

Project number: 2023-1-PL01-KA220-SCH-000154043



Co-funded by  
the European Union

## **IoT4Schools**

**“Bringing the Internet of Things in school education as a  
tool to address 21<sup>st</sup> century challenges”**

# **Smart waste bins: how to improve waste management in smart cities?**

## **Worksheet**

**Authors:** Angelika Tefelska, Dariusz Tefelski

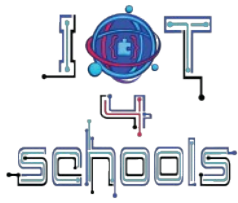
**Organization:** Warsaw University of Technology, Faculty of Physics

*License: CC BY-NC 4.0 LEGAL CODE, Attribution-NonCommercial 4.0 International*



Co-funded by  
the European Union

*The European Commission's support to produce this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.*



Team: .....

### 1. Time for a brainstorming session

What do you know about waste management in the city? (collection methods, processing, recycling statistics, fuel consumption statistics for garbage trucks) Search for information on this topic.

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

How can the current waste management system be improved? How can the collection of waste from houses/apartments be improved?

.....  
.....  
.....  
.....  
.....  
.....

If you want to build a smart trash can, how should it work?

.....  
.....  
.....  
.....  
.....



### 3.1 First warm-up tasks: traffic lights

Each program written in MicroPython for programming the Raspberry Pi Pico looks like this:

```

1  import machine
2
3  #instructions that are to be executed only once here,
4  #e.g. configuration, creating variables
5
6  while True:
7      #instructions to be repeated over and over|

```

At the beginning we add the machine library, which contains basic functions for operating the microcontroller. Then we need to add instructions that are to be executed only once at the very beginning, such as configuration, creating variables or creating our own functions. Then we always have an infinite loop, in which we place instructions that will be repeated over and over.

The basic functions necessary to complete the exercise are:

- **machine.Pin(pin number, machine.Pin.IN or machine.Pin.OUT)** - configures whether a given pin should be an input (machine.Pin.IN) or output (machine.Pin.OUT). Input pins are used when we want to read data from an element connected to a pin, e.g. when we have a button connected, we read whether it was pressed (value 1) or not (value 0). Output pins are used when we want to control a given element connected to a pin, e.g. an LED diode, whether it should be on (value 1) or off (value 0).
- **value(0 or 1)** - function for setting the value on a given pin.
- **utime.sleep(time in seconds)** - function that suspends program execution for a given time. To use it, you must include the utime library: import utime.

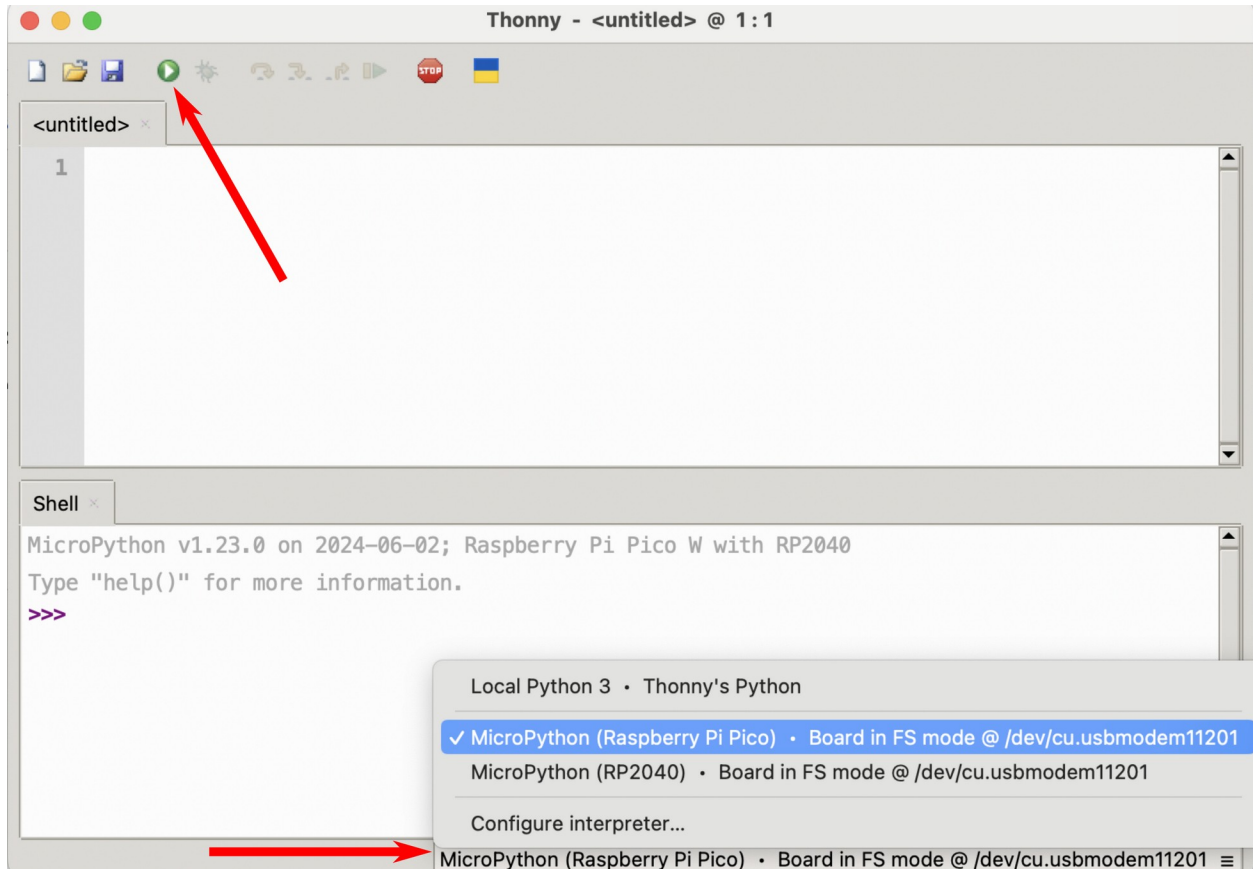
So if we wanted the LED connected to pin GP15 to blink every 1 second, the code would look like this:

```

1  import machine
2  import utime
3
4  led = machine.Pin(15, machine.Pin.OUT)
5
6  while True:
7      led.value(1)
8      utime.sleep(1)
9      led.value(0)
10     utime.sleep(1)|

```

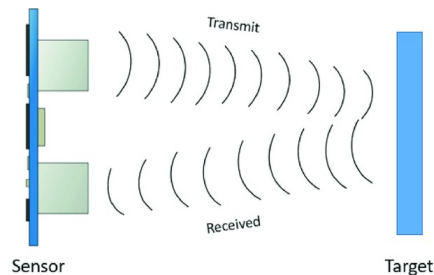
Open the Thonny editor and then select "MicroPython (Raspberry Pi Pico)" as shown in the picture. After connecting to the board correctly, the green Run button should be active (not grayed out). If it is grayed out, select "MicroPython (Raspberry Pi Pico)" again.



Now try to create a program that will simulate traffic lights.

### 3.2 Warm-up task no. 2: ultrasonic distance sensor

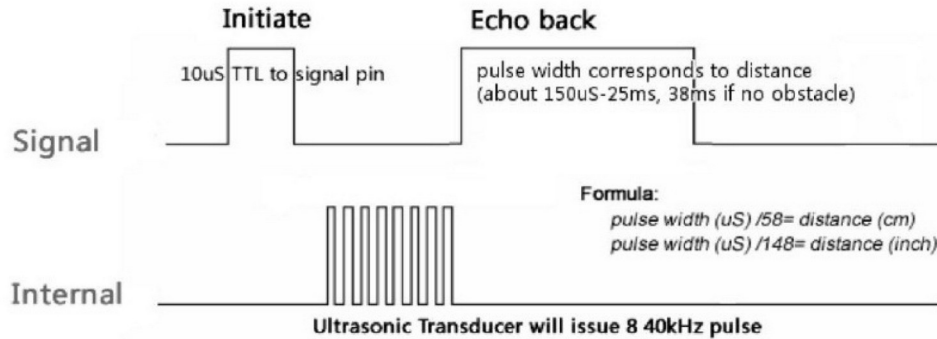
The HC-SR04 ultrasonic distance sensor allows you to measure the distance from an obstacle. The measurement idea is shown in the figure below:



Source: [https://www.researchgate.net/figure/A-block-diagram-of-Ultrasonic-sensor-working-principles\\_fig5\\_344385811](https://www.researchgate.net/figure/A-block-diagram-of-Ultrasonic-sensor-working-principles_fig5_344385811)



To measure, according to the sensor documentation (see the figure below), set the signal low on the trigger for a short time, e.g. 2µs. Then set the high signal for 10µs. To generate delays in microseconds, use the function: **utime.sleep\_us()**. In the next step, set the low signal on the trigger.



Source: <https://www.electronicoscaldas.com/datasheet/HC-SR04.pdf>

Create a program that reads the measured distance from an ultrasonic distance sensor and displays it in the terminal. To do this:

1. Add the machine and utime libraries.
2. Configure the pin to which the trigger is connected as an output.
3. Configure the pin to which the echo is connected as an input.
4. In the while loop, set the value of the trigger pin to 0.
5. Wait 2 µs (**utime.sleep\_us()** function).
6. Set the value of the trigger pin to 1.
7. Wait 10 µs.
8. Set the value of the trigger pin to 0.
9. Now we need to measure how long the high signal on the echo pin lasts, because the duration of the signal on the echo pin is related to distance. To do this, we will use the **utime.ticks\_us()** function, which measures how much time has passed in µs since the program was started. First, we will create a while loop that will execute as long as the signal is low. Inside, we will place the **tick\_us()** function. This way, we will get information about when the signal was last low. Add this code snippet to your program:

```
while echo.value()==0:
    signal_off = utime.ticks_us()
```

10. Similarly, measure when the signal was last high and save the value to a variable named e.g. **signal\_on**.

11. The difference between the time of the last occurrence of the high and low signal is the duration of the high pulse on the echo pin. How do we translate the pulse duration into distance? At the beginning, a



sound wave is emitted, which reflects from the object and returns to the sensor. Therefore, in the measured time ( $t$ ), the wave travels twice the distance between the sensor and the object and moves at a speed of about 340 m/s (the speed of sound in air). Therefore, we can write the equation for the speed:

$$v = \frac{2d}{t}$$

$$d = v \cdot \frac{t}{2} = 0.034 \frac{\text{cm}}{\mu\text{s}} \cdot \frac{t}{2} \approx \frac{t[\mu\text{s}]}{58}$$

Hence, the obtained pulse duration should be divided by 58 to get the distance in centimeters. Add the following lines to the program:

```
diff = signal_on - signal_off
distance = diff / 58.0
print("Distance=" + str(distance))
```

Now you can test the operation of the program. If you have difficulty reading data because it is displayed too quickly, you can add a small delay to the print function.

## 4. Smart waste bin – level 1

Now create a program in which you will use the skills you have acquired during the warm-up tasks and measure the fill level of the waste bin. Display the fill level on the LEDs according to the percentage ranges of the bin occupancy that you assumed at the stage of designing the waste bin.

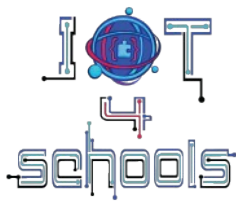
Create a waste bin from an unused shipping/shoe box and test how your bin works.

## 5. Smart waste bin – level 2

Now let's modify the trash can to be even smarter. To do this, we will send data to the cloud about the fullness of the trash can and its location. This data can later be used to create an algorithm that develops a route for the garbage truck to collect trash only from those locations where the bins are full, thus minimizing the carbon footprint. For this purpose, we will use the Adafruit IO cloud.

First, you need to create a free account at <https://io.adafruit.com>. Next, you will want to send two pieces of data: garbage container filling and location to the cloud. To do this, you need to create two feeds. Feeds are objects that store data. To create a feed, go to the "Feed" tab and select the "New Feed" button.





Then a window will appear where you need to enter the feed name, e.g. Filling or Location.

Create a new Feed ✕

Name

Maximum length: 128 characters. Used: 0

Description

Cancel Create

Create two feeds. Once you do that, you should see the feeds you created on your page, similar to the screenshot below:

Shop Learn Blog Forums **IO** LIVE! AdaBox Hi, Angelika Tefelska | Account 0

adafruit Devices Feeds Dashboards Actions Power-Ups 🔑 + New Device

angtef / Feeds 🔍 ? Help

+ New Feed + New Group

Feed Name	Key	Last value	Recorded
<input type="checkbox"/> Filling	filling-the-waste-bin	71.62069	about 5 hours ago
<input type="checkbox"/> Location	location	52.2297,21.0122	about 6 hours ago

The next step is to create a dashboard. To do this, select the "Dashboards" tab and then "New Dashboard":

Shop Learn Blog Forums **IO** LIVE! AdaBox Hi, Angelika Tefelska | Account 0

adafruit Devices Feeds **Dashboards** Actions Power-Ups 🔑 + New Device

angtef / Dashboards 🔍 ? Help

+ New Dashboard

A window will pop up where you should enter the selected dashboard name. Then on the right side, select the settings symbol and choose "Create New Block".

Shop Learn Blog Forums **IO** LIVE! AdaBox Account 0

adafruit Devices Feeds Dashboards Actions Power-Ups 🔑 + New Device

angtef / Dashboards / Test ⚙️



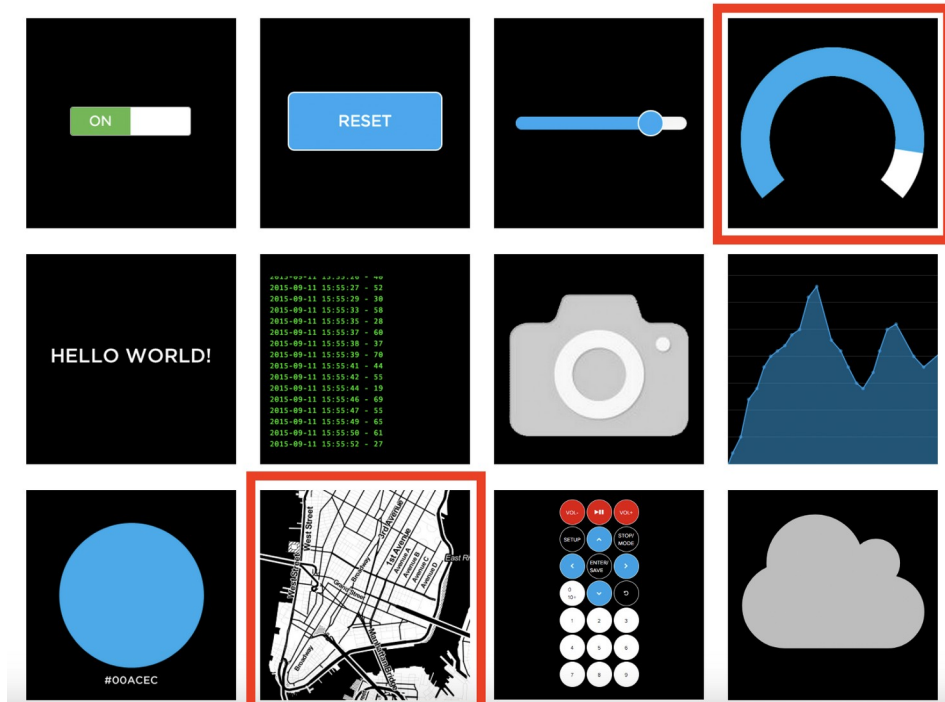


Adafruit IO has various blocks available for displaying data (text boxes, gauges, charts, maps, etc.). In our case, let's choose a gauge and a map:

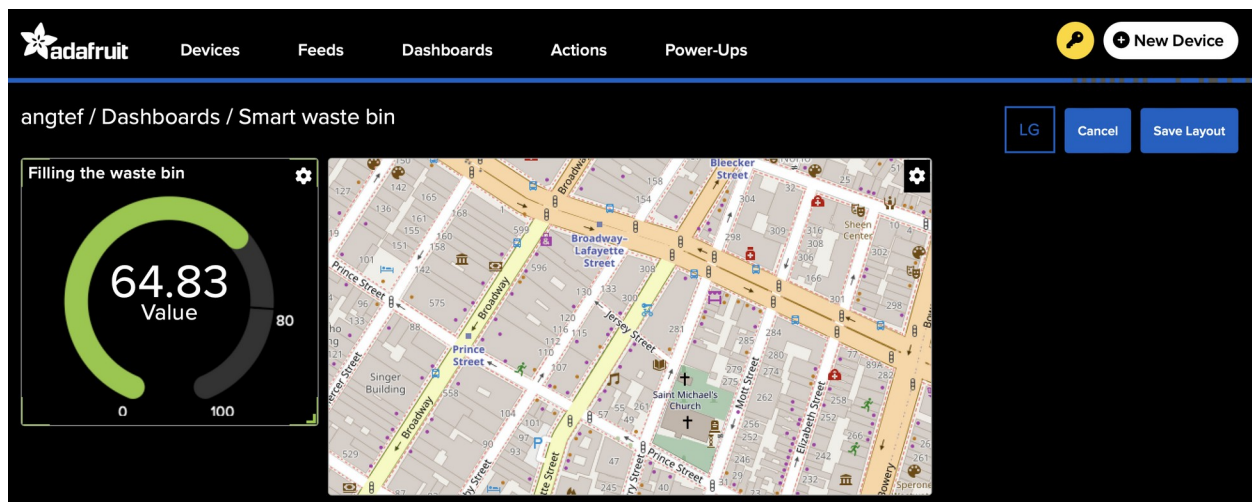
### Create a new block



Click on the block you would like to add to your dashboard. You can always come back and switch the block type later if you change your mind.

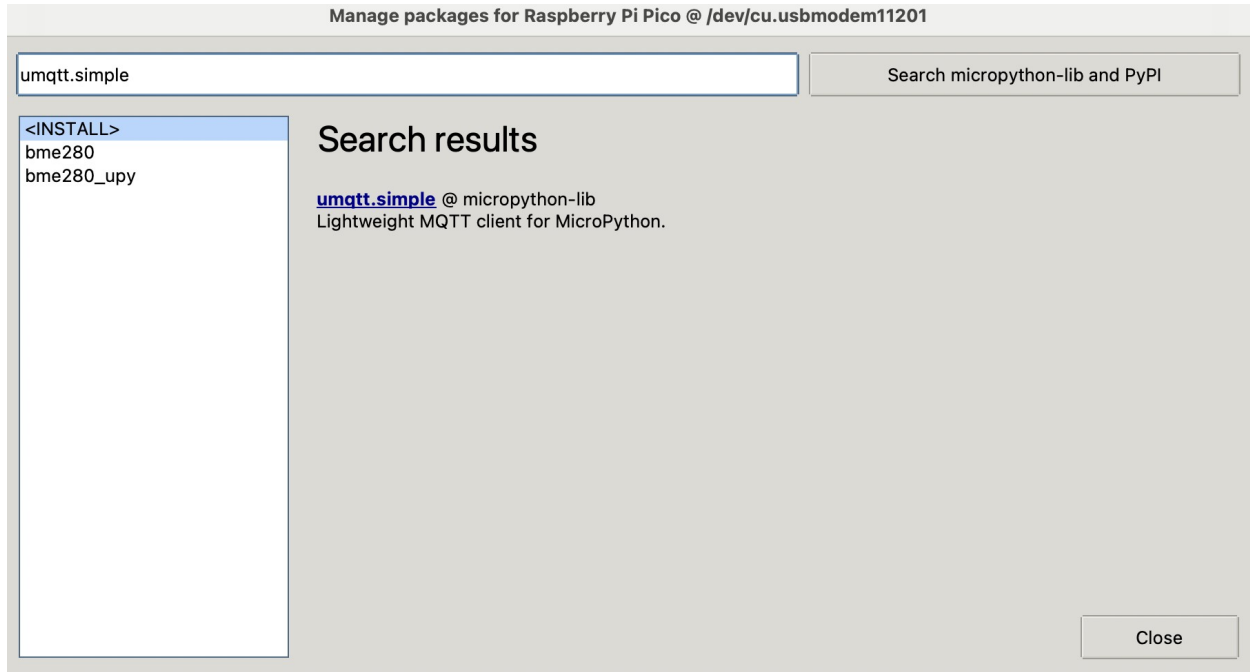


Then a window will pop up asking which feed we want to connect the block to. Select the feed defining the waste bin filling to the indicator and the feed defining the location to the map. Then select the settings icon again and choose "Edit Layout". Arrange the blocks on the screen as you see fit. You can also enlarge and reduce the blocks. An example of the appearance is shown in the screenshot below:





Now let's go to the Thonny editor and install the ***umqtt.simple*** library. To do this, select "Tools" and then "Manage packages". A window will pop up where you should enter the name of the library you want to install, in this case ***umqtt.simple***:



When you click on the name of the found ***umqtt.simple*** library, a window with library data will open and you will be able to install the library by clicking the Install button.

Now we can go back to our code and add the pieces necessary to send data to the cloud. To do this, follow these steps:

1. Add the necessary libraries (network library and import symbol MQTTClient from ***umqtt.simple*** library):

```

smart_waste_bin.py * <untitled> *
1 import machine
2 import utime
3 import network
4 from umqtt.simple import MQTTClient
5
6 trigger = machine.Pin(18, machine.Pin.OUT)
7 echo = machine.Pin(19, machine.Pin.IN)
8

```

2. Add a piece of code that will allow you to connect to your WIFI. Here you need to fill in lines 15-18. First, enter the name of your WIFI and then the password.



```

smart_waste_bin.py × backup.py
1  import machine
2  import utime
3  import network
4  from umqtt.simple import MQTTClient
5
6  trigger = machine.Pin(18, machine.Pin.OUT)
7  echo = machine.Pin(19, machine.Pin.IN)
8
9  led_green = machine.Pin(13, machine.Pin.OUT)
10 led_yellow = machine.Pin(12, machine.Pin.OUT)
11 led_red = machine.Pin(11, machine.Pin.OUT)
12
13 depth = 100
14
15 WIFI_SSID = "your_wifi_name"
16 WIFI_PASSWORD = "your_wifi_password"
17 ADAFRUIT_IO_USERNAME = "username"
18 ADAFRUIT_IO_KEY = "key"
19
20 def connect_wifi():
21     wlan = network.WLAN(network.STA_IF)
22     wlan.active(True)
23     wlan.connect(WIFI_SSID, WIFI_PASSWORD)
24     while not wlan.isconnected():
25         print("Connecting to Wi-Fi...")
26         utime.sleep(1)
27     print("Connected to Wi-Fi:", wlan.ifconfig())
28
29 connect_wifi()
30
31 while True:

```

The last two parameters are the data from Adafruit IO. Go back to the Adafruit website and click the key symbol. When you do this, your login and key will appear. Copy this data to the program to lines 17-18:

### YOUR ADAFRUIT IO KEY

Your Adafruit IO Key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your Adafruit IO Key can view all of your data, create new feeds for your account, and manipulate your active feeds.


If you need to regenerate a new Adafruit IO Key, all of your existing programs and scripts will need to be manually changed to the new key.

**Username**

**Active Key**

**REGENERATE KEY**

✕



+ New Device

LG
Cancel
Save Layout

3. Now we will use MQTT (Message Queuing Telemetry Transport), a lightweight communication protocol based on the publish/subscribe model. It was designed specifically for sending data in resource-constrained environments, such as IoT (Internet of Things) devices. To do this, use the code below, changing only the feed names in lines 33-34. The example uses feeds named: "Filling" and "Location". Replace them with your own. Just remember to leave /csv in the case of Location because when using maps, you must provide data in csv format.

```

smart_waste_bin.py * x backup.py
13 depth = 100
14
15 WIFI_SSID = "your_wifi_name"
16 WIFI_PASSWORD = "your_wifi_password"
17 ADAFRUIT_IO_USERNAME = "username"
18 ADAFRUIT_IO_KEY = "key"
19
20 def connect_wifi():
21     wlan = network.WLAN(network.STA_IF)
22     wlan.active(True)
23     wlan.connect(WIFI_SSID, WIFI_PASSWORD)
24     while not wlan.isconnected():
25         print("Connecting to Wi-Fi...")
26         utime.sleep(1)
27     print("Connected to Wi-Fi:", wlan.ifconfig())
28
29 ADAFRUIT_IO_SERVER = "io.adafruit.com"
30 ADAFRUIT_IO_PORT = 1883 # Port MQTT
31 CLIENT_ID = "raspberrypi_pico"
32
33 FEED_FILL_LEVEL = "angtef/feeds/Filling"
34 FEED_LOCATION = "angtef/feeds/Location/csv"
35
36 client = MQTTClient(CLIENT_ID, ADAFRUIT_IO_SERVER, ADAFRUIT_IO_PORT, ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)
37
38 def connect_mqtt():
39     client.connect()
40     print("Connected to Adafruit IO!")
41
42 connect_wifi()
43 connect_mqtt()

```

4. Now let's add functions to send data to the cloud:

```

38 def connect_mqtt():
39     client.connect()
40     print("Connected to Adafruit IO!")
41
42 connect_wifi()
43 connect_mqtt()
44
45 def send_data(Filling, Location):
46     client.publish(FEED_FILL_LEVEL, str(Filling))
47     print("Fill level sent:"+str(Filling))
48     client.publish(FEED_LOCATION, Location)
49     print("Location sent:"+str(Location))
50

```

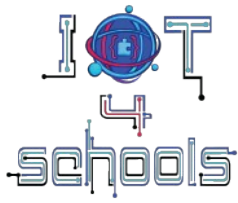
The last step is to pass the measured occupancy and location to the function in while loop after your code. You can read your location, for example, from Google maps and replace it in the "location" variable.

```

56 while True:
57     #your code
58
59     #value, latitude, longitude, altitude
60     location = "0, 52.2297,20.0122,0"
61     #fill_level - this is the measured fullness of the bin,
62     #correct the variable name to the one you used in your code
63     send_data(fill_level, location)
64

```

Test the smart waste bin. Remember to return to the Adafruit IO page on the dashboard you created to see if you are sending your data to the cloud correctly.



## 6. Wrapping up: reflecting on functionality and possible improvements

Do you think that using smart bins could improve waste collection and reduce the carbon footprint? In the project, we did not optimize the route of garbage trucks, but you can imagine that there is a system that reads data from the cloud from each bin and prepares an optimal route. . If you are interested in how to make such an algorithm, you can familiarize yourself with the programs from the website: <https://colab.research.google.com/drive/1aOq9jRh6c6fhaw1ahe0a1-yKVdMnO613?usp=sharing%2F> .

.....

.....

.....

.....

.....

What other changes can be made to make smart bins work even more efficiently?

.....

.....

.....

.....

.....