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 21st century challenges”

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

Teachers’ guidelines

Authors: Angelika Tefelska, Dariusz Tefelski, Adam Kowalczyk (fotografie)

Organization: Warsaw University of Technology, Faculty of Physics

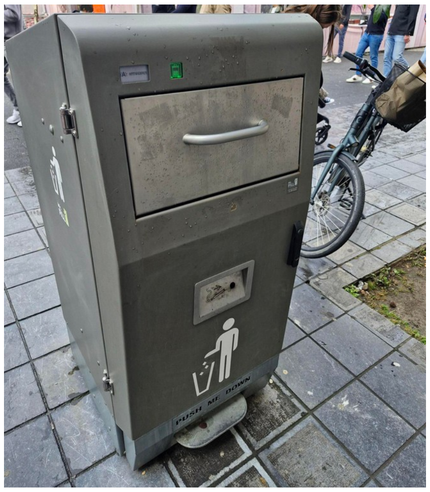
Co-funded by
the European Union

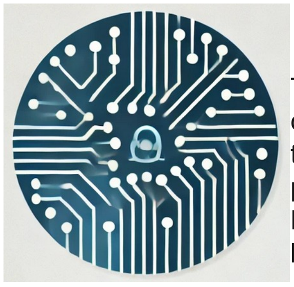
Table of contents

1 Introduction to the project

1.1 Scenario and Scope of the project

The aim of this project is to introduce students to the Internet of Things (IoT) technology in the context of smart cities. For several years, the concept of smart cities has been developed, in which IoT solutions are and will be used to improve living conditions and help minimize the negative impact of cities on the environment. One of the important problems of modern cities is garbage collection. Currently, most cities collect garbage on a set schedule, driving past all properties regardless of the level of filling of the bins. It is similar with bins that stand along the sidewalks and are used by passers-by to throw away garbage. Such a system is inefficient and causes significant fuel consumption. The project aims to show how to create an intelligent waste bin using a Raspberry Pi Pico board and an ultrasonic distance sensor to measure the level of filling of the bin. The level of filling of the bin will be sent to the cloud to help optimize the route of the garbage truck and thus minimize the carbon footprint. Such solutions are already slowly being introduced in some cities, e.g. in Antwerp (see photos below).





The project aims to develop students' digital competences. IoT technology combines many fields: electronics (measuring sensors), programming (in this case, the MicroPython programming language - a version of Python dedicated to programming microcontrollers), networking (using clouds) and data analysis. During the project, students will develop the digital and technological competences listed above.

As part of the project, students will learn about the problem of waste management in the city, which is an extremely important issue to minimize the negative impact of cities on the environment by reducing the carbon footprint (less fuel consumption by garbage trucks). In addition, the topic of recycling will be discussed.

The project will be carried out in teams by students from different backgrounds and countries. Through joint cooperation, we hope that students will integrate despite cultural differences or language barriers.

1.2 Learning objectives

Through this project the students will be able to:

- Create a simple program in the MicroPython programming language.
- Build an electronic circuit using the Raspberry Pi Pico board.
- Build and program a device that can measure the filling of waste bins.
- Learn how to use sensors such as the ultrasonic sensor.
- Understand how IoT devices can collect and transmit data to the cloud.
- Understand and explain how data can be monitored in real time.
- Indicate solutions aimed at more efficient and ecological waste management in cities.
- Indicate the advantages of recycling and current difficulties in waste processing.

1.3 Learning pathway – Stages of implementation

As part of the project, students will create solutions to improve waste management in cities to minimize their carbon footprint. To do this, they will use a Raspberry Pi Pico board and an ultrasonic distance sensor.

Here are some suggested stages to smoothly and effectively implement the smart waste bin project with your students:

1. **Group formation:** Divide your students into teams of two or three.
2. **Brainstorming:** Encourage each team to search for information about possible ways to manage waste in the city to make it smarter (method of waste collection, route optimization, advantages and disadvantages of different solutions, recycling statistics, fuel consumption of garbage trucks, etc.).

3. **Discussion and assignment of the activity:** Encourage each team to share their findings and ideas on how to improve the waste management system to make it smarter. Following the discussion, introduce the specific aim of the project. (Note: It is recommended that the specific aim of the project is introduced after the brainstorming in order to encourage your students to consider the smart waste bin project in a broader context).

4. **Planning:** Encourage each team to think about how they will construct the device (how the bin will be made; where to place the distance sensor; what data should be transmitted to the cloud, when data should be sent and how often).

5. **Creation:** Using the student worksheets, encourage each team to create their own smart waste bin. Depending on the skills of the students, you may wish to consider role allocation.

6. **Testing - optimization:** After completing the project, encourage your students to test their smart waste bin. Based on the test results, you can encourage each team to optimize their project. If the project was not too difficult for the students, consider doing extensions, where you develop an algorithm to optimize the route (the traveling salesman problem).

7. **Presentation - sharing:** Encourage your students to present their projects in plenary and ask them to reflect on the whole experience. Encourage all teams to consider the impact of such smart waste bins on the functioning of cities and the environment.

1.4 Learning prerequisites

Students should be familiar with the basics of programming in any programming language. If they do not have such experience, it is still possible to complete the project because the materials are prepared in such a way that anyone can complete the project.

1.5 Hardware and software

Hardware:

- Raspberry Pi Pico W board
- Breadboard (contact plate)
- Female – Male and Male – Male connection wires
- Red, Yellow and Green LED diodes
- 330 Ohm resistors
- External power source: 2AAA battery holder or a power bank of low voltage output (up to 5V) - optional
- Ultrasonic distance sensor HC-SR04

Software:

- Thonny editor
- Adafruit IO cloud

Co-funded by the European Union

1.6 Time plan

It is estimated that you will need 4 to 6 hours to complete the project

In particular, it is estimated that you will need:

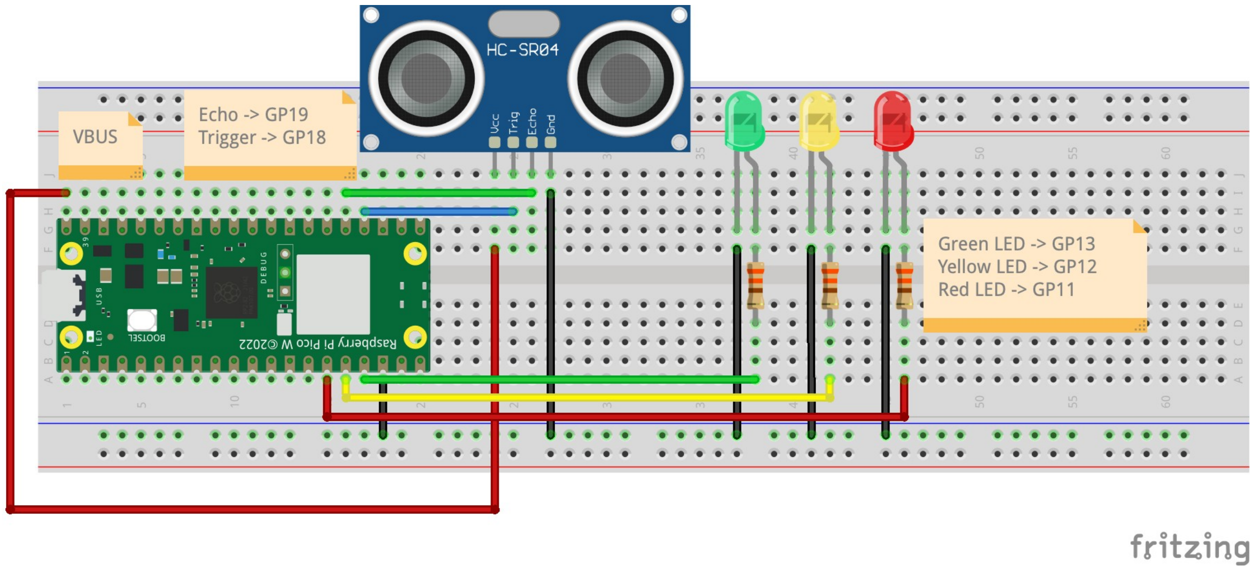
- 45-90 minutes to introduce the project (including brainstorming and discussion), planning and warm-up activity
- 45 minutes to complete Level 1
- 45 minutes for level 2
- 45-90 minutes for extensions
- 30 minutes for wrap-up and discussion

2 Implementation of the project

2.1 Level 1

2.1.1 Circuit making process

Raspberry Pi Pico W should be connected with an ultrasonic distance sensor and three LEDs via 330Ω resistors. An example connection is shown in the figure below. At first, you can test the system connected to the computer via USB. Ultimately, however, you will need a power bank or batteries to power the system. For ecological and economic reasons, we recommend a power bank, which can be charged after the classes and used again.



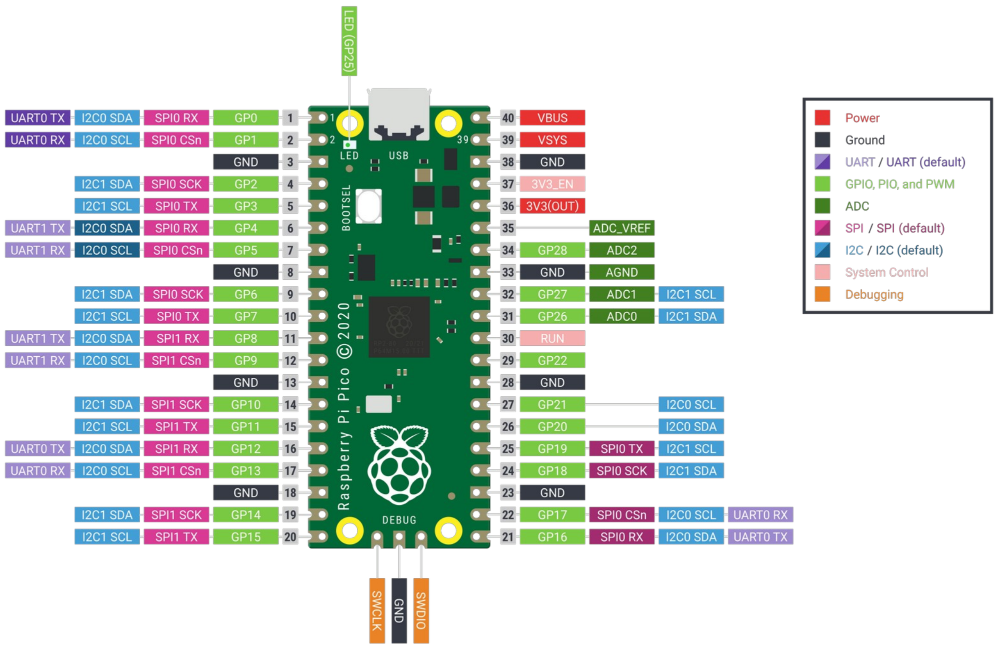
Note that on some breadboards the pins under the blue line that we used for GND are not connected along the entire length of the board. If there is a gap in the blue stripe, it means that only part of the board has the pins connected. Pay attention to this when connecting the board.

2.1.2 Programming

In this project, we will be using general-purpose input/output (GPIO) pins, which are used to generate or read digital signals, i.e. those that have only two possible values: 0 or 1. GPIO are pins that can be used both as input pins, from which we can read information, e.g. about turning on a button, or as output pins, on which we can generate digital signals, e.g. to blink a LED. The figure below shows the 40 pins from the Raspberry Pi Pico board, which are numbered from the top left. Pins serving as GPIO are abbreviated GPx (green rectangles), where x is the ordinal number of the GPIO pin, starting from 0 to 28. More information in the guide: "Technical guide about Raspberry Pi Pico and MicroPython" available at: https://www.iot.fizyka.pw.edu.pl/results/ .

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

Co-funded by the European Union

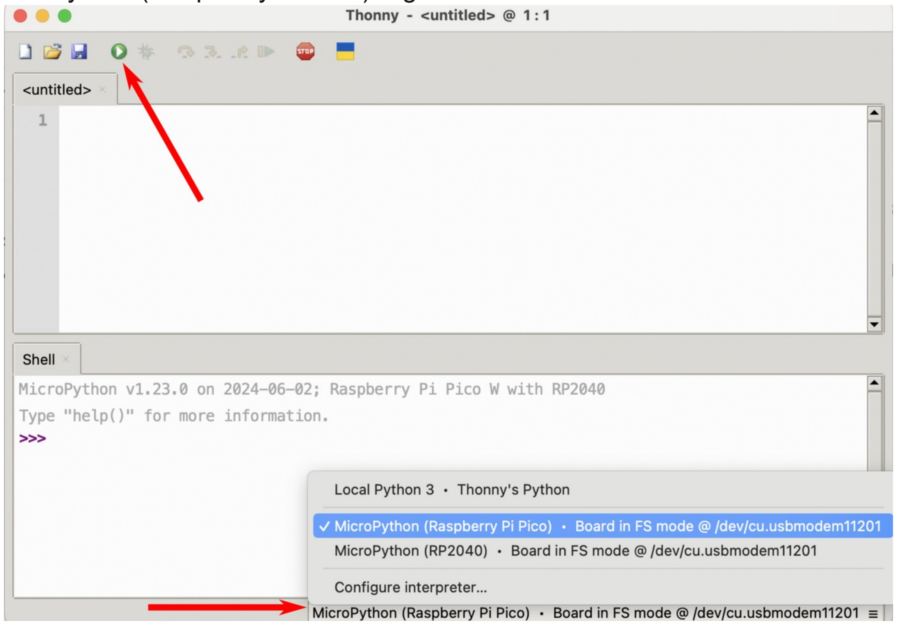

LED (GP25)
USB
LED
BOOTSEL
Raspberry Pi Pico ©2020
DEBUG
UART0 TX | I2C0 SDA | SPI0 RX | GP0 | 1
UART0 RX | I2C0 SCL | SPI0 CSn | GP1 | 2
GND | 3
I2C1 SDA | SPI0 SCK | GP2 | 4
I2C1 SCL | SPI0 TX | GP3 | 5
UART1 TX | I2C0 SDA | SPI0 RX | GP4 | 6
UART1 RX | I2C0 SCL | SPI0 CSn | GP5 | 7
GND | 8
I2C1 SDA | SPI0 SCK | GP6 | 9
I2C1 SCL | SPI0 TX | GP7 | 10
UART1 TX | I2C0 SDA | SPI1 RX | GP8 | 11
UART1 RX | I2C0 SCL | SPI1 CSn | GP9 | 12
GND | 13
I2C1 SDA | SPI1 SCK | GP10 | 14
I2C1 SCL | SPI1 TX | GP11 | 15
UART0 TX | I2C0 SDA | SPI1 RX | GP12 | 16
UART0 RX | I2C0 SCL | SPI1 CSn | GP13 | 17
GND | 18
I2C1 SDA | SPI1 SCK | GP14 | 19
I2C1 SCL | SPI1 TX | GP15 | 20
40 | VBUS
39 | VSYS
38 | GND
37 | 3V3_EN
36 | 3V3(OUT)
35 | ADC_VREF
34 | GP28 | ADC2
33 | GND | AGND
32 | GP27 | ADC1 | I2C1 SCL
31 | GP26 | ADC0 | I2C1 SDA
30 | RUN
29 | GP22
28 | GND
27 | GP21 | I2C0 SCL
26 | GP20 | I2C0 SDA
25 | GP19 | SPI0 TX | I2C1 SCL
24 | GP18 | SPI0 SCK | I2C1 SDA
23 | GND
22 | GP17 | SPI0 CSn | I2C0 SCL | UART0 RX
21 | GP16 | SPI0 RX | I2C0 SDA | UART0 TX
SWCLK
GND
SWDIO
🟥 Power
⬛ Ground
▨ UART / UART (default)
🟩 GPIO, PIO, and PWM
🟩 ADC
▨ SPI / SPI (default)
▨ I2C / I2C (default)
🟥 System Control
🟧 Debugging

Source: https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html .

In order for students to do as much as possible on their own, we suggest doing two warm-up exercises first.

Warm-up activity 1:

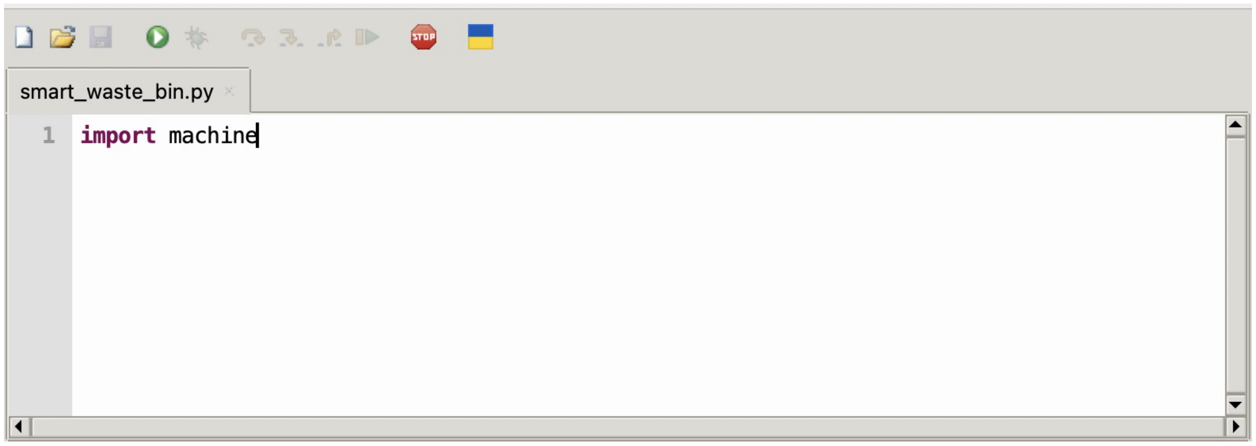
The first warm-up exercise is to create a program that will simulate traffic lights. To do this, open the Thonny editor and then select "MicroPython (Raspberry Pi Pico)" as shown in the picture. After connecting to the board correctly, the green Run button should be active (not grayed out). If it is grayed out, select "MicroPython (Raspberry Pi Pico)" again.


Thonny - <untitled> @ 1 : 1
<untitled>
Shell
Local Python 3 • Thonny's Python
✓ MicroPython (Raspberry Pi Pico) • Board in FS mode @ /dev/cu.usbmodem11201
MicroPython (RP2040) • Board in FS mode @ /dev/cu.usbmodem11201
Configure interpreter...
MicroPython (Raspberry Pi Pico) • Board in FS mode @ /dev/cu.usbmodem11201

```
MicroPython v1.23.0 on 2024-06-02; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>
```

Now let's write a program for traffic lights. To do this, follow these steps:

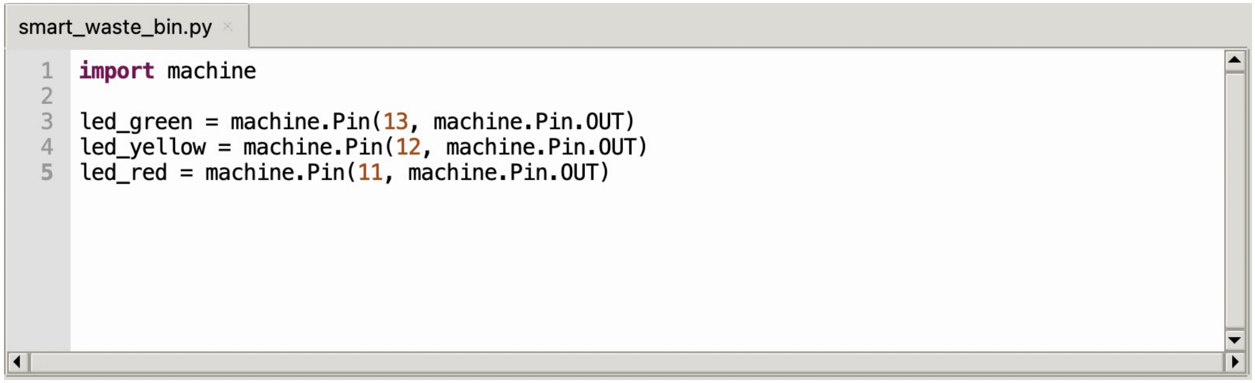
1. Let's start by including the library in MicroPython:



```
import machine
```

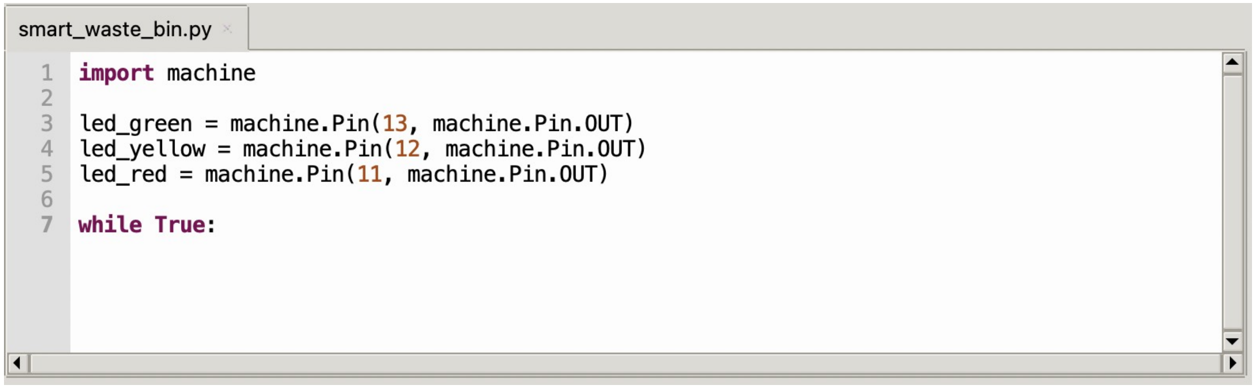
The *machine* library contains all the necessary instructions for communicating with the Raspberry Pi Pico. The command import includes the specified library in the project.

2. Next, you need to configure the GP13, GP12 and GP11 pins to work as an output pins because we control the LEDs by sending a value of 1 or 0 to the GPIO pins. The Pin function from the machine library is used for this purpose. To call functions from library in the Micropython language, first we provide the name of the library and after the dot the name of the function from this library, i.e. *machine.Pin()*. The Pin function takes two arguments. The first one is the GPIO pin number. The second one is the specification of whether the GPIO pin will work as an input (*machine.Pin.IN*) or output (*machine.Pin.OUT*). Hence, in this case the function call would look like *machine.Pin(13, machine.Pin.OUT)*. The object returned by the Pin function was assigned to the created variable *led_green*.



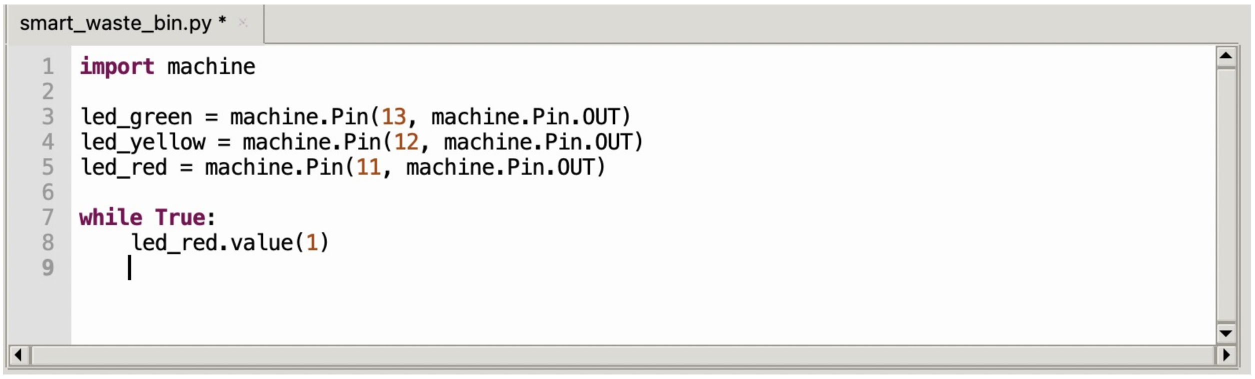
```
import machine

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)
```

3. In the next step we place an infinite loop, which will contain the main part of the program:


smart_waste_bin.py

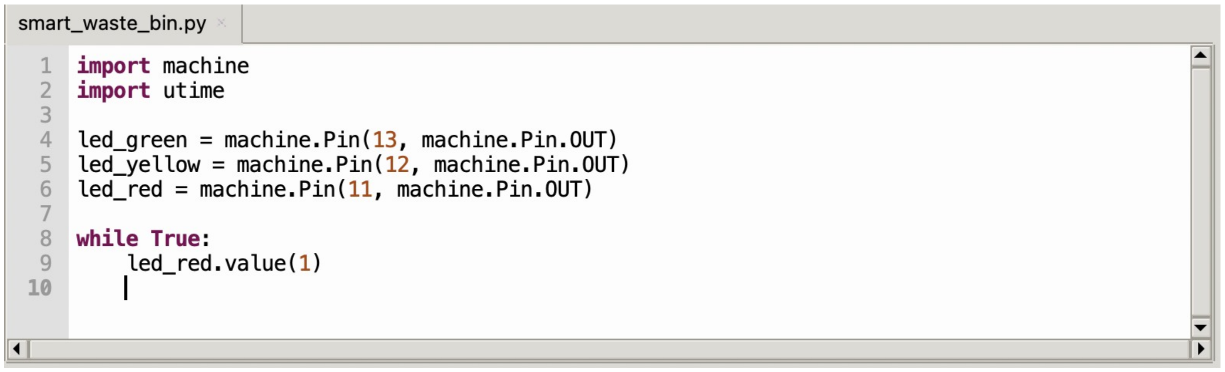
```python
import machine

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)

while True:
```

4. In the next step we light up the red LED by sending the value 1 to the GP11 pin. The value() function is used to set the value on the pin, which takes the value 0 or 1 as an argument.


smart_waste_bin.py *

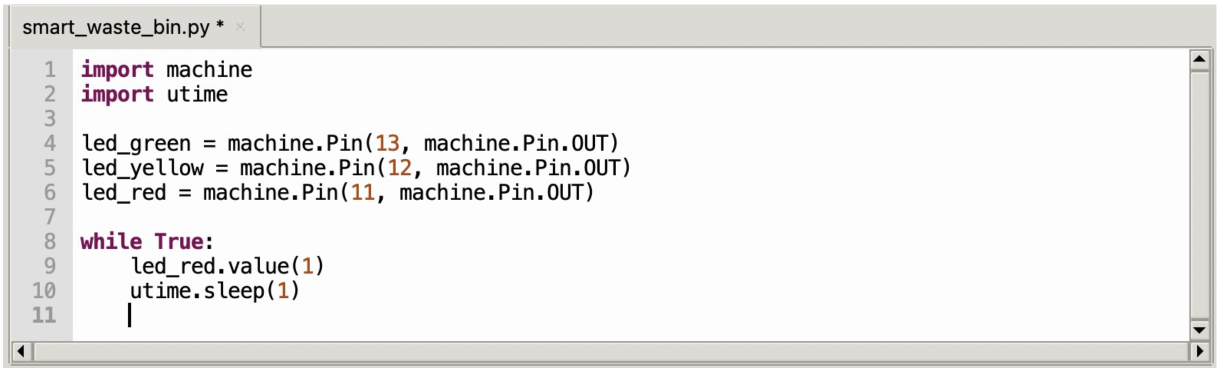
```python
import machine

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)

while True:
    led_red.value(1)
    
```

5. Now the program should wait 1s so that we can see the effect of lighting up the diode. For this purpose we will use the *utime* library, which contains functions for delays. First, we need to add the library:


smart_waste_bin.py

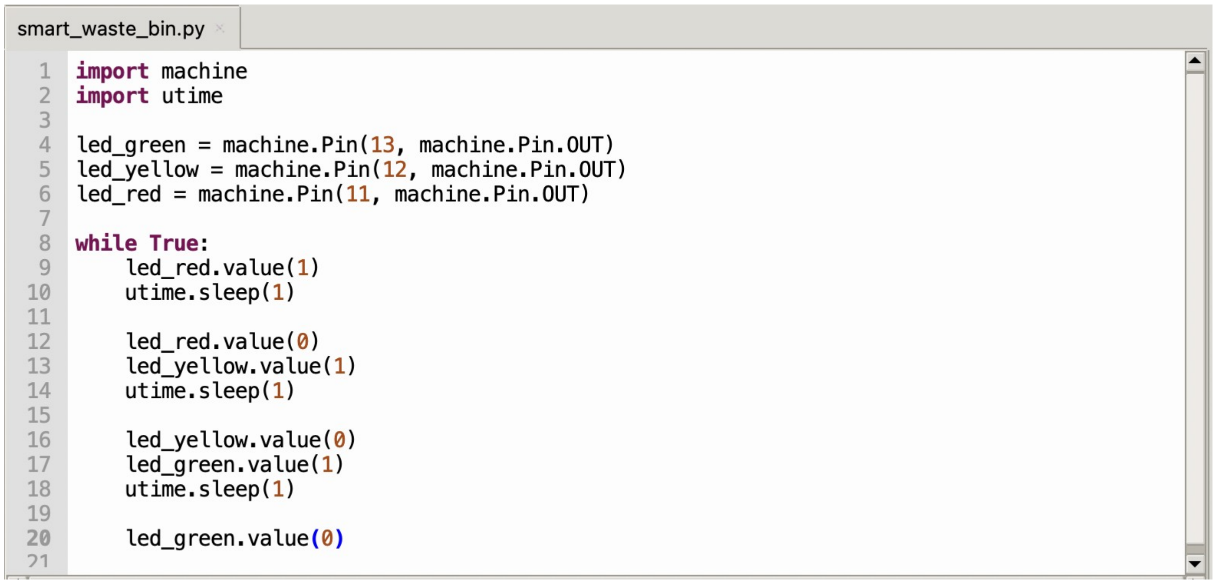
```python
import machine
import utime

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)

while True:
    led_red.value(1)
    
```

6. The *sleep* function from the *utime* library allows you to delay the program for a selected number of seconds. Let’s set a delay of 1s after the diode lights up:


smart_waste_bin.py *

```python
import machine
import utime

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)

while True:
    led_red.value(1)
    utime.sleep(1)
    
```

7. Now let's add the rest of the code that will light up the remaining LEDs in the correct order.


smart_waste_bin.py

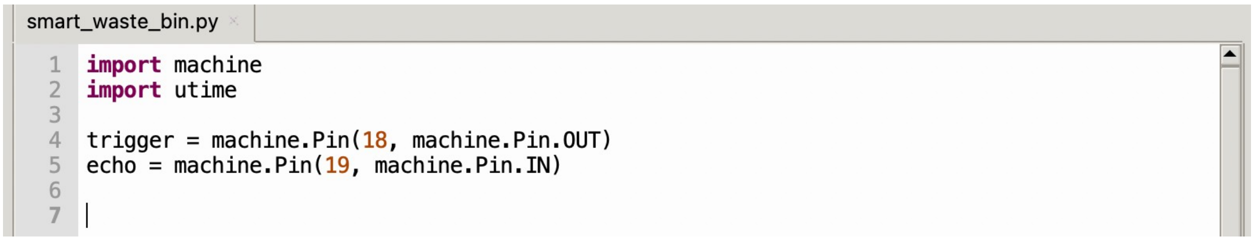
```python
import machine
import utime

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)

while True:
    led_red.value(1)
    utime.sleep(1)

    led_red.value(0)
    led_yellow.value(1)
    utime.sleep(1)

    led_yellow.value(0)
    led_green.value(1)
    utime.sleep(1)

    led_green.value(0)
```

Thanks to this activity, students will learn how to use LEDs, which we will use to create a smart waste bin.

Warm-up activity 2:

Now let's create a program that will read the distance using an ultrasonic distance sensor. To do this, follow these steps:

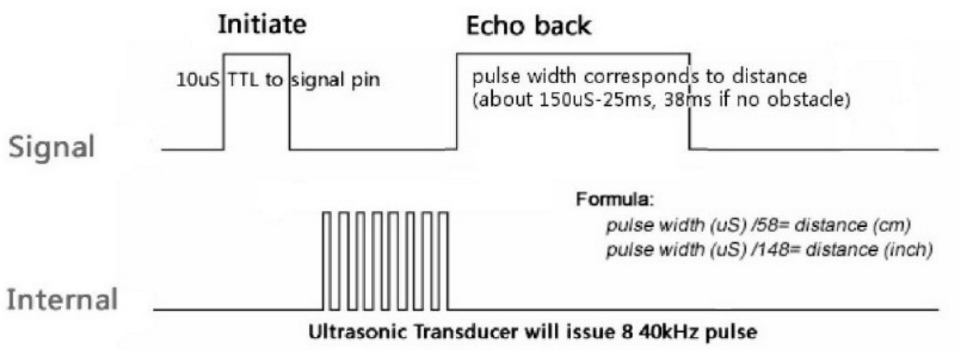
1. First, let's add the necessary libraries, i.e. machine and utime, and configure the pins: GP18 (trigger) as output, and GP19 (echo) as input.


smart_waste_bin.py

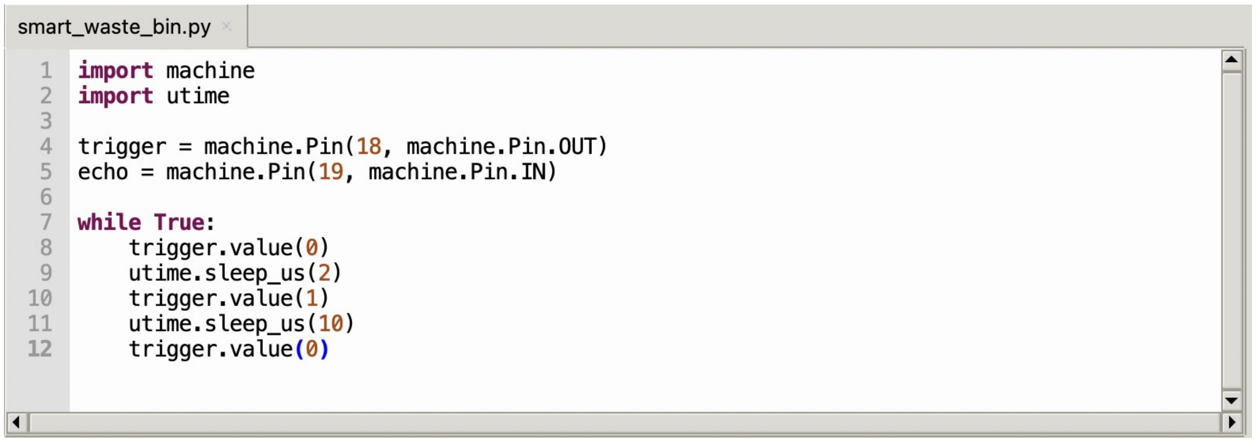
```python
import machine
import utime

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)

```

2. Next, in the main loop, let's add a code fragment to measure the distance from the ultrasonic distance sensor. According to the documentation for the HC-SR04 sensor (see the figure below), first set the low

signal 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.

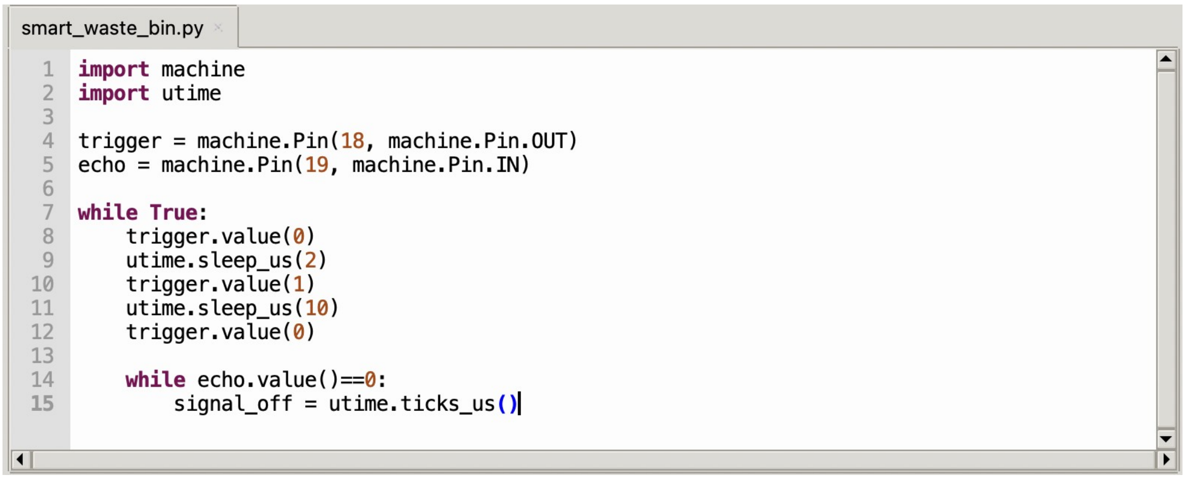

Initiate
Echo back
10uS TTL to signal pin
pulse width corresponds to distance
(about 150uS-25ms, 38ms if no obstacle)
Signal
Formula:
pulse width (uS) /58= distance (cm)
pulse width (uS) /148= distance (inch)
Internal
Ultrasonic Transducer will issue 8 40kHz pulse

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



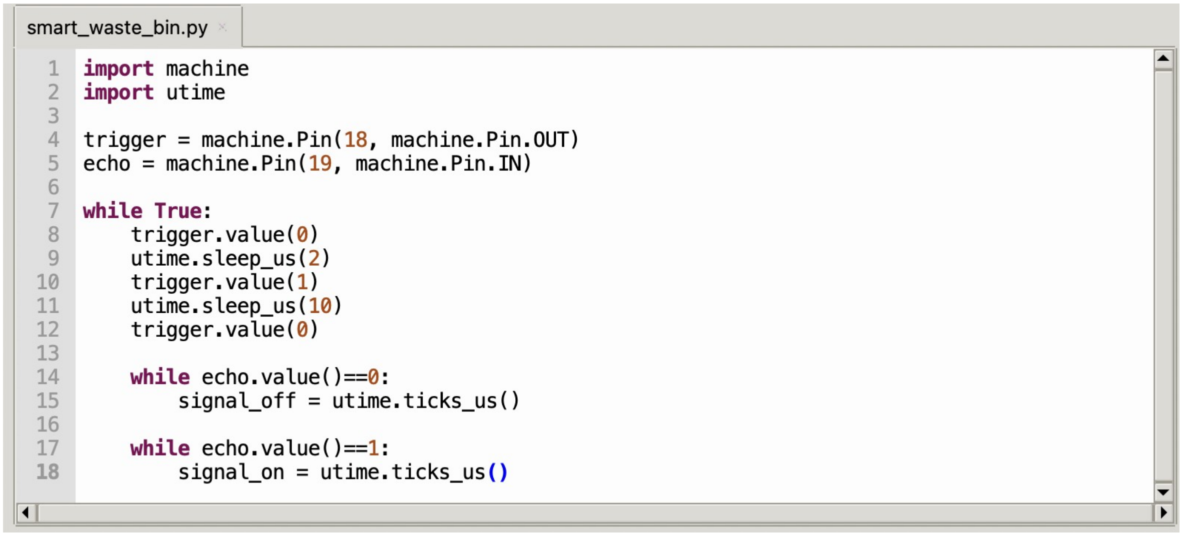
```
import machine
import utime

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)

while True:
    trigger.value(0)
    utime.sleep_us(2)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)
```

3. Now we need to measure how long the high signal on the echo pin lasted, 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.



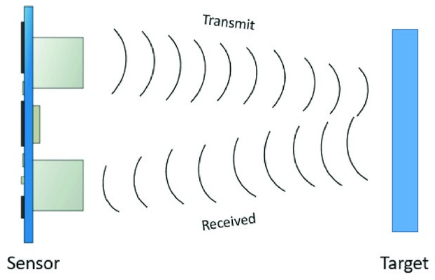
```
import machine
import utime

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)

while True:
    trigger.value(0)
    utime.sleep_us(2)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)

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

4. Similarly, we will measure when the signal was last high on the echo pin:


smart_waste_bin.py

```python
import machine
import utime

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)

while True:
    trigger.value(0)
    utime.sleep_us(2)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)

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

    while echo.value()==1:
        signal_on = utime.ticks_us()
```

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? Look at the figure below, showing the idea of measuring distance with an ultrasonic distance sensor.



Source: https://www.researchgate.net/figure/ A-block-diagram-of-Ultrasonic-sensor-working-principles_fig5_ 344385811

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:

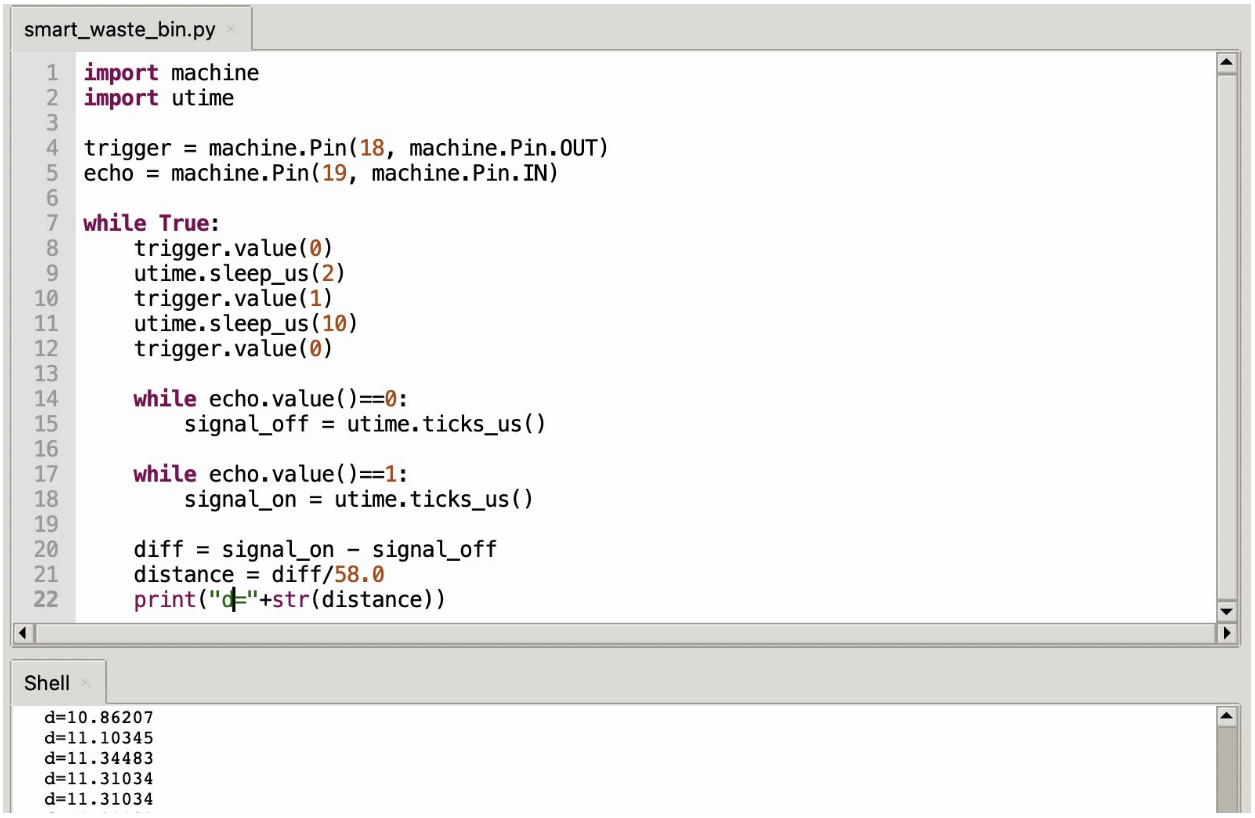
$$v=\frac{2d}{t}$$

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

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

Co-funded by the European Union

Hence, the obtained pulse duration should be divided by 58 to get the distance in centimeters. To display a value in the Thonny editor terminal, you need to use the *print* function. The *str* function converts a floating point variable to a string, which is necessary to display the value in the terminal. The plus sign allows you to combine your own text with variables:


smart_waste_bin.py

```
import machine
import utime

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)

while True:
    trigger.value(0)
    utime.sleep_us(2)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)

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

    while echo.value()==1:
        signal_on = utime.ticks_us()

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

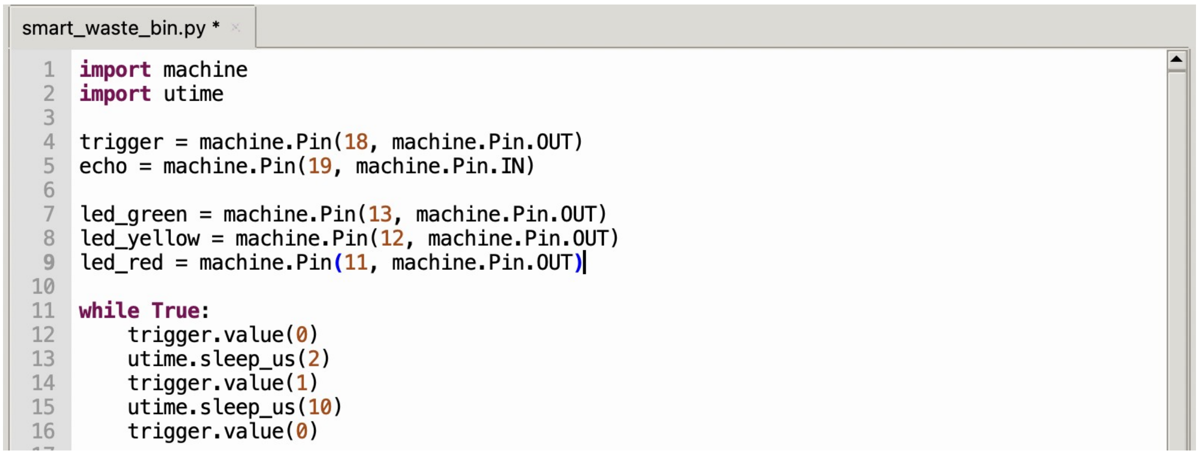
Shell

```
d=10.86207
d=11.10345
d=11.34483
d=11.31034
d=11.31034
```

Now the students can use all the elements necessary to build a smart waste bin. Let's move on to the project.

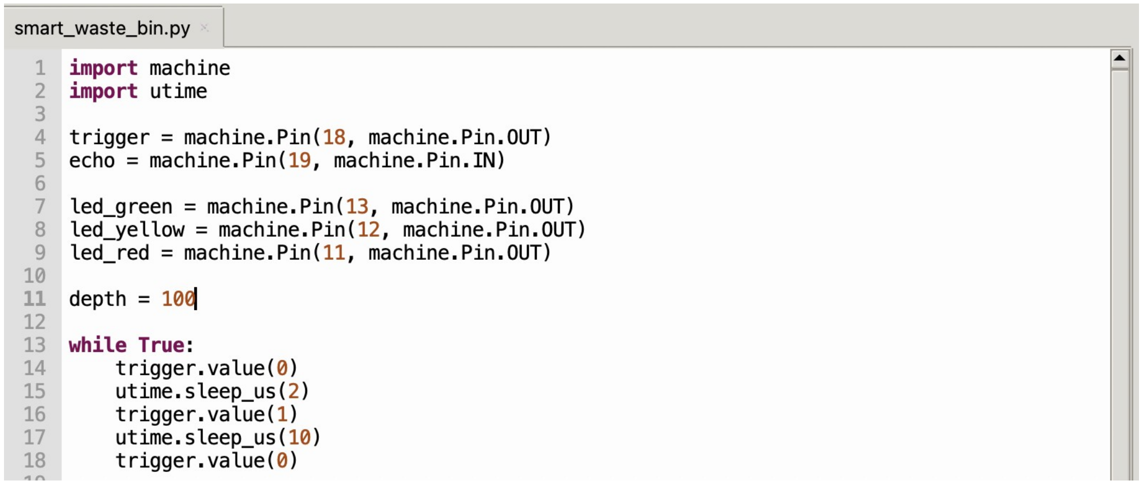
Smart waste bin project:

When creating a smart waste bin project, we will start by modifying the program from warm-up activity 2. To do this, we will add three LED diodes:


smart_waste_bin.py *

```
import machine
import utime

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)

while True:
    trigger.value(0)
    utime.sleep_us(2)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)
```

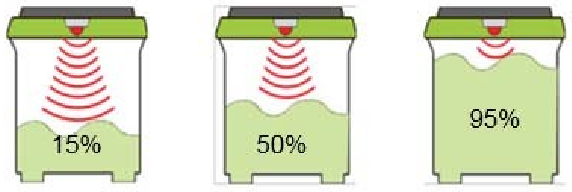
Next, let's create a variable depth that will reflect the depth of the trash can. Let's temporarily set the value to 100 cm and in a later step we'll adjust it to the actual depth of the trash can we'll create:



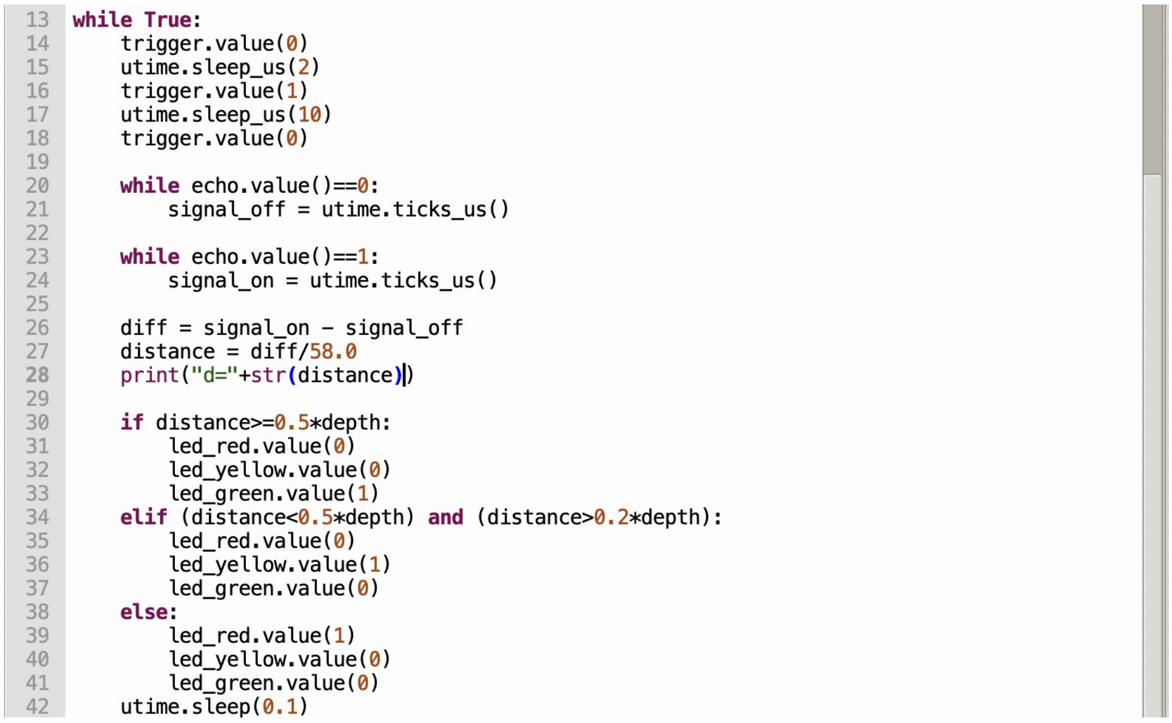
smart_waste_bin.py

```python
import machine
import utime

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)

depth = 100

while True:
    trigger.value(0)
    utime.sleep_us(2)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)
```

Now, depending on how full the bin is, we will light up a different LED. When it is below 50% full, the green diode will light up, when it is above 50% full but less than 80% full, the yellow diode will light up, and when it is over 80% full, the red diode will light up. Remember that 0% full is for the depth of the bin, i.e. in this example 100 cm, and 50% full will be when we read the distance of 50 cm by the sensor, because the sensor will be placed in the bin cover at the very top.



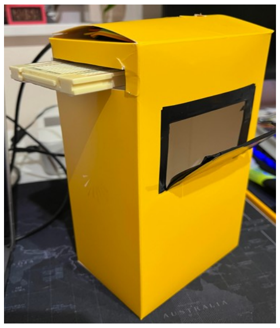
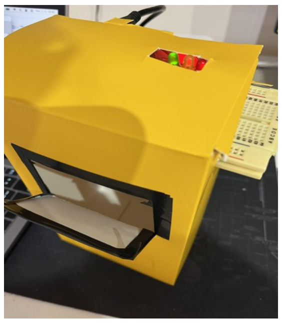
Source: https://www.researchgate.net/figure/Ultrasonic-fill-level-sensor_fig3_304711436



```python
while True:
    trigger.value(0)
    utime.sleep_us(2)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)

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

    while echo.value()==1:
        signal_on = utime.ticks_us()

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

    if distance>=0.5*depth:
        led_red.value(0)
        led_yellow.value(0)
        led_green.value(1)
    elif (distance<0.5*depth) and (distance>0.2*depth):
        led_red.value(0)
        led_yellow.value(1)
        led_green.value(0)
    else:
        led_red.value(1)
        led_yellow.value(0)
        led_green.value(0)
    utime.sleep(0.1)
```

2.1.3 Crafting

Now it's time to create your own smart garbage container. To do this, use any shipping/shoe box, etc. . Place a breadboard with an ultrasonic sensor in the upper cover. Remember to point the sensor down. Cut a hole for the trash in the front. You can place the diodes either in the top of the bin or by the trash hole. You can connect the power bank to the garbage container or use the power supply from your computer's USB. An example implementation is shown in the photos below. Remember that after constructing the smart garbage container, you should read what distance (depth) the sensor returns when the garbage container is empty, and correct the value in the *depth* variable. Otherwise, the smart garbage container will show the wrong filling.

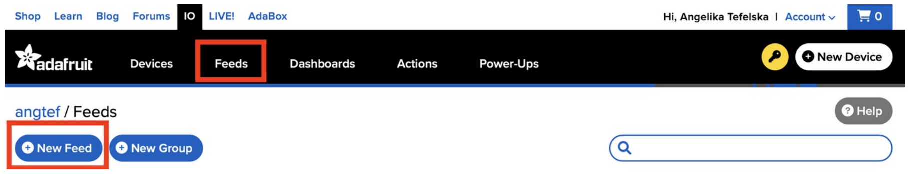



2.2 Level 2

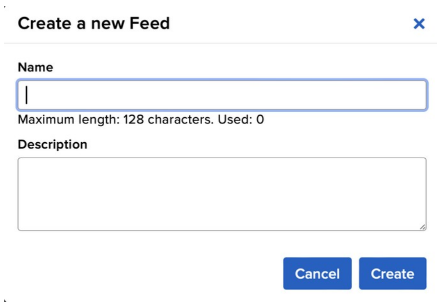
In level 2, we will add sending data about the filling of the garbage container and about the garbage container location to the cloud. We will manually enter the location in a variable in the program, assuming that the garbage container will always belong to a given house/block. Below is a step-by-step description of the program.

2.2.1 Programming

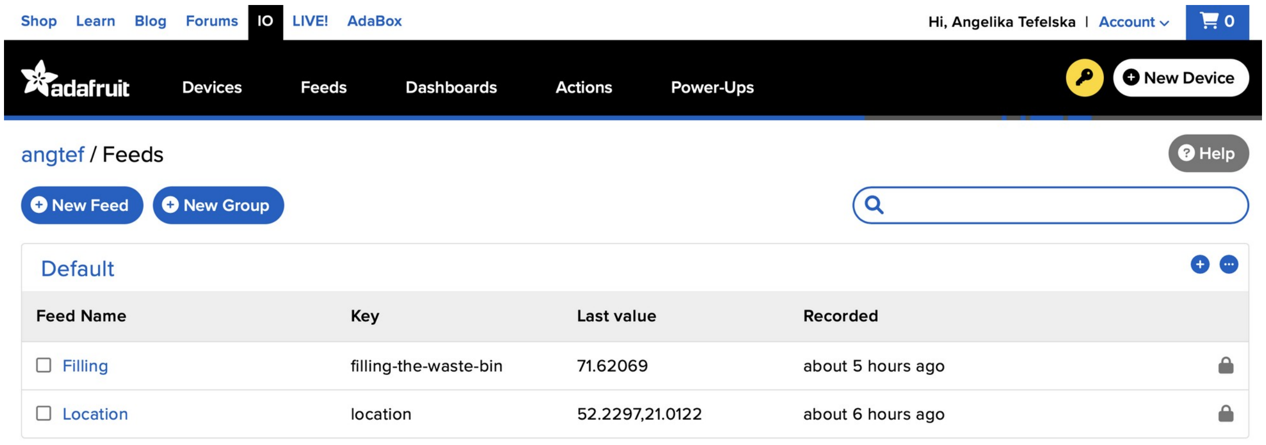
There are a few different clouds you can use. However, we recommend using Adafruit IO. First, you need to create a free account at https://io.adafruit.com . Next, you will want to send two pieces of data: garbage container filling and location to the cloud. To do this, you need to create two feeds. Feeds 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.

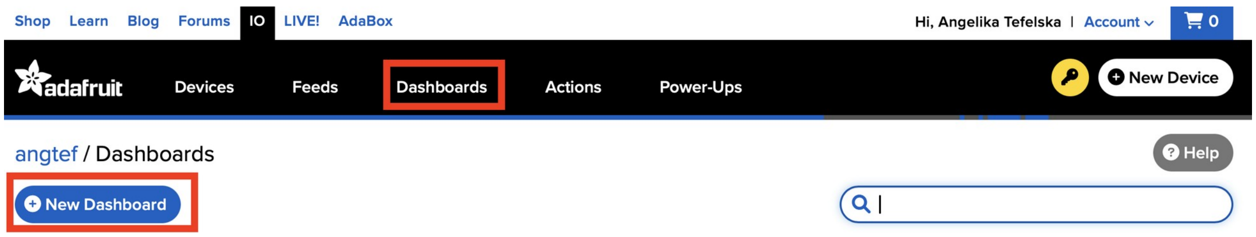


Create two feeds. Once you do that, you should see the feeds you created on your page, similar to the screenshot below:

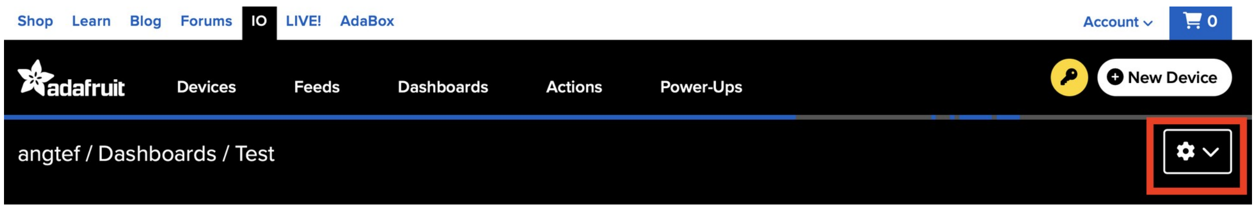


Feed Name	Key	Last value	Recorded
Filling	filling-the-waste-bin	71.62069	about 5 hours ago
Location	location	52.2297,21.0122	about 6 hours ago

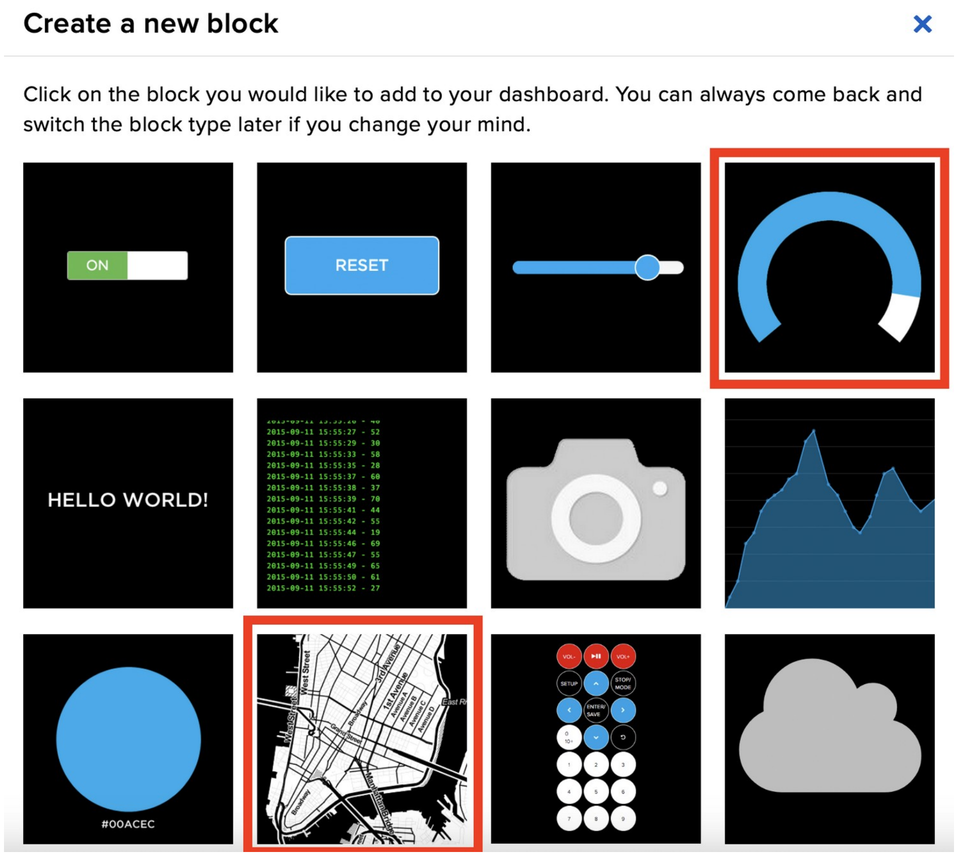
The next step is to create a dashboard. To do this, select the "Dashboards" tab and then "New Dashboard":



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".

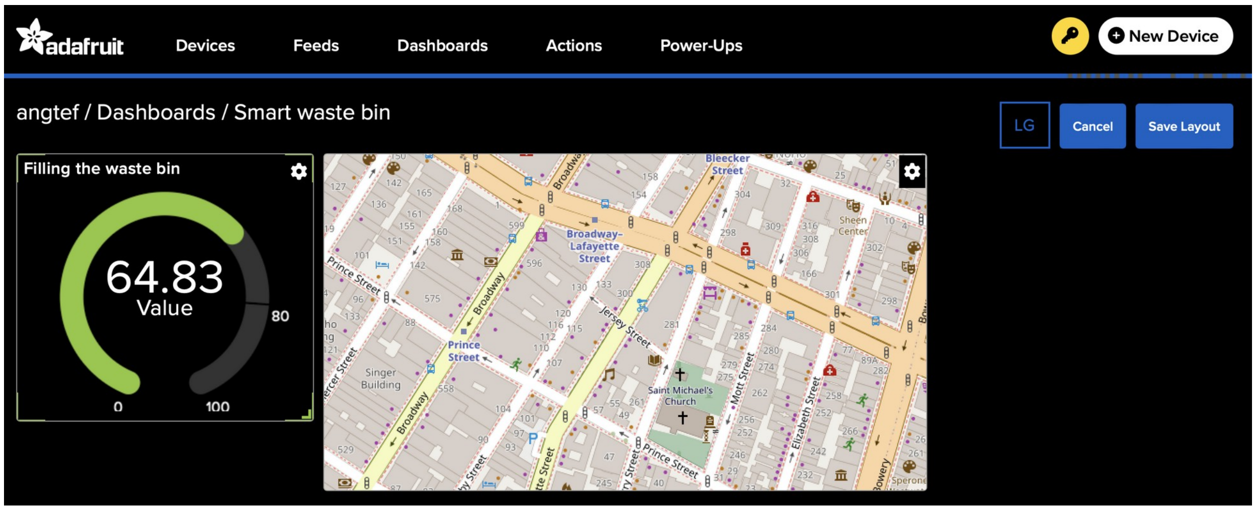


Adafruit IO has various blocks available for displaying data (text boxes, gauges, charts, maps, etc.). In our case, let's choose a gauge and a map:

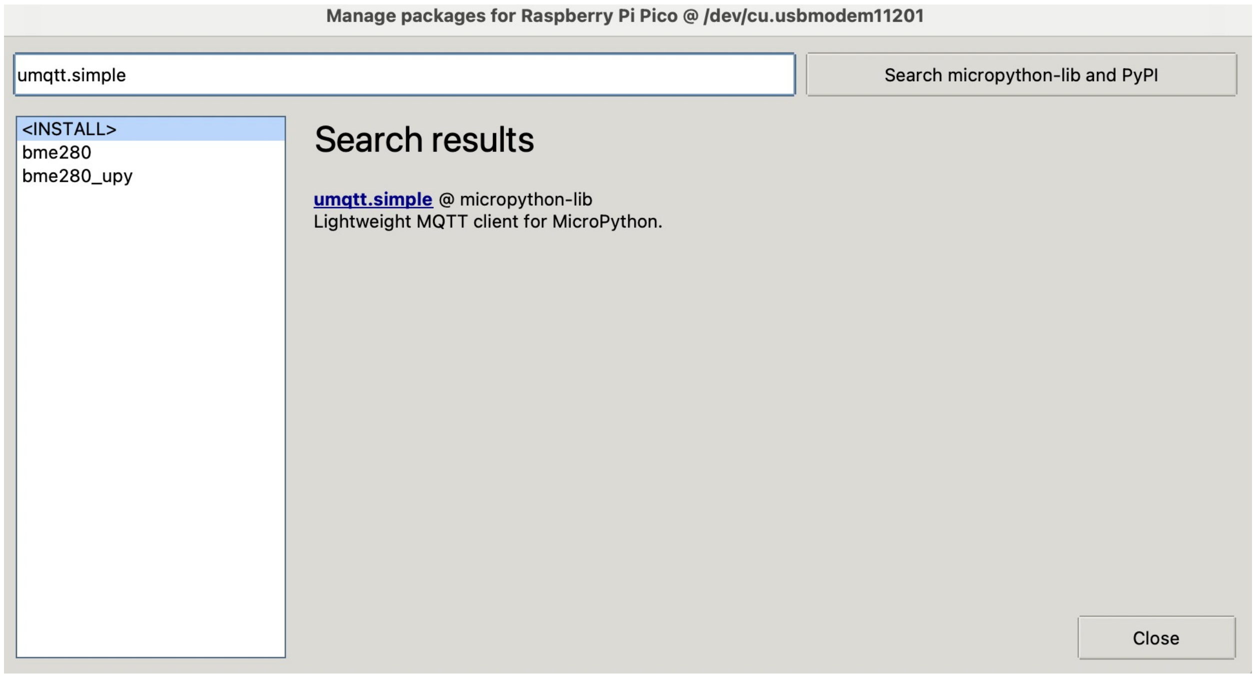

Create a new block

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*:


Manage packages for Raspberry Pi Pico @ /dev/cu.usbmodem11201
umqtt.simple
Search micropython-lib and PyPI
<INSTALL>
bme280
bme280_upy
Close

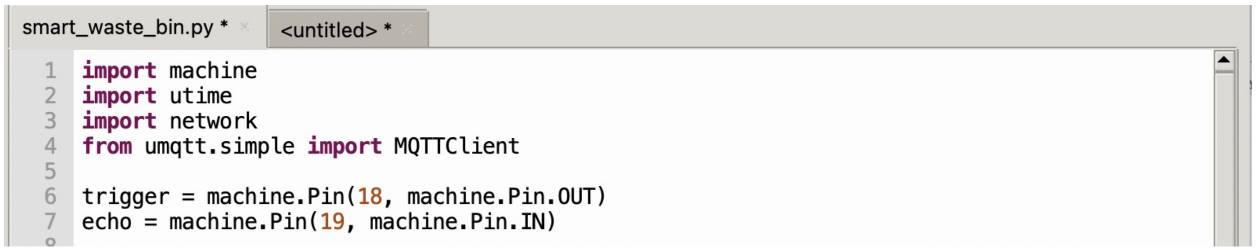
Search results

umqtt.simple @ micropython-lib
Lightweight MQTT client for MicroPython.

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.

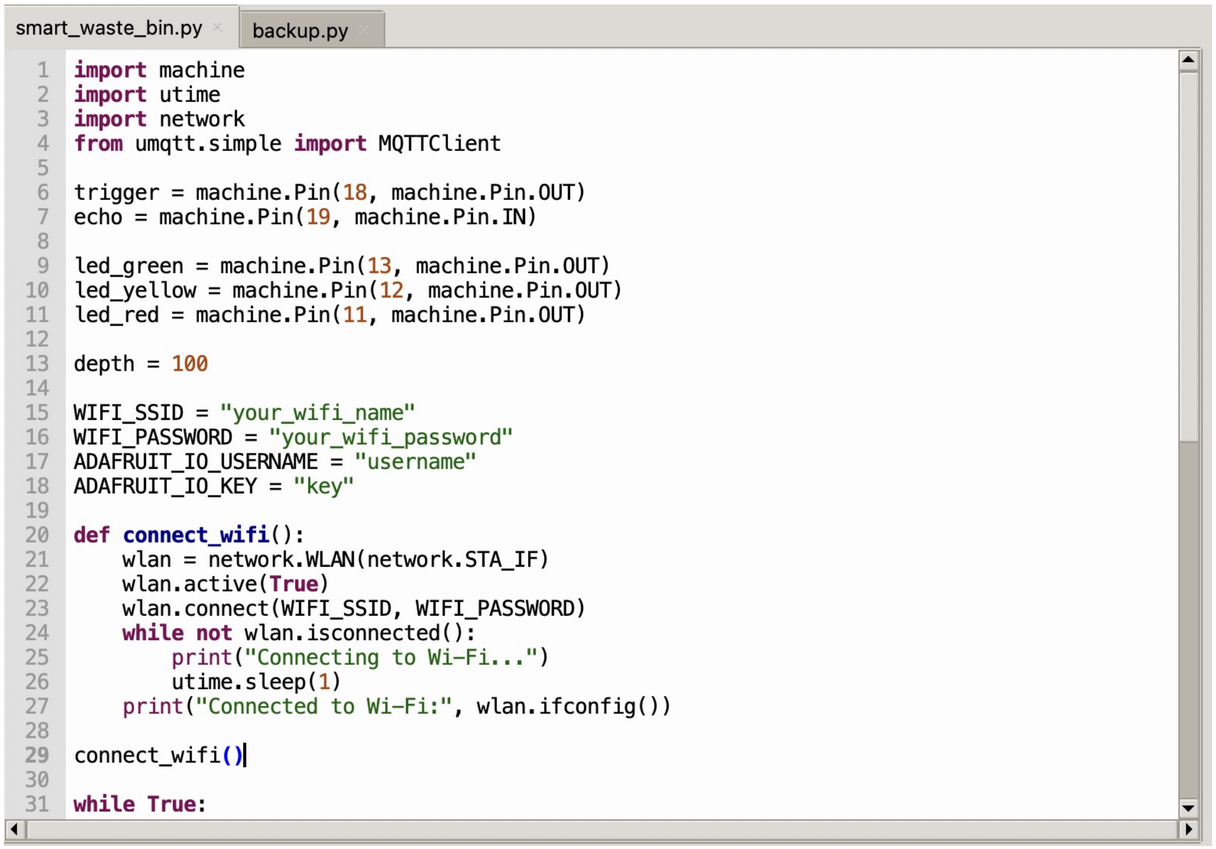
Now we can go back to our code and add the pieces necessary to send data to the cloud. To do this, follow these steps:

1. Add the necessary libraries:



```
import machine
import utime
import network
from umqtt.simple import MQTTClient

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)
```

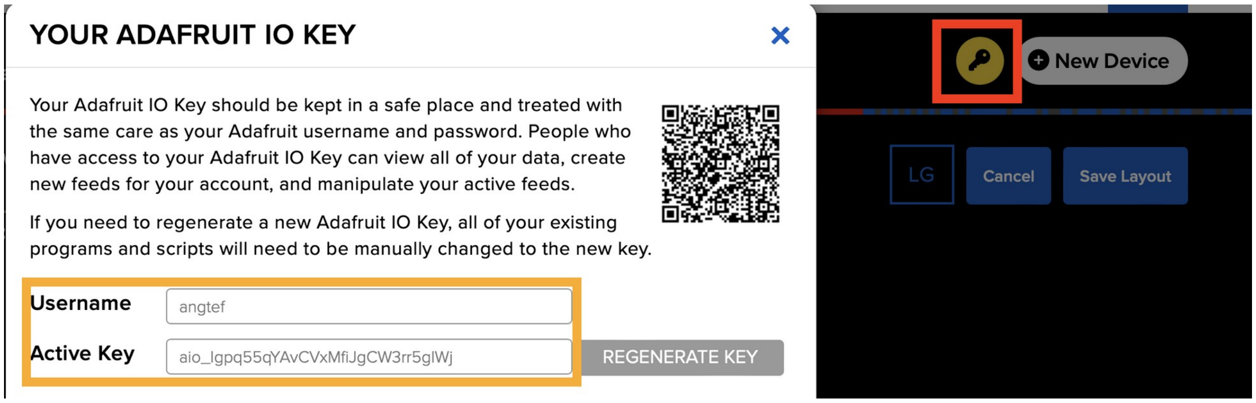
2. Add a piece of code that will allow you to connect to your WIFI. Here you need to fill in lines 15-18. First, enter the name of your WIFI and then the password.



smart_waste_bin.py backup.py

```
import machine
import utime
import network
from umqtt.simple import MQTTClient

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)

depth = 100

WIFI_SSID = "your_wifi_name"
WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_IO_USERNAME = "username"
ADAFRUIT_IO_KEY = "key"

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(WIFI_SSID, WIFI_PASSWORD)
    while not wlan.isconnected():
        print("Connecting to Wi-Fi...")
        utime.sleep(1)
    print("Connected to Wi-Fi:", wlan.ifconfig())

connect_wifi()

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.

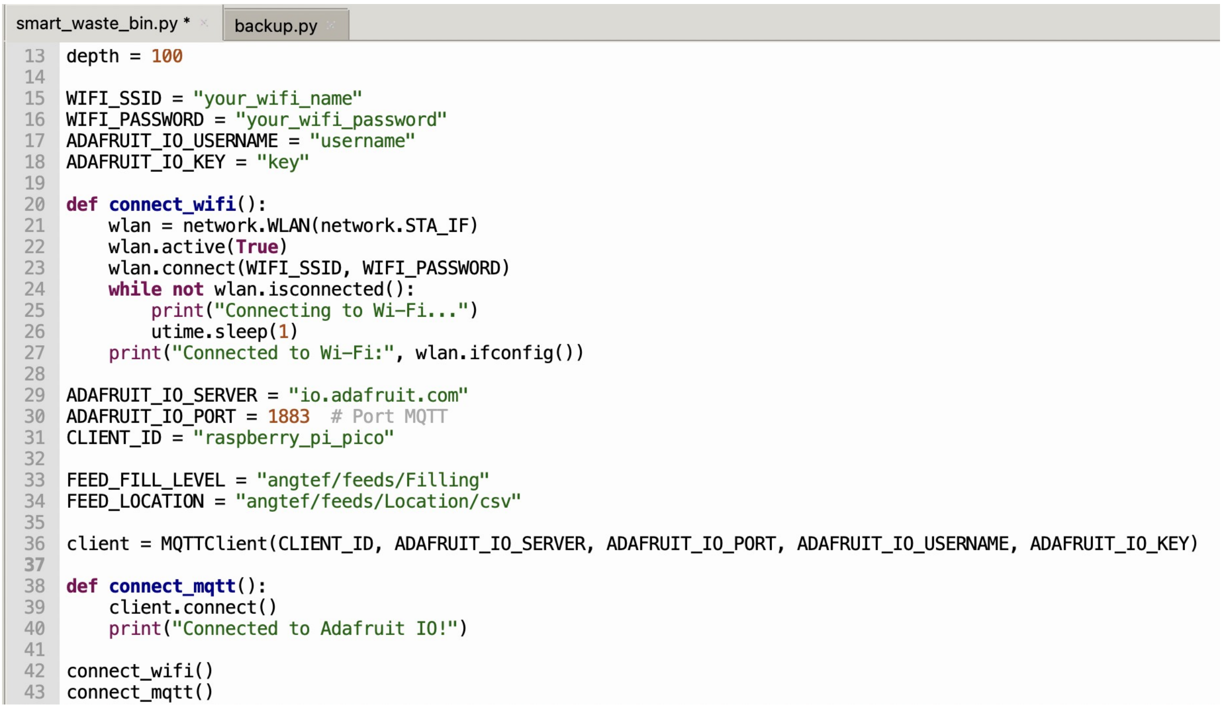
Username: angtef

Active Key: aio_Igpq55qYAvCVxMfiJgCW3rr5glWj

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. More details in "Technical guide about Raspberry Pi Pico and microPython". 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.

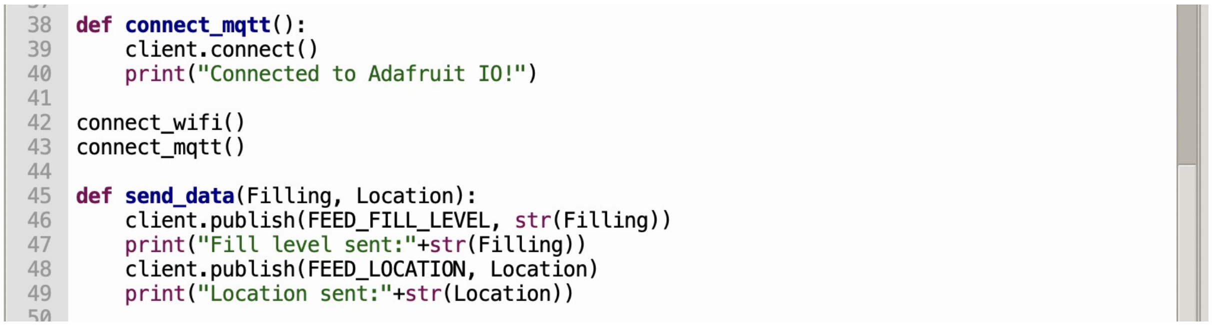
Project number: 2023-1-PL01-KA220-SCH-000154043

Co-funded by
the European Union



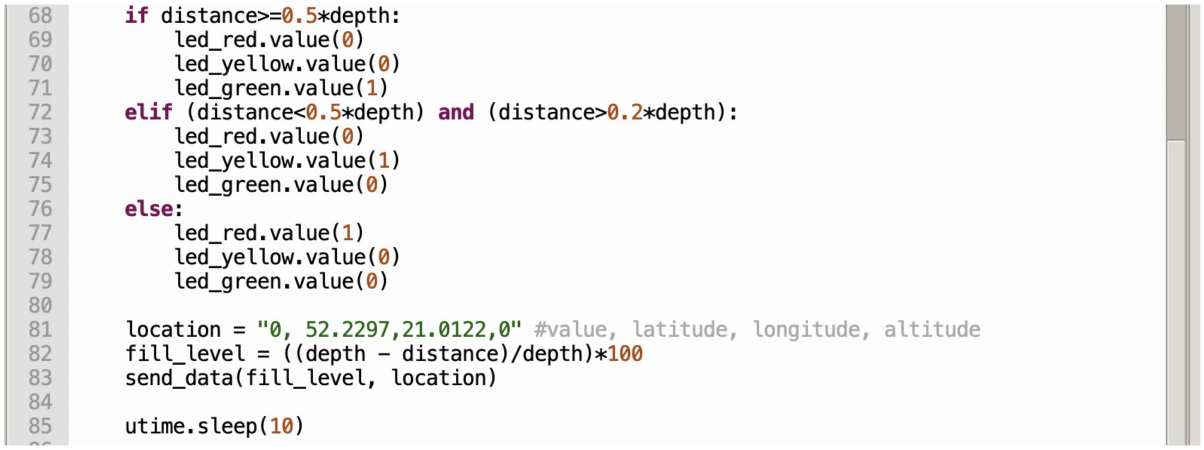
```python
depth = 100

WIFI_SSID = "your_wifi_name"
WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_IO_USERNAME = "username"
ADAFRUIT_IO_KEY = "key"

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(WIFI_SSID, WIFI_PASSWORD)
    while not wlan.isconnected():
        print("Connecting to Wi-Fi...")
        utime.sleep(1)
    print("Connected to Wi-Fi:", wlan.ifconfig())

ADAFRUIT_IO_SERVER = "io.adafruit.com"
ADAFRUIT_IO_PORT = 1883  # Port MQTT
CLIENT_ID = "raspberry_pi_pico"

FEED_FILL_LEVEL = "angtef/feeds/Filling"
FEED_LOCATION = "angtef/feeds/Location/csv"

client = MQTTClient(CLIENT_ID, ADAFRUIT_IO_SERVER, ADAFRUIT_IO_PORT, ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

def connect_mqtt():
    client.connect()
    print("Connected to Adafruit IO!")

connect_wifi()
connect_mqtt()
```

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



```python
def connect_mqtt():
    client.connect()
    print("Connected to Adafruit IO!")

connect_wifi()
connect_mqtt()

def send_data(Filling, Location):
    client.publish(FEED_FILL_LEVEL, str(Filling))
    print("Fill level sent:"+str(Filling))
    client.publish(FEED_LOCATION, Location)
    print("Location sent:"+str(Location))
```

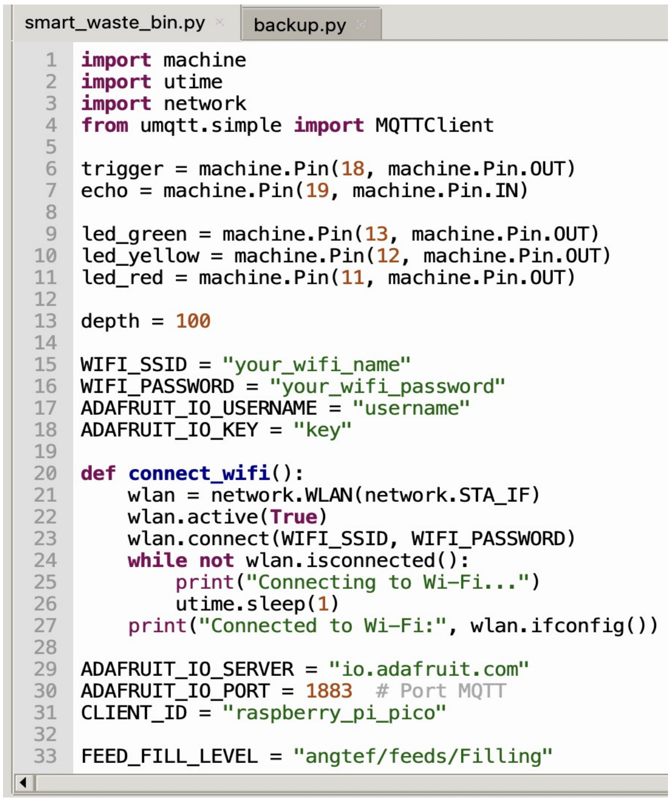
The last step is to pass the measured occupancy and location to the function:



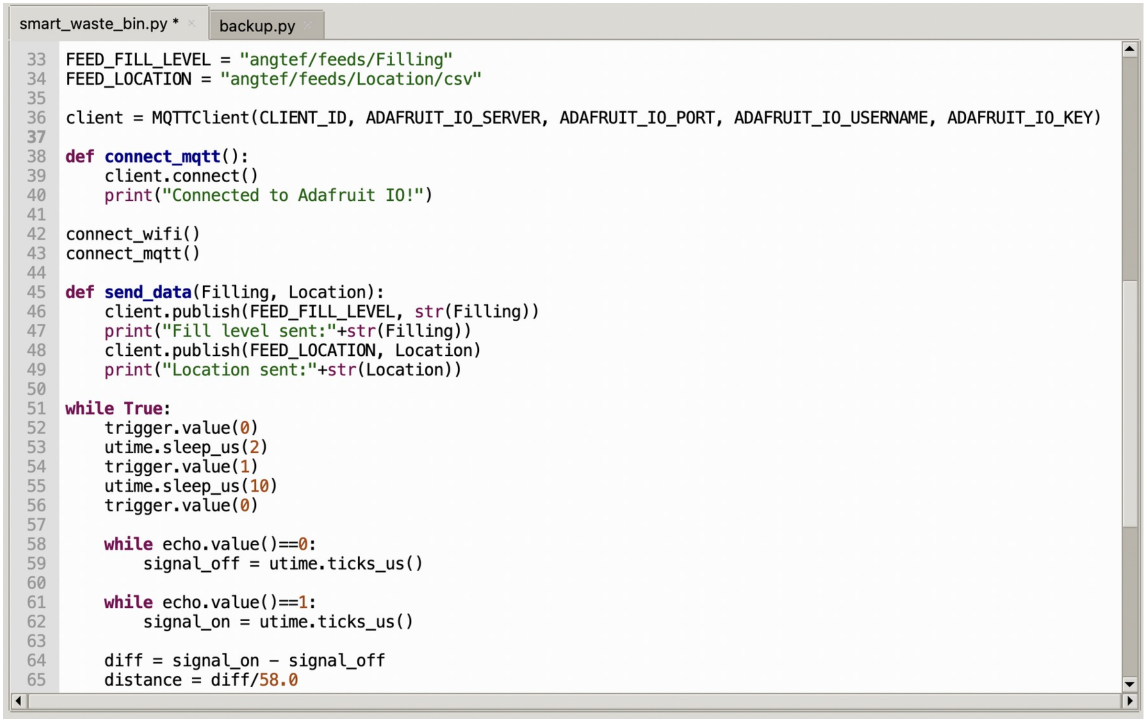
```python
    if distance>=0.5*depth:
        led_red.value(0)
        led_yellow.value(0)
        led_green.value(1)
    elif (distance<0.5*depth) and (distance>0.2*depth):
        led_red.value(0)
        led_yellow.value(1)
        led_green.value(0)
    else:
        led_red.value(1)
        led_yellow.value(0)
        led_green.value(0)

    location = "0, 52.2297,21.0122,0" #value, latitude, longitude, altitude
    fill_level = ((depth - distance)/depth)*100
    send_data(fill_level, location)

    utime.sleep(10)
```

The program is ready. Run it and go to the created dashboard on the Adafruit IO page.

The entire code looks like this:



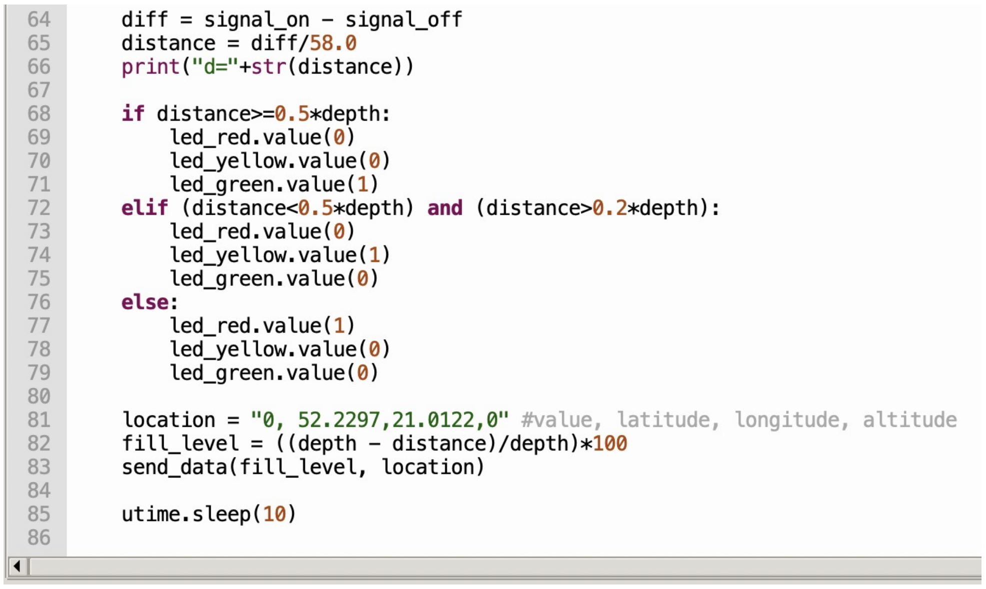
```python
import machine
import utime
import network
from umqtt.simple import MQTTClient

trigger = machine.Pin(18, machine.Pin.OUT)
echo = machine.Pin(19, machine.Pin.IN)

led_green = machine.Pin(13, machine.Pin.OUT)
led_yellow = machine.Pin(12, machine.Pin.OUT)
led_red = machine.Pin(11, machine.Pin.OUT)

depth = 100

WIFI_SSID = "your_wifi_name"
WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_IO_USERNAME = "username"
ADAFRUIT_IO_KEY = "key"

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(WIFI_SSID, WIFI_PASSWORD)
    while not wlan.isconnected():
        print("Connecting to Wi-Fi...")
        utime.sleep(1)
    print("Connected to Wi-Fi:", wlan.ifconfig())

ADAFRUIT_IO_SERVER = "io.adafruit.com"
ADAFRUIT_IO_PORT = 1883  # Port MQTT
CLIENT_ID = "raspberry_pi_pico"

FEED_FILL_LEVEL = "angtef/feeds/Filling"
```



```python
FEED_FILL_LEVEL = "angtef/feeds/Filling"
FEED_LOCATION = "angtef/feeds/Location/csv"

client = MQTTClient(CLIENT_ID, ADAFRUIT_IO_SERVER, ADAFRUIT_IO_PORT, ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

def connect_mqtt():
    client.connect()
    print("Connected to Adafruit IO!")

connect_wifi()
connect_mqtt()

def send_data(Filling, Location):
    client.publish(FEED_FILL_LEVEL, str(Filling))
    print("Fill level sent:"+str(Filling))
    client.publish(FEED_LOCATION, Location)
    print("Location sent:"+str(Location))

while True:
    trigger.value(0)
    utime.sleep_us(2)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)

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

    while echo.value()==1:
        signal_on = utime.ticks_us()

    diff = signal_on - signal_off
    distance = diff/58.0
```

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

Co-funded by
the European Union



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

    if distance>=0.5*depth:
        led_red.value(0)
        led_yellow.value(0)
        led_green.value(1)
    elif (distance<0.5*depth) and (distance>0.2*depth):
        led_red.value(0)
        led_yellow.value(1)
        led_green.value(0)
    else:
        led_red.value(1)
        led_yellow.value(0)
        led_green.value(0)

    location = "0, 52.2297,21.0122,0" #value, latitude, longitude, altitude
    fill_level = ((depth - distance)/depth)*100
    send_data(fill_level, location)

    utime.sleep(10)
```

3 Tips and recommendations

The project can be extended in two directions:

1. By adding GPS to measure the actual location of smart garbage containers. An example of reading data from GPS can be found in the "Technical guide about Raspberry Pi Pico and microPython".

2. By reading data from the cloud and writing code, e.g. in Python, to determine the optimal route for the garbage truck. Example codes that can be used for this are here: https://colab.research.google.com/drive/1aOq9jRh6c6fhaw1ahe0a1-yKVdMnO613?usp=sharing/