_DHT=8` (so the DHT11 data pin is on GP8), and creates the sensor object.
- Inside `while True:`, it calls `dht.measure()` to take a reading, then gets `temperature()` and `humidity()` and prints lines like `Room Temperature: 24 °C` and `Room Humidity: 48 %`.
- By using `sleep(2)` we respect the DHT11's ~1 Hz max update rate.

```
from machine import Pin
from dht import DHT11
from time import sleep

#Define pins for each component
PIN_DHT = 8

#Setup DHT11
dht = DHT11(PIN_DHT)

#Main loop
while True:
    dht.measure()
    temp = dht.temperature()
    hum = dht.humidity()
    print("Room Temperature:", temp, "°C")
    print("Room Humidity:", hum, "%")
    sleep(2)
```

Level 1 – Connecting the PIR Sensor

- VCC (red cable) is connected to 5V rail (+)
- GND (black cable) is connected to GND rail (-)
- OUT (yellow cable) is connected to GPIO22 pin



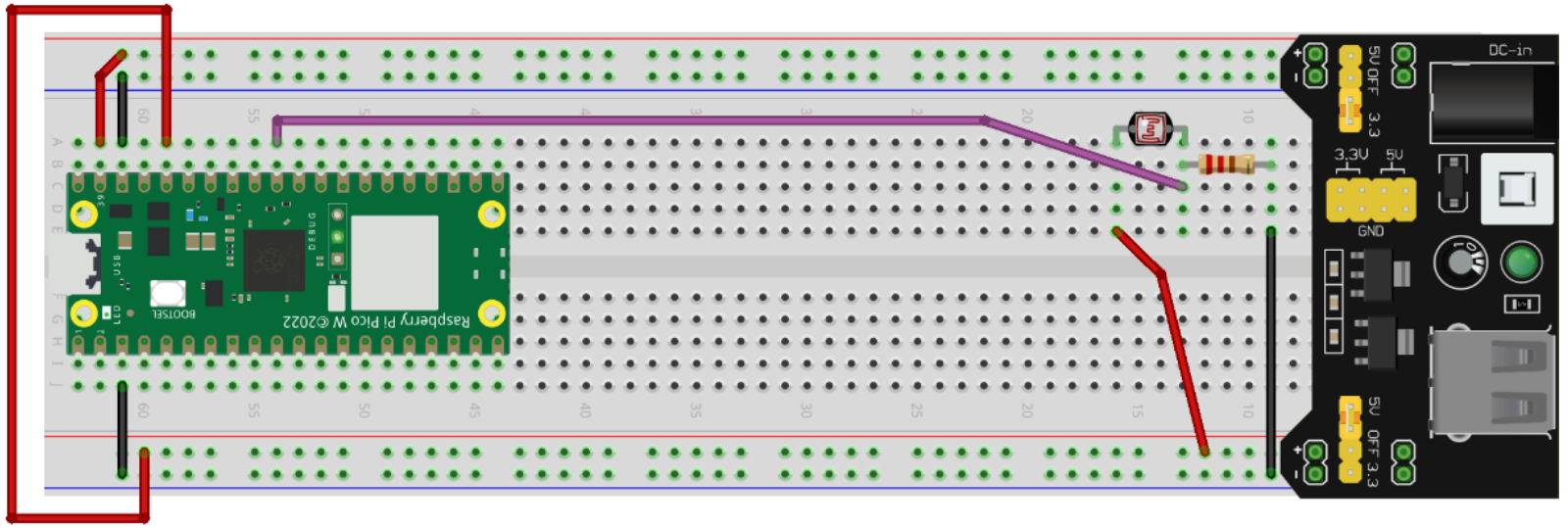
Level 1 – Programming the PIR Sensor

- The program imports `Pin` and `sleep`, and sets GP22 (`PIN_PIR = 22`) as a digital input.
- In an infinite loop, it checks the pin: **if the input is HIGH (1)** – which most PIR modules output when motion is detected – it prints **“Motion Detected”**.
- `sleep(1)` slows the loop to avoid spamming the console.

```
from machine import Pin
from time import sleep

#Define pins for each component
PIN_PIR = 22

#Setup PIR
PIR = Pin(PIN_PIR, Pin.IN)

#Main loop
while True:
    if PIR.value() == 1:
    print("Motion Detected")
    sleep(1)
```

Level 1 – Connecting the LDR Photoresistor

- left side of LDR is connected to 3V rail ((+) red)
- right side of LDR is connected to a 220 Ohm resistor and to GPIO26 ADC0 (purple cable) pin
- right side of resistor is connected to GND rail ((-) black)



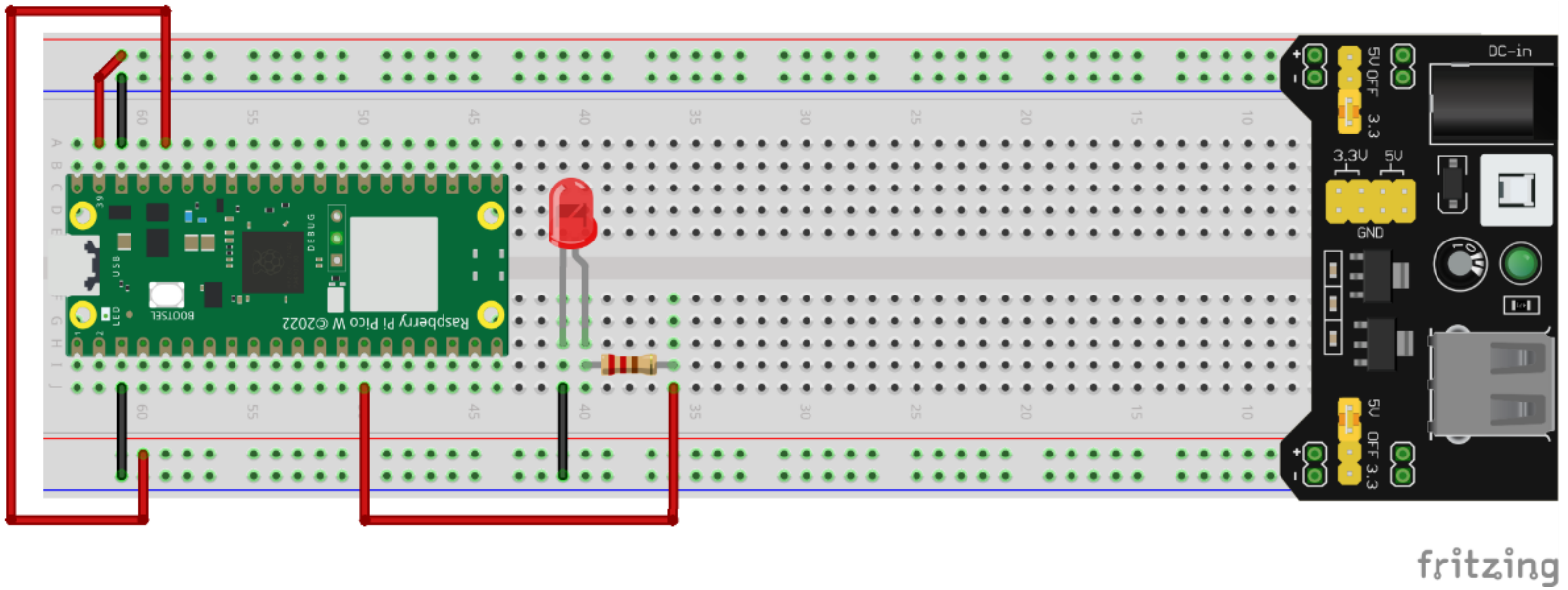
Level 1 – Programming the LDR Photoresistor

- `PIN_LDR = 26` selects GP26/ADC0.
- `LDR = ADC(Pin(PIN_LDR))` sets that pin as an ADC input.
- In the loop, `LDR.read_u16()` returns a 16-bit value (0–65535) proportional to the voltage from your LDR voltage divider (bright light → lower value).
- `sleep(1)` spaces readings by one second.

```
from machine import Pin, ADC
from time import sleep

#Define pins for each component
PIN_LDR = 26

#Setup LDR
LDR = ADC(Pin(PIN_LDR))

#Main loop
while True:
    print(LDR.read_u16())
    sleep(1)
```

Level 1 – Connecting the LEDs

- connect the longer end (+) of the LED to a 220Ohm resistor
- connect the shorter end (-) of the LED to GND rail ((-) black)
- for red LED connect the resistor to GPIO10 (red cable)



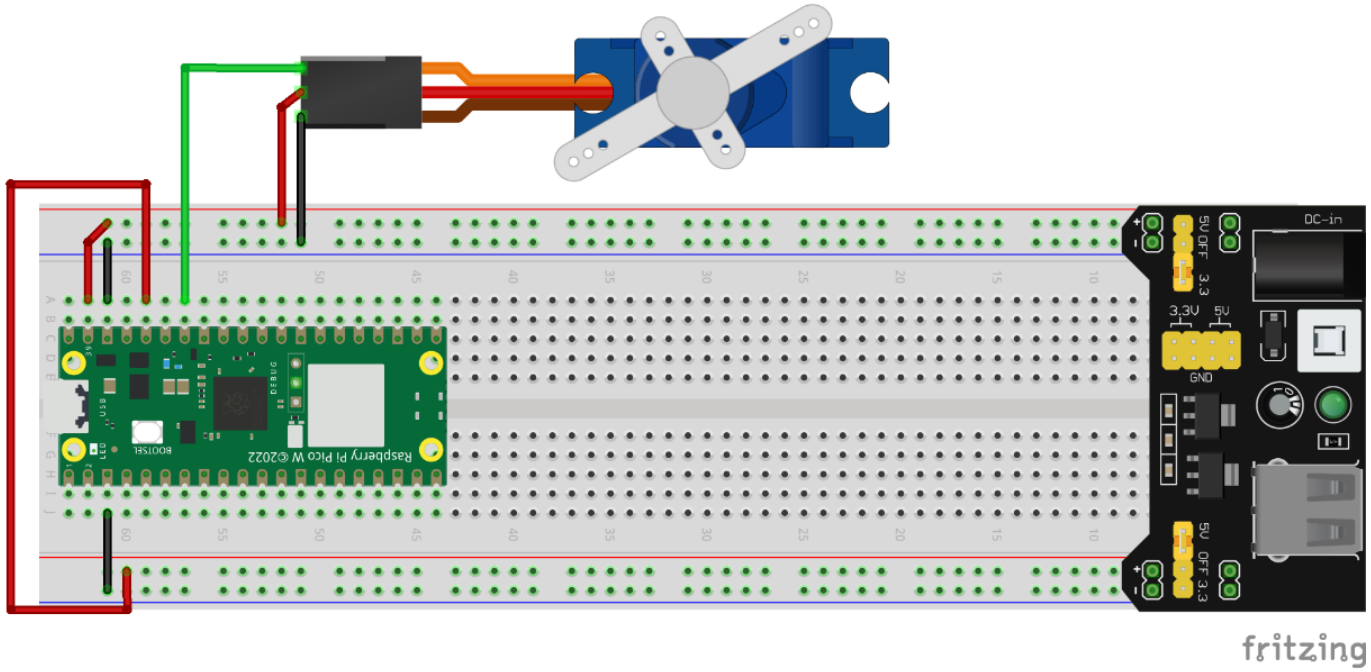
Level 1 – Programming the LEDs

- The program sets GP10 as a digital output (`Pin(LED_RED, Pin.OUT)`)
- Then, turns it off once (`ledred.value(0)`),
- Then enters an infinite loop where it toggles the pin state every 1 second (`ledred.toggle(); sleep(1)`), making the LED alternately turn on and off each second.
- Follow the same coding principle to add additional leds. Use GPIO 11 for blue LED

```
from machine import Pin
from time import sleep

#Define pins for each component
PIN_RED = 10

#Setup LEDs
ledred = Pin(PIN_RED, Pin.OUT)

#Initial value set to "off"
ledred.value(0)

#Main loop
while True:
    ledred.toggle()
    sleep(1)
```

Level 1 – Connecting the Servo Motor

- red cable is connected to 5V rail (+)
- black/brown cable is connected to GND rail (-)
- green cable is connected to GPIO28 ADC2 pin



Level 1 – Programming the Servo Motor

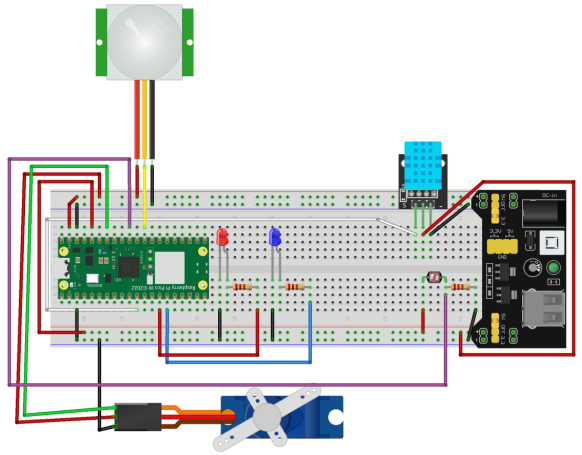
- First you need to install the "**micropython-servo"** library by going to **Tools → Manage packages**, then **search** for the library and click **"Install".**
- Setup: `PIN_SERVO = 28`. You make a Pin on GP28 (`servo = Pin(PIN_SERVO)`) and create a Servo controller on that pin (`angle = Servo(pin_id=servo)`). You center it at 0° with `angle.write(0)` and wait 0.5 s.
- Main loop: Inside `while True` it commands 90° → wait 2 s, 180° → wait 2 s, then 0° → wait 2 s.
- Exit: `break` immediately leaves the loop after reaching 0°, so the sequence runs once and stops.

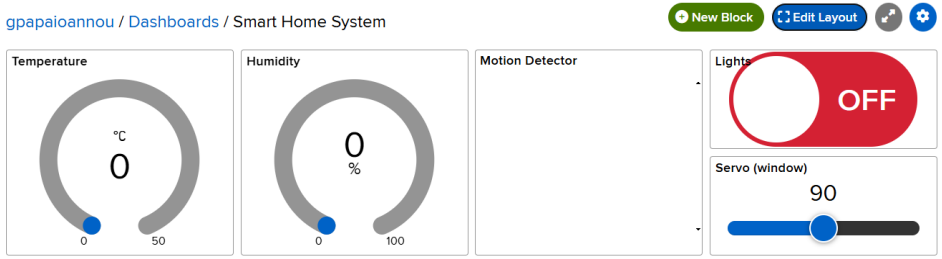
```
from machine import Pin
from servo import Servo
from time import sleep

#Define pins for each component
PIN_SERVO = 28

#Setup servo motor
servo = Pin(PIN_SERVO)
angle = Servo(pin_id=servo)
angle.write(0) #set initial angle at 0 degrees
sleep(0.5)

#Main loop
while True:
    angle.write(90)
    sleep(2)
    angle.write(180)
    sleep(2)
    angle.write(0)
    sleep(2)
    break
```

Level 2 – Integrate System & Adafruit IO

Sessions 6–8 focus on:

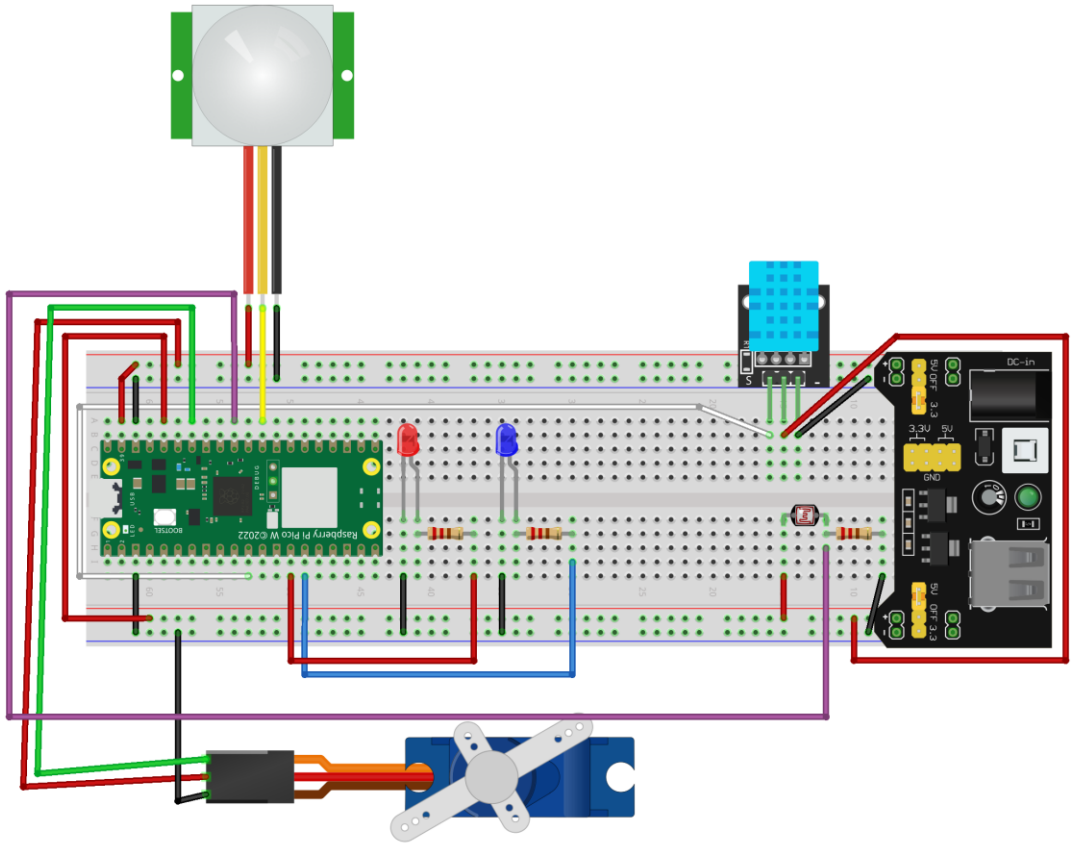
- Building the **complete Smart Home circuit** with all sensors & actuators
- Connecting Pico W to **Wi-Fi** and **Adafruit IO**
- Sending sensor data (temperature, humidity, motion) to Adafruit IO feeds
- Subscribing to command feeds for LEDs and servo motor
- Building and online dashboard
- Testing, debugging, documenting, and **presenting student projects**





Level 2 – Complete Smart Home Circuit

Follow the schematic to connect all sensors and actuators:



Level 2 – Programming the Smart Home (1/3)

The complete code creates a SmarHome system with the following features:

- A smart temperature system (DHT11, Servo motor)
- A smart lights system (LDR photoresistor, LEDs)
- A smart security system (PIR, LEDS, Servo motor

The complete code is the following:

***code snippet (1/5)**

```
from machine import Pin, ADC
from dht import DHT11
from servo import Servo
from time import sleep

#Define pins for each component
PIN_SERVO = 28
PIN_DHT = 8
LED_RED = 10
LED_BLUE = 11
PIN_PIR = 22
PIN_LDR = 26
PIN_SERVO = 28

#Setup DHT11
dht = DHT11(PIN_DHT)
```

Level 2 – Programming the Smart Home (2/3)

***code snippet (2/5)**

```
#Setup PIR
PIR = Pin(PIN_PIR, Pin.IN)

#Setup LED lights
ledred = Pin(LED_RED, Pin.OUT)
ledblue = Pin(LED_BLUE, Pin.OUT)
ledred.value(0)
ledblue.value(0)

#Setup LDR
LDR = ADC(Pin(PIN_LDR))

#Setup servo motor
servo = Pin(PIN_SERVO)
angle = Servo(pin_id=servo)
angle.write(0) #we set this to 0 degrees
sleep(0.5)
```

***code snippet (3/5)**

```
#Global variables
SERVO_CLOSED = 0
SERVO_OPEN = 90
TEMP_threshold = 28 #adjust as needed
HUM_threshold = 75 #adjust as needed

#Main loop
while True:
    dht.measure()
    temp = dht.temperature()
    hum = dht.humidity()
    print("Room Temperature:", temp, "°C")
    print("Room Humidity:", hum, "%")
    sleep(5)
```

Level 2 – Programming the Smart Home (3/3)

*code snippet (4/5)

```
#Smart temperature system
if temp > TEMP_threshold:
    angle.write(SERVO_OPEN)
    sleep(2)
elif hum > HUM_threshold:
    angle.write(SERVO_CLOSED)
    sleep(2)
else:
     pass #keep current position

#Smart lights system
light_level = LDR.read_u16()
if light_level > 5000: #adjust as needed
    ledred.value(1)
    ledblue.value(1)
    sleep(5)
```

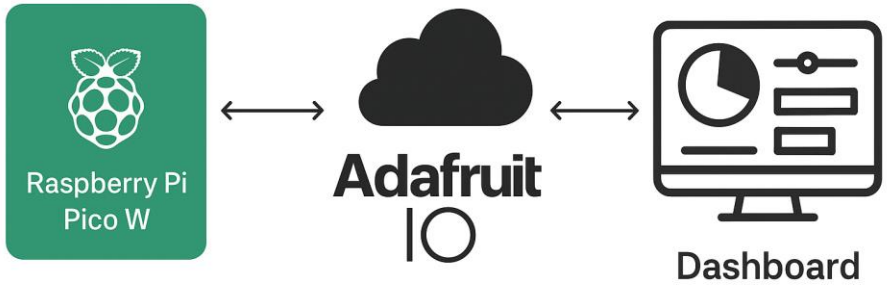
*code snippet (5/5)

```
#Smart security system
if PIR.value() == 1:
    print("Motion Detected")
    ledred.value(1)
    angle.write(SERVO_CLOSED)
    sleep(2)
    break
```

Level 2 – Adafruit IO Setup

A complete Smart Home Application using the Adafruit IO is developed, that has the following features:

- On/Off LEDs switch
- Open/Close servo motor
- Temperature/Humidity monitoring
- Security monitoring

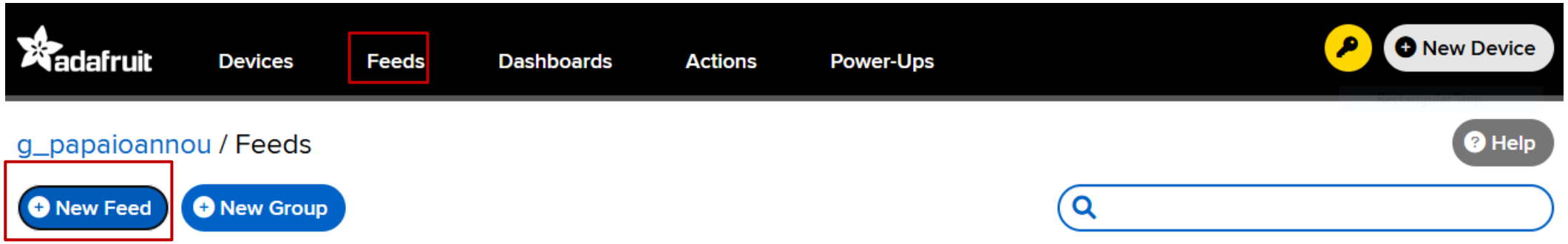


First, create a free account at https://io.adafruit.com

Level 2 – Creating Feeds (1/2)

"Feeds" are used to store data. In this project, we need feeds to send sensors' data to the application and be able to send commands from the application to the Smar Home system.

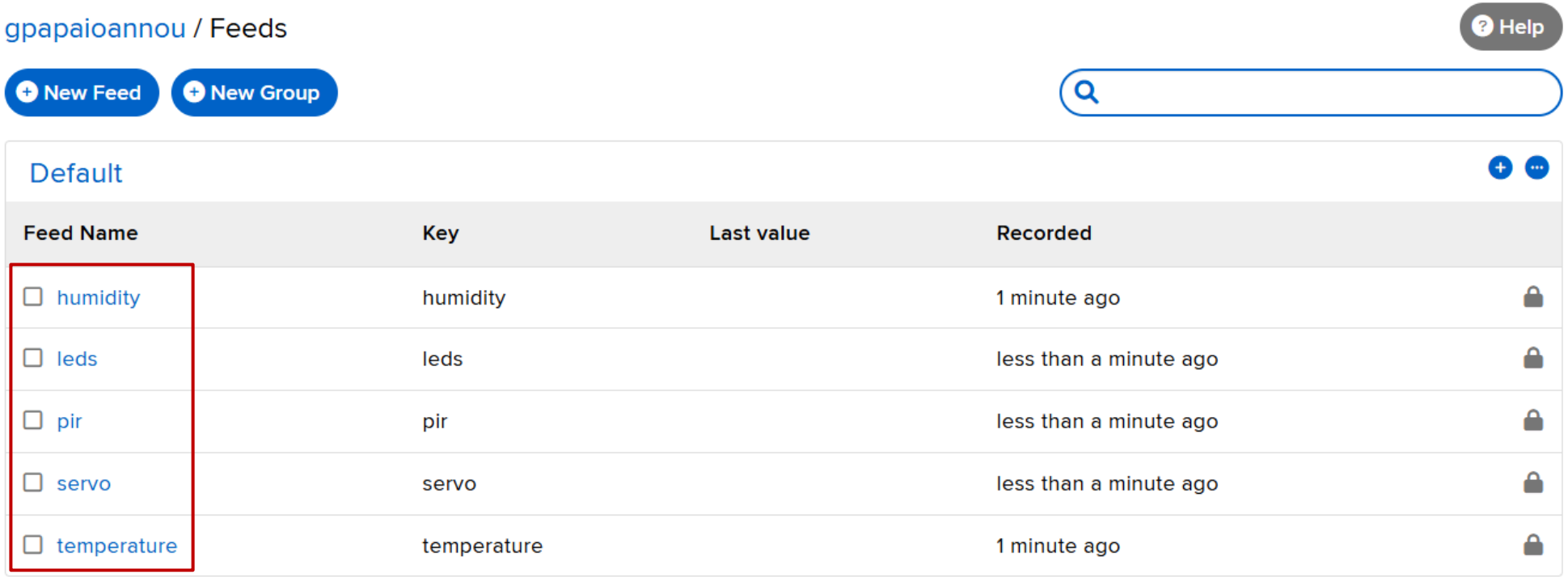
Create a feed by going to the "Feeds" tab and select "New Feed".



Then, name your Feed and click on "Create"

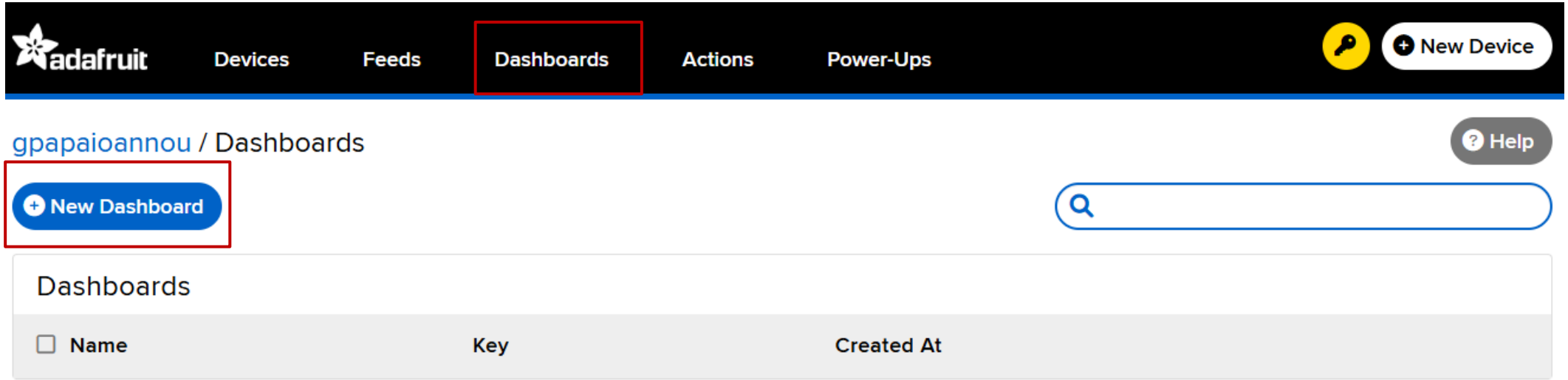
Level 2 – Creating Feeds (2/2)

You need to create five feeds:



Level 2 – Creating the Dashboard (1/3)

Next, you need to create a dashboard by going to the “Dashboards” tab and then clicking on “New dashboard”. Name your Dashboard and then click on “Create”

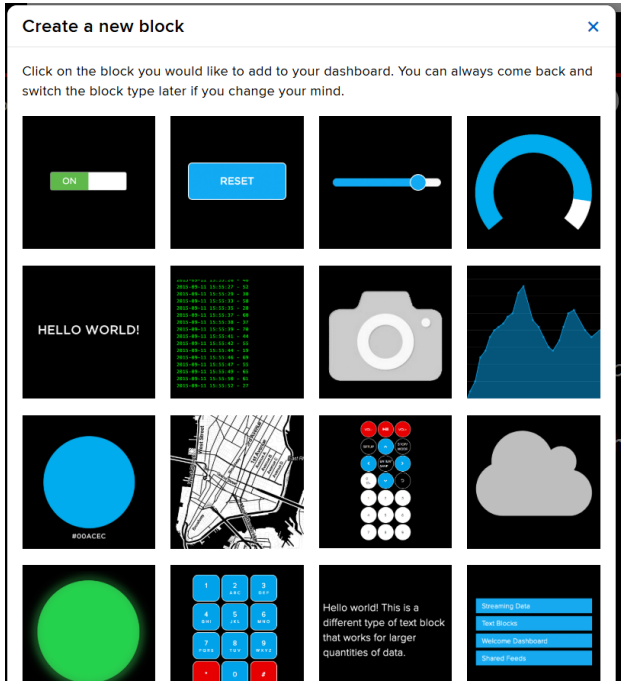


Level 2 – Creating the Dashboard (2/3)

Click on the name of your Dashboard to enter it. You will need to add blocks for displaying and interacting with the data. Add one toggle, one slider, two gauges, and one stream block.

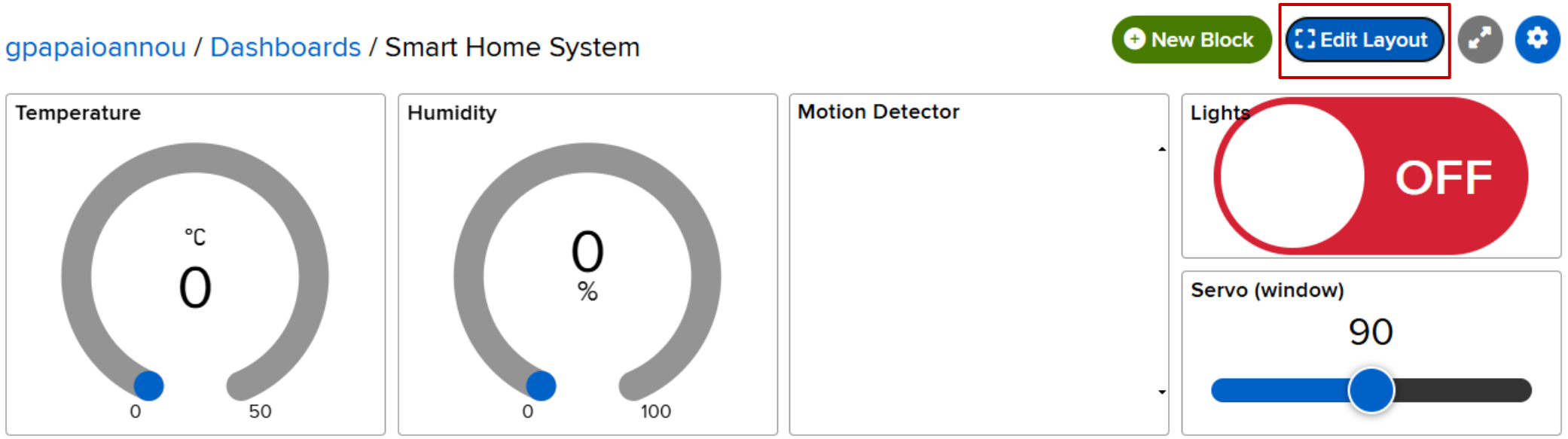
Make sure to associate each with the relevant feed:

- Gauge #1 with “temperature” feed
- Gauge #2 with “humidity” feed
- Toggle with “leds” feed
- Slider with “servo” feed
- Stream with “pir” feed



Level 2 – Creating the Dashboard (3/3)

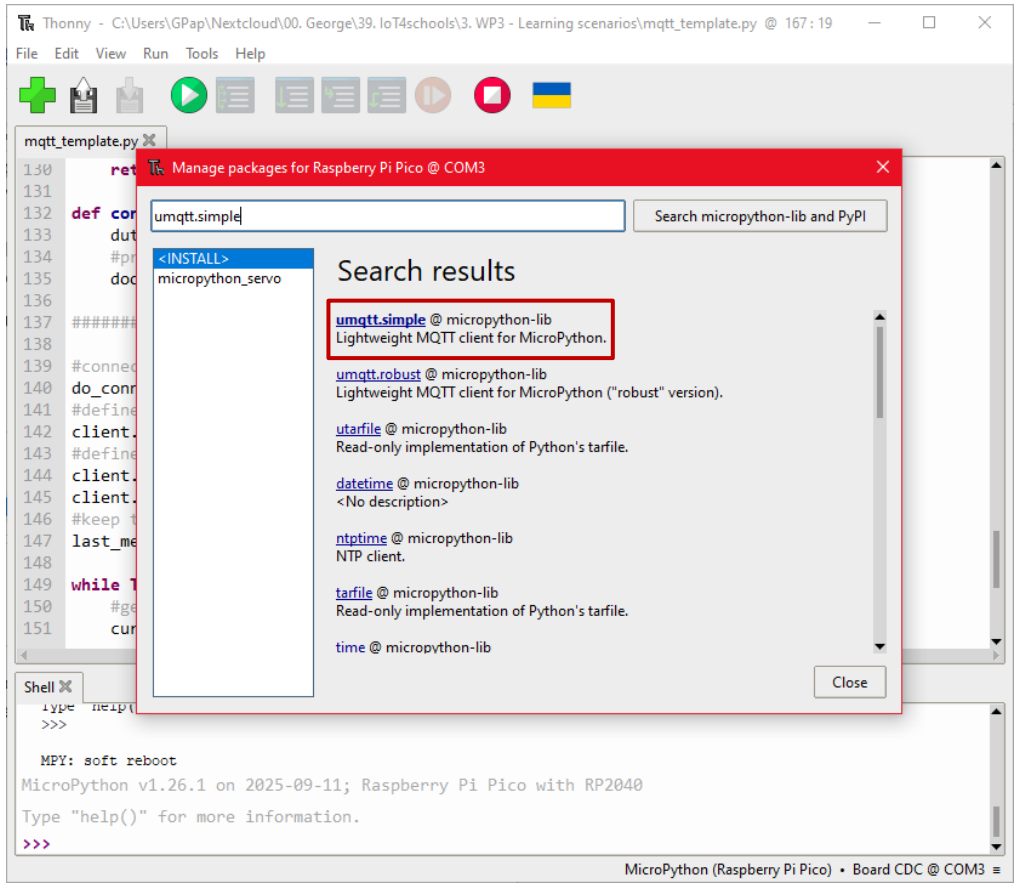
Your dashboard should look like the following image. But if it doesn't, you can edit the layout by clicking on the "Edit Layout" button.



Level 2 – Installing communication libraries

Two library need to be installed on your Raspberry Pi Pico W:

- umqtt.simple by going to Tools → Manage packages → search umqtt.simple → Install
- iot4schools.py by visiting: https://github.com/angsam/iot4schools → download library → open with Thonny → Save as at Raspberry Pi Pico → Click “Ok”



Level 2 – Connecting RPi Pico W to Wi-Fi & Adafruit IO

Going back to your code, you need to add the following pieces:

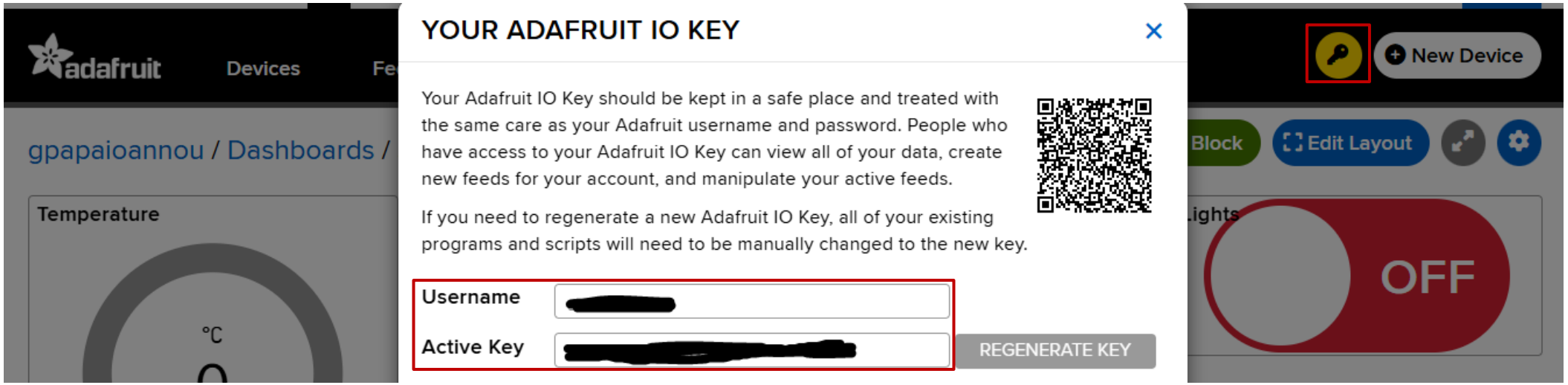
- First import the library iot4schools.py
- The next lines are used to setup the Wi-Fi connection
- The `ADAFRUIT_IO_USERNAME` and `ADAFRUIT_IO_KEY` are found at your Adafruit IO account
- MQTT setup is used to send and receive data

***final code snippet (1/7)**

```
from machine import Pin, ADC
from dht import DHT11
from servo import Servo
from time import sleep

import iot4schools

#Setup WIFI connection
WIFI_SSID = "your_wifi_name"
WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_IO_USERNAME = "your username"
ADAFRUIT_IO_KEY = "your_key_from_adafruit"

iot4schools.connect_wifi(WIFI_SSID, WIFI_PASSWORD)

#Setup MQTT
client = iot4schools.create_mqtt_client(ADAFRUIT_IO_USERNAME,
ADAFRUIT_IO_KEY)
iot4schools.connect_mqtt(client)
```

Level 2 – Adafruit Username & Key

Simply go to the Adafruit IO account, then click on the “key” icon and you will see your Username and IO Key.



Level 2 – Adding parts from the initial program

***code snippet (2/7)**

```
#Define pins for each component
PIN_SERVO = 28
PIN_DHT = 8
LED_RED = 10
LED_BLUE = 11
PIN_PIR = 22
PIN_LDR = 26
PIN_SERVO = 28

#Setup DHT11
dht = DHT11(PIN_DHT)

#Setup PIR
PIR = Pin(PIN_PIR, Pin.IN)

#Setup LDR
LDR = ADC(Pin(PIN_LDR))
```

***code snippet (3/7)**

```
#Setup LED lights
ledred = Pin(LED_RED, Pin.OUT)
ledblue = Pin(LED_BLUE, Pin.OUT)
ledred.value(0)
ledblue.value(0)

#Setup servo motor
servo = Pin(PIN_SERVO)
angle = Servo(pin_id=servo)
angle.write(0) #we set this to 0 degrees
sleep(0.5)

#Global variables
SERVO_CLOSED = 0
SERVO_OPEN = 90
TEMP_threshold = 28 #adjust as needed
HUM_threshold = 75 #adjust as needed
```

Level 2 – Paths to “feeds” and Send_Data function

Now you are ready to store paths to feeds and create a function that send data to feeds.

***final code snippet (4/7)**

```
#Setup FEEDS
FEED_TEMP = "your_username/feeds/temperature"
FEED_HUM = "your_username/feeds/humidity"
FEED_LEDS = "your_username/feeds/leds"
FEED_PIR = "your_username/feeds/pir"
FEED_SERVO = "your_username/feeds/servo"

#Function to send data
def Send_Data(temperature, humidity, pir):
  client.publish(FEED_TEMP, bytes(str(temperature), "utf-8"), qos=0)
  client.publish(FEED_HUM, bytes(str(humidity), "utf-8"), qos=0)
  client.publish(FEED_PIR, bytes(str(pir), "utf-8"), qos=0)
```

Level 2 – Receive_Data function

You need to create a new function that will receive data from the Adafruit IO (e.g. to turn on/off the LEDs or to turn the Servo to a specific angle)

***final code snippet (5/7)**

```
#Function to receive data
def Receive_Data(feed, msg):
    if FEED_LEDS in feed:
        #control LEDs
        action = str(msg, "utf-8")
        print("action = {} ".format(action))
        if action == "ON":
            ledred.value(1)
            ledblue.value(1)
        else:
            ledred.value(0)
            ledblue.value(0)
        if FEED_SERVO in feed:
            #control servo
            action = str(msg, "utf-8")
            print("action = {} ".format(action))
            angle.write((float(action))
```

Level 2 – Subscribing data and main program loop

- First you need to call the `set.callback()` which informs the Raspberry Pi Pico that you send a “message” using the Adafruit IO application.
- Then, in order for your program to know which Feeds sent a “message” you need to use the `subscribe` function

***final code snippet (6/7)**

```
#Define Receive_Data function name to client
client.set_callback(Receive_Data)

#Define Feed names to get messages from
client.subscribe(FEED_LEDS)
client.subscribe(FEED_SERVO)
```

***final code snippet (7/7)**

```
#Main loop
while True:
    dht.measure()
    temp = dht.temperature()
    hum = dht.humidity()
    if PIR.value() == 1:
        print("Motion Detected")
        break

    Send_Data()

    sleep(30)

    #try to read message from client (eg from FEED_SERVO)
    try:
        client.check_msg()
    except KeyboardInterrupt:
        print("Ctrl-C pressed...exiting")
        client.disconnect()
        sys.exit()
```

Level 2 – Understanding the main program loop

In the main program loop:

- The script asks the DHT sensor to take measures of temperature and humidity.
- It asks the PIR sensor to inform if motion is detected.
- Then sends this data to the Adafruit IO Application.
- The script then waits for 30 seconds before checking again.
- Then the script checks if a message is received from the Adafruit IO application.
- Finally, if the user wants to stop the script from running by pressing Ctrl+C, the script disconnects the Pico from the Adafruit IO application and turns off the system.

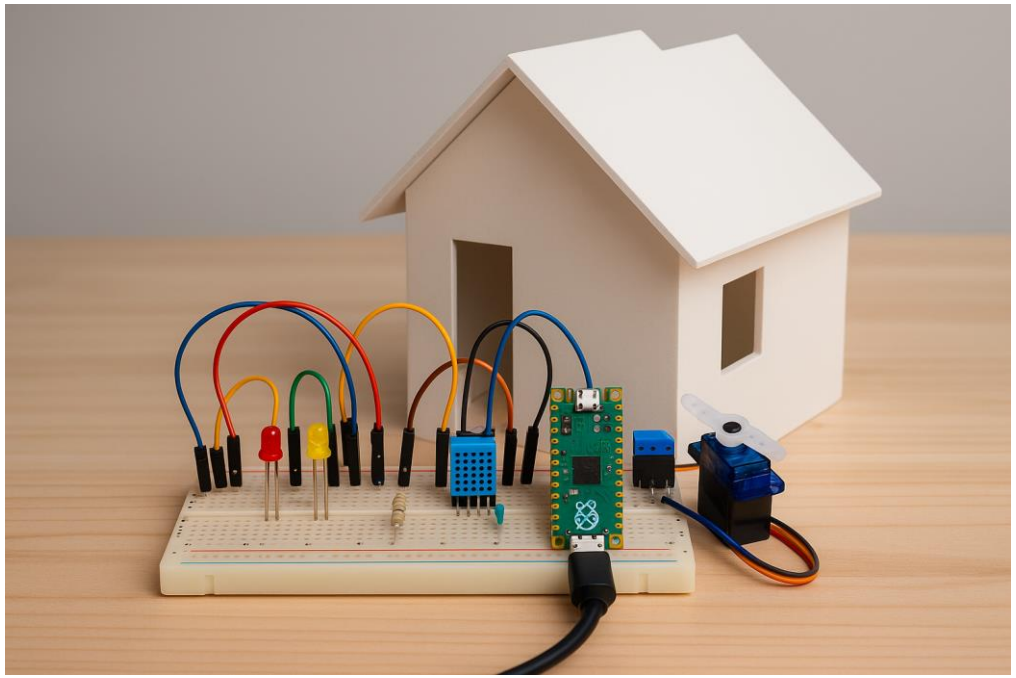
Outcomes & Extensions

By completing the project, students will:

- Build and demonstrate a working Smart Home prototype
- Document and present their design decisions
- Reflect on sustainability, privacy and security in IoT

Possible extensions:

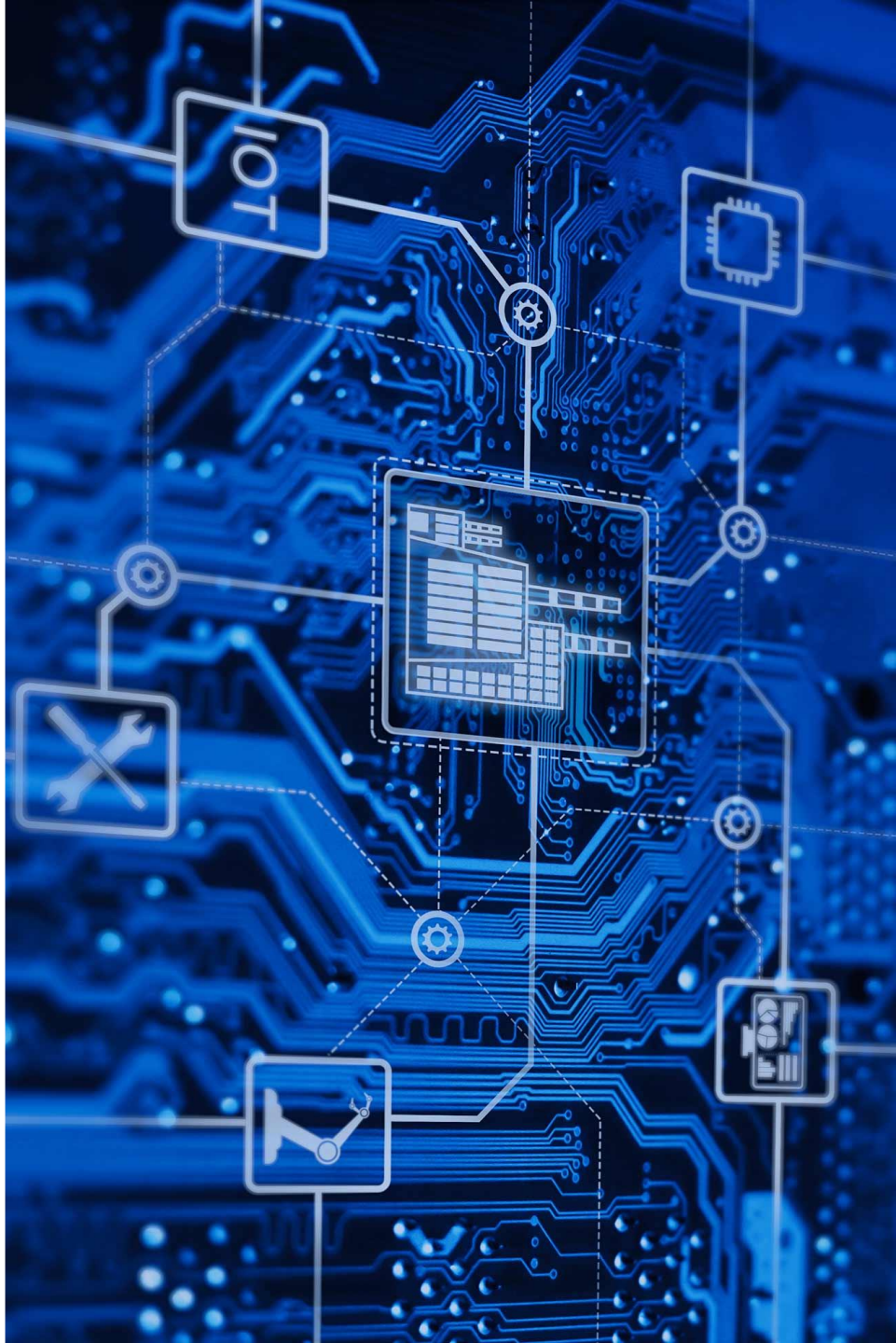
- Add more sensors/actuators (buzzer, OLED display, etc.)
- Extend Adafruit IO dashboard with charts and data logging
- Connect with other IoT4Schools activities or school projects



PROJECT NUMBER: 2023-1-PL01-KA220-SCH-000154043

Bringing the Internet of Things in school education as a tool to address 21st century challenges

Thank you & Good luck!



Co-funded by the European Union