



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”

Inteligentne kosze na śmieci: jak usprawnić gospodarkę odpadami w inteligentnych miastach?

Przewodnik dla nauczycieli

Autorzy: Angelika Tefelska, Dariusz Tefelski, Adam Kowalczyk (ffotografie)

Instytucja: 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.



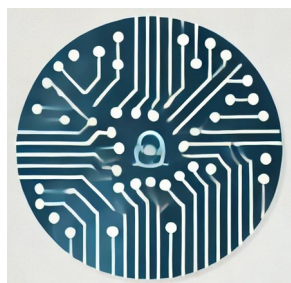
Table of contents

1 Wprowadzenie do projektu.....	3
1.1 Scenariusz i zakres projektu.....	3
1.2 Cele edukacyjne.....	4
1.3 Ścieżka kształcenia – etapy wdrażania.....	4
1.4 Wymagania wstępne.....	5
1.5 Sprzęt i oprogramowanie.....	5
1.6 Plan czasowy.....	6
2 Wykonanie projektu.....	7
2.1 Poziom 1.....	7
2.1.1 Tworzenie obwodu.....	7
2.1.2 Programowanie.....	7
2.1.3 Budowa kosza.....	16
2.2 Poziom 2.....	16
2.2.1 Programowanie.....	16
3 Porady i rozszerzenia.....	23

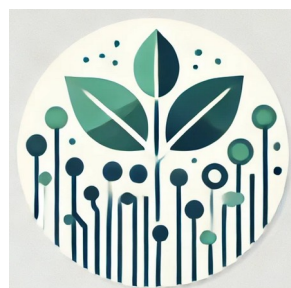
1 Wprowadzenie do projektu

1.1 Scenariusz i zakres projektu

Celem tego projektu jest zapoznanie studentów z technologią Internetu Rzeczy (IoT) w kontekście inteligentnych miast. Od kilku lat rozwijana jest koncepcja inteligentnych miast, w których rozwiązania IoT są i będą wykorzystywane do poprawy warunków życia i minimalizacji negatywnego wpływu miast na środowisko. Jednym z ważnych problemów współczesnych miast jest wywóz śmieci. Obecnie większość miast odbiera śmieci według ustalonego harmonogramu, przejeżdżając obok wszystkich posesji bez względu na poziom zapelnienia pojemników. Podobnie jest z pojemnikami, które stoją wzdłuż chodników i są używane przez przechodniów do wyrzucania śmieci. Taki system jest nieefektywny i powoduje znaczne zużycie paliwa. Projekt ma na celu pokazanie, jak stworzyć inteligentny kosz na śmieci przy użyciu płytki Raspberry Pi Pico i ultradźwiękowego czujnika odległości do pomiaru poziomu zapelnienia pojemnika. Poziom zapelnienia pojemnika będzie przesyłany do chmury, aby pomóc zoptymalizować trasę śmieciarki i tym samym zminimalizować ślad węglowy. Takie rozwiązania są już powoli wprowadzane w niektórych miastach, np. w Antwerpii (patrz zdjęcia poniżej).



Celem projektu jest rozwijanie kompetencji cyfrowych uczniów. Technologia IoT łączy wiele dziedzin: elektronikę (czujniki pomiarowe), programowanie (w tym przypadku język MicroPython - wersja Pythona dedykowana programowaniu mikrokontrolerów), zagadnienia sieciowe (wykorzystywanie chmur) i analizę danych. Podczas realizacji projektu uczniowie będą rozwijać wymienione powyżej kompetencje cyfrowe i technologiczne.



W ramach projektu uczniowie zapoznają się z problemem gospodarki odpadami w mieście, co jest niezwykle ważną kwestią, aby zminimalizować negatywny wpływ miast na środowisko poprzez zmniejszenie śladu węglowego (mniejsze zużycie paliwa przez śmieciarki). Ponadto zostanie omówiony temat recyklingu.



Projekt będzie realizowany w zespołach przez uczniów z różnych środowisk i krajów. Poprzez wspólną współpracę mamy nadzieję, że uczniowie zintegrują się pomimo różnic kulturowych lub barier językowych.

1.2 Cele edukacyjne

Dzięki temu projektowi uczniowie będą potrafili:

- Stworzyć prosty program w języku programowania MicroPython.
- Zbudować obwód elektroniczny przy użyciu płytki Raspberry Pi Pico.
- Zbudować i zaprogramować urządzenie, które może mierzyć wypełnienie pojemników na śmieci.

Ponadto:

- Nauczą się korzystać z czujników, takich jak ultradźwiękowy czujnik odległości.
- Zrozumieją, w jaki sposób urządzenia IoT mogą zbierać i przysyłać dane do chmury.
- Zrozumieją w jaki sposób dane mogą być monitorowane w czasie rzeczywistym.
- Będą potrafili wskazać rozwiązania mające na celu bardziej wydajne i ekologiczne zarządzanie odpadami w miastach.
- Będą potrafili wskazać zalety recyklingu i obecne trudności w przetwarzaniu odpadów.

1.3 Ścieżka kształcenia – etapy wdrażania

W ramach projektu uczniowie stworzą rozwiązania mające na celu poprawę sposobu zarządzania odpadami w miastach, aby zminimalizować ich ślad węglowy. W tym celu wykorzystają płytkę Raspberry Pi Pico i ultradźwiękowy czujnik odległości.

Poniżej przedstawiono kilka proponowanych etapów, które pozwolą sprawnie i skutecznie wdrożyć projekt inteligentnego kosza na odpady z uczniami:

1. **Tworzenie zespołów:** Podziel uczniów na zespoły dwu- lub trzyosobowe.
2. **Burza mózgów:** Zachęć każdy zespół do poszukiwania informacji na temat możliwych sposobów zarządzania odpadami w mieście, aby uczynić je bardziej inteligentnym (metoda zbiorczy

odpadów, optymalizacja tras, zalety i wady różnych rozwiązań, statystyki dotyczące recyklingu, zużycie paliwa przez śmieciarki itp.).

3. **Dyskusja:** Zachęć każdy zespół do podzielenia się swoimi wnioskami i pomysłami na temat tego, jak ulepszyć system gospodarowania odpadami, aby uczynić go efektywniejszym. Po dyskusji przedstaw konkretny cel projektu. (Uwaga: Zaleca się, aby konkretny cel projektu został przedstawiony po burzy mózgów, aby zachęcić uczniów do rozważenia projektu inteligentnego kosza na odpady w szerszym kontekście).
4. **Planowanie:** Zachęć każdy zespół do zastanowienia się nad sposobem skonstruowania urządzenia (jak będzie wykonany pojemnik, gdzie umieścić czujnik odległości, jakie dane powinny być przesyłane do chmury, kiedy dane powinny być przesyłane i jak często).
5. **Tworzenie:** Korzystając z arkuszy pracy uczniów, zachęć każdy zespół do stworzenia własnego inteligentnego kosza na śmieci. W zależności od umiejętności uczniów możesz rozważyć przydział ról.
6. **Testowanie - optymalizacja:** Po zakończeniu projektu zachęć uczniów do przetestowania ich inteligentnego kosza na śmieci. Na podstawie wyników testu możesz zachęcić każdy zespół do zoptymalizowania swojego projektu. Jeśli projekt nie był zbyt trudny dla uczniów, rozważ rozszerzenie, w którym opracujesz algorytm optymalizacji trasy (problem komiwojażera).
7. **Prezentacja – dzielenie się:** Zachęć swoich uczniów do zaprezentowania swoich projektów na forum i poproś ich o refleksję nad zdobytym doświadczeniem przy realizacji projektu. Zachęć wszystkie zespoły do rozważenia wpływu takich inteligentnych pojemników na odpady na funkcjonowanie miast i środowiska.

1.4 Wymagania wstępne

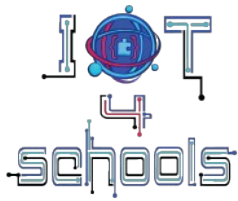
Studenci powinni znać podstawy programowania w dowolnym języku programowania. Jeśli nie mają takiego doświadczenia, nadal jest możliwe ukończenie projektu, ponieważ materiały są przygotowane w taki sposób, że każdy może ukończyć projekt.

1.5 Sprzęt i oprogramowanie

Sprzęt:

- Raspberry Pi Pico W
- Płytki stykowa
- Przewody Żeńsko-Męskie i Męsko-Męskie
- Diody LED: czerwona, żółta i zielona
- Rezystor 330 Ohm
- Zewnętrzne zasilanie: pojemnik na 2 baterie AAA lub power bank o niskim napięciu wyjściowym (do 5 V) - opcjonalnie
- Ultradźwiękowy czujnik odległości HC-SR04

Oprogramowanie:



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



Co-funded by
the European Union

- Edytor Thonny
- Chmura Adafruit IO

1.6 Plan czasowy

Szacuje się, że na ukończenie projektu potrzeba od 4 do 6 godzin

W szczególności szacuje się, że poszczególne etapy projektu zajmą:

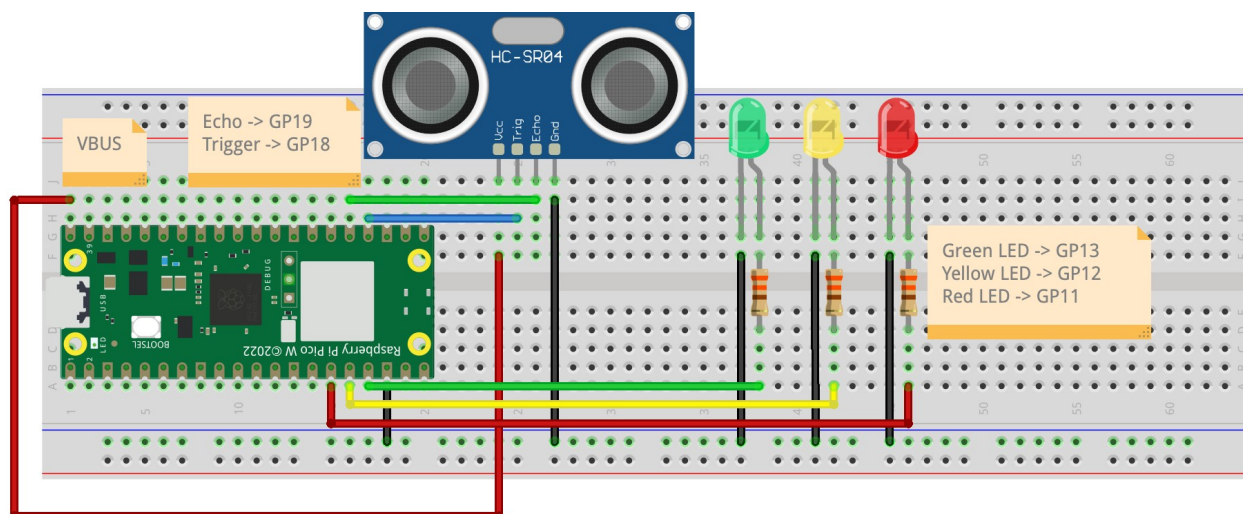
- 45-90 minut na wprowadzenie projektu (w tym burzę mózgów i dyskusję), planowanie i ćwiczenia wstępne
- 45 minut na ukończenie poziomu 1
- 45 minut na poziom 2
- 45-90 minut na rozszerzenia
- 30 minut na podsumowanie i dyskusję

2 Wykonanie projektu

2.1 Poziom 1

2.1.1 Tworzenie obwodu

Raspberry Pi Pico W należy połączyć z ultradźwiękowym czujnikiem odległości i trzema diodami LED za pomocą rezystorów 330Ω. Przykładowe połączenie pokazano na poniższym rysunku. Na początku można przetestować układ podłączony do komputera przez USB. Ostatecznie jednak do zasilania układu potrzebny będzie power bank lub baterie. Ze względów ekologicznych i ekonomicznych zalecamy power bank, który można naładować po zajęciach i ponownie wykorzystać.

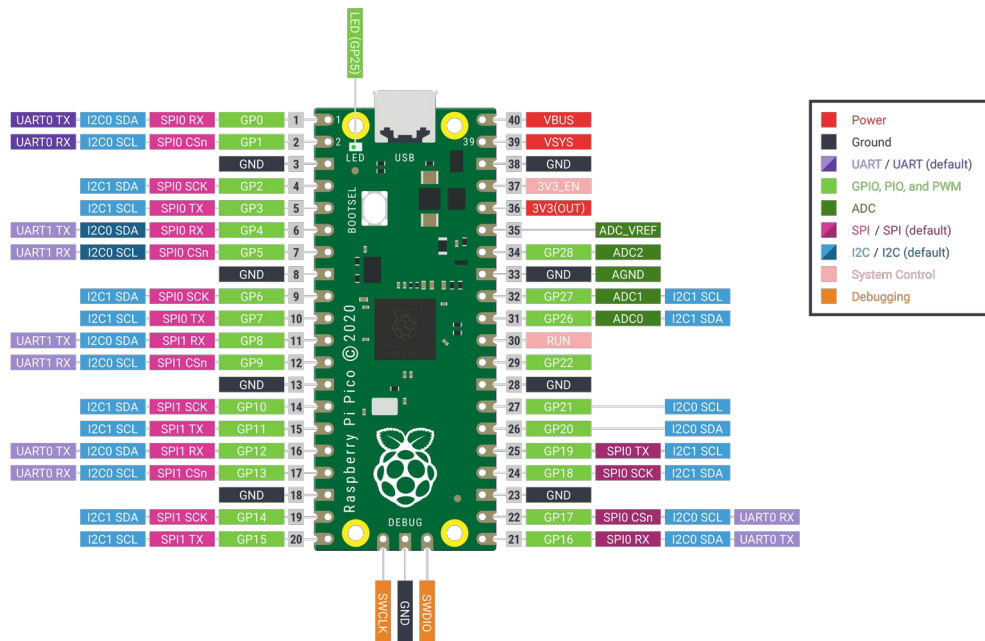


fritzing

Należy pamiętać, że na niektórych płytkach stykowych piny pod niebieską linią, której użyliśmy do GND, nie są połączone na całej długości płytki. Jeśli w niebieskim pasku jest przerwa, oznacza to, że tylko część płytki ma połączone piny. Należy zwrócić na to uwagę podczas podłączania płytki.

2.1.2 Programowanie

W tym projekcie będziemy używać pinów wejścia/wyjścia ogólnego przeznaczenia (GPIO), które służą do generowania lub odczytywania sygnałów cyfrowych, czyli takich, które mają tylko dwie możliwe wartości: 0 lub 1. GPIO to piny, które mogą być używane zarówno jako piny wejściowe, z których możemy odczytać informacje, np. o włączeniu przycisku, jak i jako piny wyjściowe, na których możemy generować sygnały cyfrowe, np. do mrugania diodą LED. Na poniższym rysunku przedstawiono 40 pinów z płytki Raspberry Pi Pico, które są ponumerowane od lewego górnego rogu. Piny pełniące funkcję GPIO są skrótowo oznaczane GPx (zielone prostokąty), gdzie x jest numerem porządkowym pinu GPIO, zaczynającym się od 0 do 28. Więcej informacji w poradniku: „Technical guide about Raspberry Pi Pico and MicroPython” dostępnym pod adresem: <https://www.iot.fizyka.pw.edu.pl/results/>.

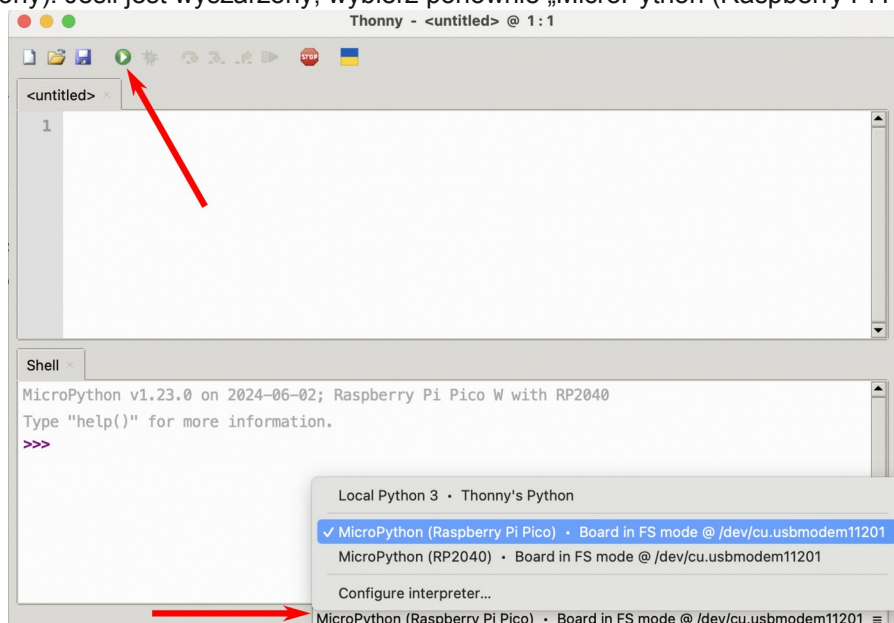


Źródło: <https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html>.

Aby uczniowie mogli jak najwięcej zrobić samodzielnie, sugerujemy, aby najpierw wykonali dwa ćwiczenia wstępne.

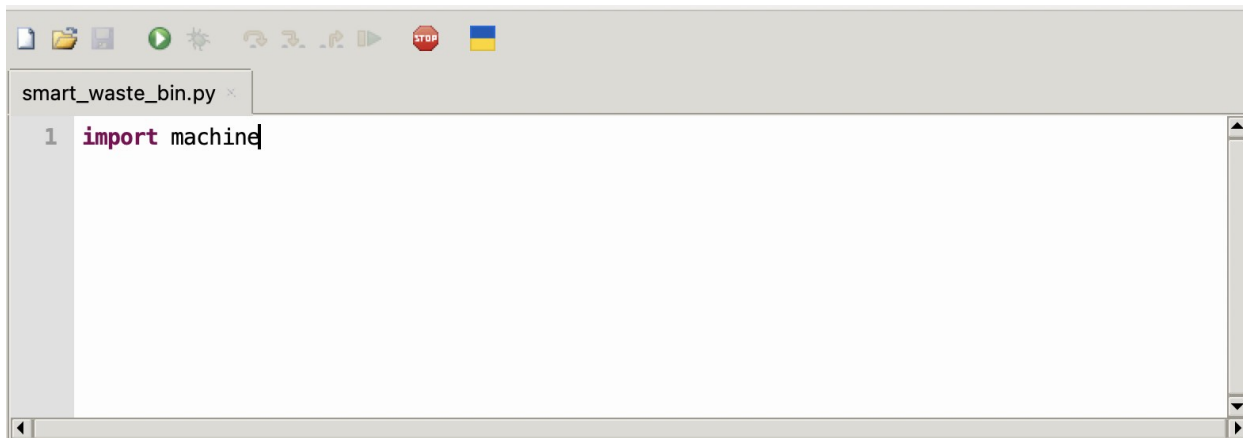
Ćwiczenie wstępne nr 1:

Pierwszym ćwiczeniem wstępnym jest stworzenie programu, który będzie symulował sygnalizację świetlną. Aby to zrobić, otwórz edytor Thonny, a następnie wybierz „MicroPython (Raspberry Pi Pico)”, jak pokazano na rysunku. Po poprawnym podłączeniu do płytki zielony przycisk Run powinien być aktywny (nie wyszarzony). Jeśli jest wyszarzony, wybierz ponownie „MicroPython (Raspberry Pi Pico)”.



Teraz napiszmy program dla sygnalizacji świetlnej. Aby to zrobić, wykonaj następujące kroki:

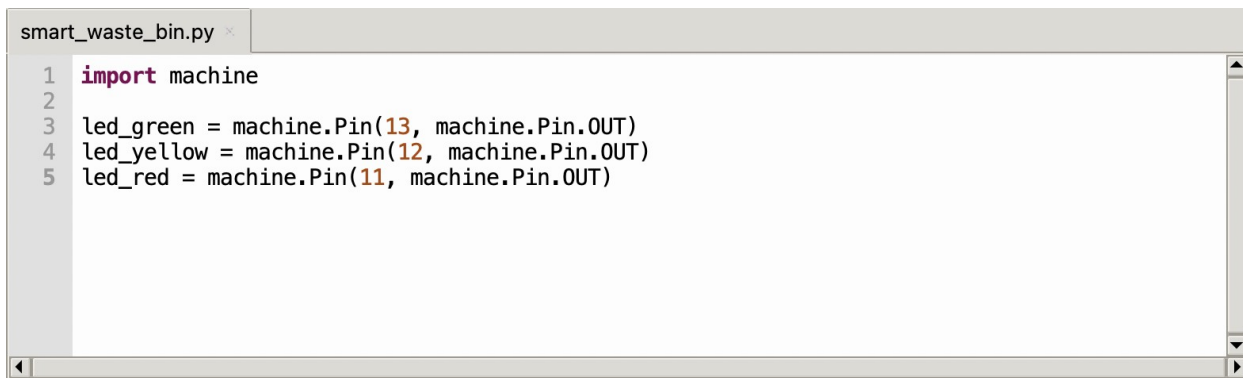
1. Zacznijmy od dołączenia biblioteki:



```
smart_waste_bin.py x
1 import machine
```

Biblioteka *machine* zawiera wszystkie niezbędne instrukcje do komunikacji z Raspberry Pi Pico. Polecenie *import* dodaje określoną bibliotekę do programu.

2. Następnie należy skonfigurować piny GP13, GP12 i GP11 tak, aby działały jako piny wyjściowe, ponieważ sterujemy diodami LED, wysyłając wartość 1 lub 0 do pinów GPIO. W tym celu używana jest funkcja *Pin* z biblioteki *machine*. Aby wywołać funkcję z biblioteki w języku Micropython, najpierw podajemy nazwę biblioteki, a po kropce nazwę funkcji z tej biblioteki, tj. *machine.Pin()*. Funkcja *Pin* przyjmuje dwa argumenty. Pierwszy to numer pinu GPIO. Drugi to określenie, czy pin GPIO będzie działał jako wejście (*machine.Pin.IN*) czy wyjście (*machine.Pin.OUT*). Stąd w tym przypadku wywołanie funkcji wyglądałoby następująco: *machine.Pin(13, machine.Pin.OUT)*. Obiekt zwrócony przez funkcję *Pin* został przypisany do utworzonej zmiennej *led_green*.



```
smart_waste_bin.py x
1 import machine
2
3 led_green = machine.Pin(13, machine.Pin.OUT)
4 led_yellow = machine.Pin(12, machine.Pin.OUT)
5 led_red = machine.Pin(11, machine.Pin.OUT)
```

3. W kolejnym kroku należy dodać pętlę nieskończoną, która będzie zawierała główną część programu:

```
smart_waste_bin.py x
1 import machine
2
3 led_green = machine.Pin(13, machine.Pin.OUT)
4 led_yellow = machine.Pin(12, machine.Pin.OUT)
5 led_red = machine.Pin(11, machine.Pin.OUT)
6
7 while True:
```

4. W następnym kroku zapalamy czerwoną diodę LED, wysyłając wartość 1 na pin GP11. Funkcja `value()` służy do ustawienia wartości na pinie, która przyjmuje wartość 0 lub 1 jako argument.

```
smart_waste_bin.py * x
1 import machine
2
3 led_green = machine.Pin(13, machine.Pin.OUT)
4 led_yellow = machine.Pin(12, machine.Pin.OUT)
5 led_red = machine.Pin(11, machine.Pin.OUT)
6
7 while True:
8     led_red.value(1)
9     |
```

5. Teraz program powinien odczekać 1s, abyśmy mogli zobaczyć efekt zapalenia diody. W tym celu wykorzystamy bibliotekę `utime`, która zawiera funkcje do opóźnień. Najpierw musimy dodać bibliotekę:

```
smart_waste_bin.py x
1 import machine
2 import utime
3
4 led_green = machine.Pin(13, machine.Pin.OUT)
5 led_yellow = machine.Pin(12, machine.Pin.OUT)
6 led_red = machine.Pin(11, machine.Pin.OUT)
7
8 while True:
9     led_red.value(1)
10    |
```

6. Funkcja `sleep` z biblioteki `utime` pozwala na opóźnienie programu o wybraną liczbę sekund. Ustawmy opóźnienie 1 s po zapaleniu się diody:

```
smart_waste_bin.py *
1 import machine
2 import utime
3
4 led_green = machine.Pin(13, machine.Pin.OUT)
5 led_yellow = machine.Pin(12, machine.Pin.OUT)
6 led_red = machine.Pin(11, machine.Pin.OUT)
7
8 while True:
9     led_red.value(1)
10    utime.sleep(1)
11    |
```

7. Teraz dodajmy resztę kodu, który spowoduje zapalenie pozostałych diod LED we właściwej kolejności.

```
smart_waste_bin.py
1 import machine
2 import utime
3
4 led_green = machine.Pin(13, machine.Pin.OUT)
5 led_yellow = machine.Pin(12, machine.Pin.OUT)
6 led_red = machine.Pin(11, machine.Pin.OUT)
7
8 while True:
9     led_red.value(1)
10    utime.sleep(1)
11
12    led_red.value(0)
13    led_yellow.value(1)
14    utime.sleep(1)
15
16    led_yellow.value(0)
17    led_green.value(1)
18    utime.sleep(1)
19
20    led_green.value(0)
21
```

Dzięki tej aktywności uczniowie dowiedzą się, jak korzystać z diod LED, które wykorzystamy do stworzenia inteligentnego kosza na śmieci.

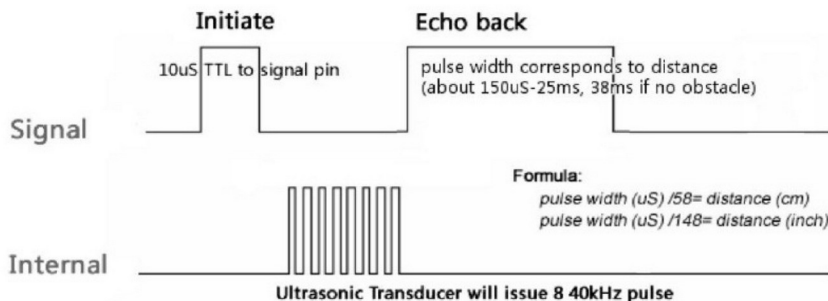
Ćwiczenie wstępne nr 2:

Teraz utworzymy program, który będzie odczytywał odległość za pomocą ultradźwiękowego czujnika odległości. Aby to zrobić, wykonaj następujące kroki:

1. Najpierw dodajmy niezbędne biblioteki, tj. machine i utime, i skonfigurujmy piny: GP18 (trigger) jako wyjście i GP19 (echo) jako wejście.

```
smart_waste_bin.py
1 import machine
2 import utime
3
4 trigger = machine.Pin(18, machine.Pin.OUT)
5 echo = machine.Pin(19, machine.Pin.IN)
6
7 |
```

2. Następnie w pętli głównej dodajmy fragment kodu, aby zmierzyć odległość od ultradźwiękowego czujnika odległości. Zgodnie z dokumentacją do czujnika HC-SR04 (patrz rysunek poniżej), najpierw należy ustawić niski sygnał na *trigger* przez krótki czas, np. 2µs. Następnie należy ustawić wysoki sygnał przez 10µs. Aby wygenerować opóźnienia w mikrosekundach, użyj funkcji: *utime.sleep_us()*. W następnym kroku ustaw niski sygnał na *trigger*.



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

```
smart_waste_bin.py
1 import machine
2 import utime
3
4 trigger = machine.Pin(18, machine.Pin.OUT)
5 echo = machine.Pin(19, machine.Pin.IN)
6
7 while True:
8     trigger.value(0)
9     utime.sleep_us(2)
10    trigger.value(1)
11    utime.sleep_us(10)
12    trigger.value(0)
```

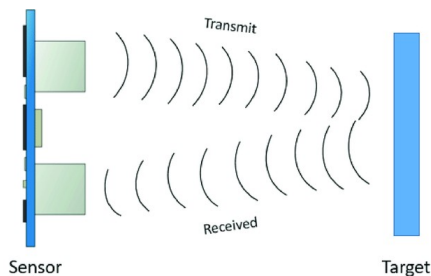
3. Teraz musimy zmierzyć, jak długo trwał wysoki sygnał na pinie echa, ponieważ czas trwania sygnału na pinie echa jest związany z odległością. Aby to zrobić, użyjemy funkcji *utime.ticks_us()*, która mierzy, ile czasu upłynęło w µs od uruchomienia programu. Najpierw utworzymy pętlę while, która będzie wykonywana tak długo, jak długo sygnał będzie niski. Wewnątrz umieścimy funkcję *tick_us()*. W ten sposób otrzymamy informacje o tym, kiedy sygnał był ostatnio niski.

```
smart_waste_bin.py
1 import machine
2 import utime
3
4 trigger = machine.Pin(18, machine.Pin.OUT)
5 echo = machine.Pin(19, machine.Pin.IN)
6
7 while True:
8     trigger.value(0)
9     utime.sleep_us(2)
10    trigger.value(1)
11    utime.sleep_us(10)
12    trigger.value(0)
13
14    while echo.value() == 0:
15        signal_off = utime.ticks_us()
```

4. Podobnie zmierzmy, kiedy sygnał na pinie echa był ostatnio wysoki:

```
smart_waste_bin.py
1 import machine
2 import utime
3
4 trigger = machine.Pin(18, machine.Pin.OUT)
5 echo = machine.Pin(19, machine.Pin.IN)
6
7 while True:
8     trigger.value(0)
9     utime.sleep_us(2)
10    trigger.value(1)
11    utime.sleep_us(10)
12    trigger.value(0)
13
14    while echo.value()==0:
15        signal_off = utime.ticks_us()
16
17    while echo.value()==1:
18        signal_on = utime.ticks_us()
```

Różnica między czasem ostatniego wystąpienia sygnału wysokiego i niskiego to czas trwania impulsu wysokiego na pinie echa. Jak przełożyć czas trwania impulsu na odległość? Spójrz na poniższy rysunek, pokazujący ideę pomiaru odległości za pomocą ultradźwiękowego czujnika odległości.



Źródło: https://www.researchgate.net/figure/principles_fig5_344385811

A-block-diagram-of-Ultrasonic-sensor-working-

Na początku emitowana jest fala dźwiękowa, która odbija się od obiektu i powraca do czujnika. Dlatego w zmierzonym czasie t fala pokonuje dwukrotność odległości między czujnikiem a obiektem i porusza się z prędkością około 340 m/s (prędkość dźwięku w powietrzu). Dlatego możemy zapisać równanie na szybkość:

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

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

Stąd otrzymany czas trwania impulsu należy podzielić przez 58, aby uzyskać odległość w centymetrach. Aby wyświetlić wartość w terminalu edytora Thonny, należy użyć funkcji `print`. Funkcja `str` konwertuje zmienną zmiennoprzecinkową na ciąg znaków, który jest niezbędny do wyświetlenia wartości w terminalu. Znak plus pozwala na połączenie własnego tekstu ze zmiennymi:

```
smart_waste_bin.py x
1 import machine
2 import utime
3
4 trigger = machine.Pin(18, machine.Pin.OUT)
5 echo = machine.Pin(19, machine.Pin.IN)
6
7 while True:
8     trigger.value(0)
9     utime.sleep_us(2)
10    trigger.value(1)
11    utime.sleep_us(10)
12    trigger.value(0)
13
14    while echo.value()==0:
15        signal_off = utime.ticks_us()
16
17    while echo.value()==1:
18        signal_on = utime.ticks_us()
19
20    diff = signal_on - signal_off
21    distance = diff/58.0
22    print("d="+str(distance))
```

```
Shell x
d=10.86207
d=11.10345
d=11.34483
d=11.31034
d=11.31034
```

Teraz uczniowie umieją korzystać z wszystkich elementów niezbędnych do zbudowania inteligentnego kosza na śmieci. Przejdźmy do projektu.

Projekt inteligentnego kosza na śmieci:

Aby stworzyć projekt inteligentnego kosza na śmieci, zaczniemy od modyfikacji programu z ćwiczenia wstępnego nr 2. W tym celu dodamy trzy diody LED:

```
smart_waste_bin.py * x
1 import machine
2 import utime
3
4 trigger = machine.Pin(18, machine.Pin.OUT)
5 echo = machine.Pin(19, machine.Pin.IN)
6
7 led_green = machine.Pin(13, machine.Pin.OUT)
8 led_yellow = machine.Pin(12, machine.Pin.OUT)
9 led_red = machine.Pin(11, machine.Pin.OUT)
10
11 while True:
12     trigger.value(0)
13     utime.sleep_us(2)
14     trigger.value(1)
15     utime.sleep_us(10)
16     trigger.value(0)
```

Następnie utworzymy zmienną *depth*, która będzie odzwierciedlać głębokość kosza na śmieci. Ustawmy tymczasowo wartość na 100 cm, a w późniejszym kroku dostosujemy ją do rzeczywistej głębokości kosza na śmieci, który utworzymy:

smart_waste_bin.py

```

1 import machine
2 import utime
3
4 trigger = machine.Pin(18, machine.Pin.OUT)
5 echo = machine.Pin(19, machine.Pin.IN)
6
7 led_green = machine.Pin(13, machine.Pin.OUT)
8 led_yellow = machine.Pin(12, machine.Pin.OUT)
9 led_red = machine.Pin(11, machine.Pin.OUT)
10
11 depth = 100
12
13 while True:
14     trigger.value(0)
15     utime.sleep_us(2)
16     trigger.value(1)
17     utime.sleep_us(10)
18     trigger.value(0)

```

Teraz, w zależności od tego, jak pełny jest kosz, zapalimy inną diodę LED. Gdy będzie poniżej 50% zapelnienia kosza, zaświeci się zielona dioda, gdy będzie powyżej 50% zapelnienia kosza, ale poniżej 80% zaświeci się żółta dioda, a gdy będzie powyżej 80% zapelnienia kosza, zaświeci się czerwona dioda. Pamiętaj, że 0% zapelnienia kosza dotyczy głębokości kosza, czyli w tym przykładzie 100 cm, a 50% zapelnienia będzie wtedy, gdy odczytamy odległość 50 cm przy czujniku, ponieważ czujnik będzie umieszczony w pokrywie kosza na samej górze.



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

```

13 while True:
14     trigger.value(0)
15     utime.sleep_us(2)
16     trigger.value(1)
17     utime.sleep_us(10)
18     trigger.value(0)
19
20     while echo.value()==0:
21         signal_off = utime.ticks_us()
22
23     while echo.value()==1:
24         signal_on = utime.ticks_us()
25
26     diff = signal_on - signal_off
27     distance = diff/58.0
28     print("d="+str(distance))
29
30     if distance>=0.5*depth:
31         led_red.value(0)
32         led_yellow.value(0)
33         led_green.value(1)
34     elif (distance<0.5*depth) and (distance>0.2*depth):
35         led_red.value(0)
36         led_yellow.value(1)
37         led_green.value(0)
38     else:
39         led_red.value(1)
40         led_yellow.value(0)
41         led_green.value(0)
42     utime.sleep(0.1)

```

2.1.3 Budowa kosza

Czas na stworzenie własnego inteligentnego pojemnika na śmieci. Aby to zrobić, użyj dowolnego pudełka po wysyłce/butach itp. Umieść płytkę stykową z czujnikiem ultradźwiękowym w górnej pokrywie. Pamiętaj, aby skierować czujnik w dół. Wytnij otwór na śmieci z przodu. Diody możesz umieścić albo na górze pojemnika, albo przy otworze na śmieci. Możesz podłączyć power bank do pojemnika na śmieci lub użyć zasilacza z USB komputera. Przykładowa implementacja jest pokazana na poniższych zdjęciach. Pamiętaj, że po skonstrowaniu inteligentnego pojemnika na śmieci powinieneś odczytać, jaką odległość (głębokość) czujnik zwraca, gdy pojemnik na śmieci jest pusty, i poprawić wartość w zmiennej *depth*. W przeciwnym razie inteligentny pojemnik na śmieci pokaże nieprawidłowe wypełnienie.



2.2 Poziom 2

W ramach poziomu 2 dodamy wysyłanie danych o wypełnieniu kosza na śmieci i o jego lokalizacji do chmury. W programie ręcznie wprowadzimy lokalizację do zmiennej, zakładając, że kosz na śmieci zawsze będzie należał do danego domu/bloku. Poniżej znajduje się opis programu krok po kroku.

2.2.1 Programowanie

Istnieje kilka różnych chmur, których możesz użyć. Jednak zalecamy skorzystać z Adafruit IO. Najpierw należy utworzyć bezpłatne konto na stronie <https://io.adafruit.com>. Następnie będziesz potrzebował/a wysłać dwie dane: wypełnienie kosza na śmieci i lokalizację kosza do chmury. Aby to zrobić, musisz utworzyć dwa kanały. Kanały to obiekty, które przechowują dane. Aby utworzyć kanał, przejdź do zakładki „Feeds” i wybierz przycisk „New Feed”.



Następnie pojawi się okno, w którym należy wpisać nazwę kanału, np. Filling czy Location.

Create a new Feed

Name

Maximum length: 128 characters. Used: 0

Description

Cancel Create



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



Co-funded by
the European Union

Utwórz dwa kanały. Gdy to zrobisz, powinieneś zobaczyć kanały, które utworzyłeś na swojej stronie, podobnie jak na zrzucie ekranu poniżej:

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

Default

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

Następnym krokiem jest utworzenie pulpitu. Aby to zrobić, wybierz zakładkę „Dashboards”, a następnie „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

Pojawi się okno, w którym należy wpisać nazwę wybranego pulpitu. Następnie po prawej stronie wybierz symbol ustawień i wybierz „Create New Block”.

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

adafruit Devices Feeds Dashboards Actions Power-Ups New Device

angtef / Dashboards / Test

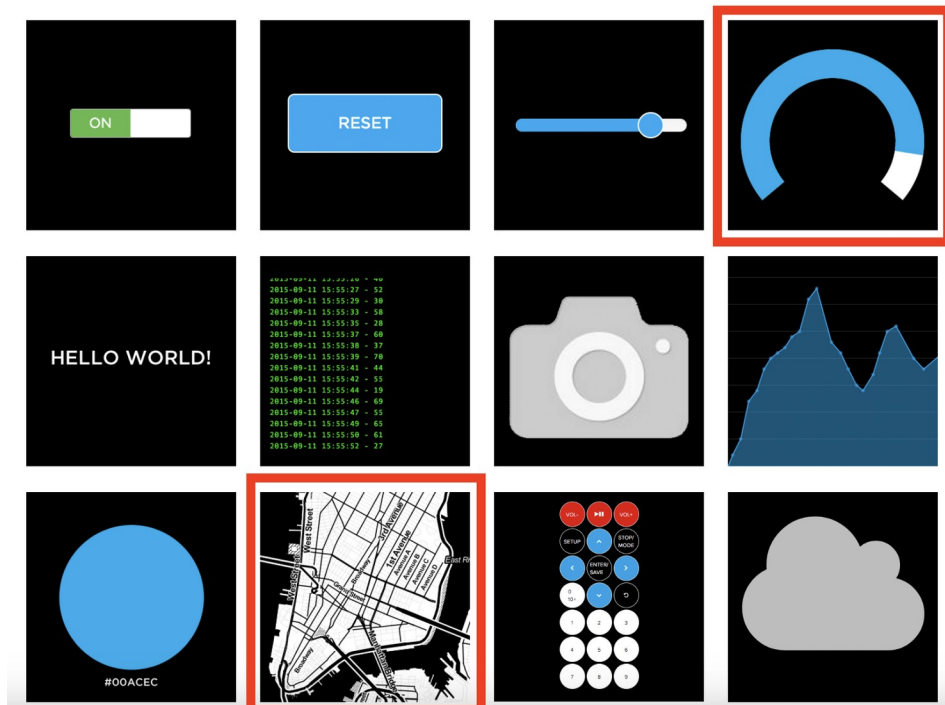
Settings

Adafruit IO ma różne bloki dostępne do wyświetlania danych (pola tekstowe, wskaźniki, wykresy, mapy itp.). W naszym przypadku wybierzmy wskaźnik i mapę:

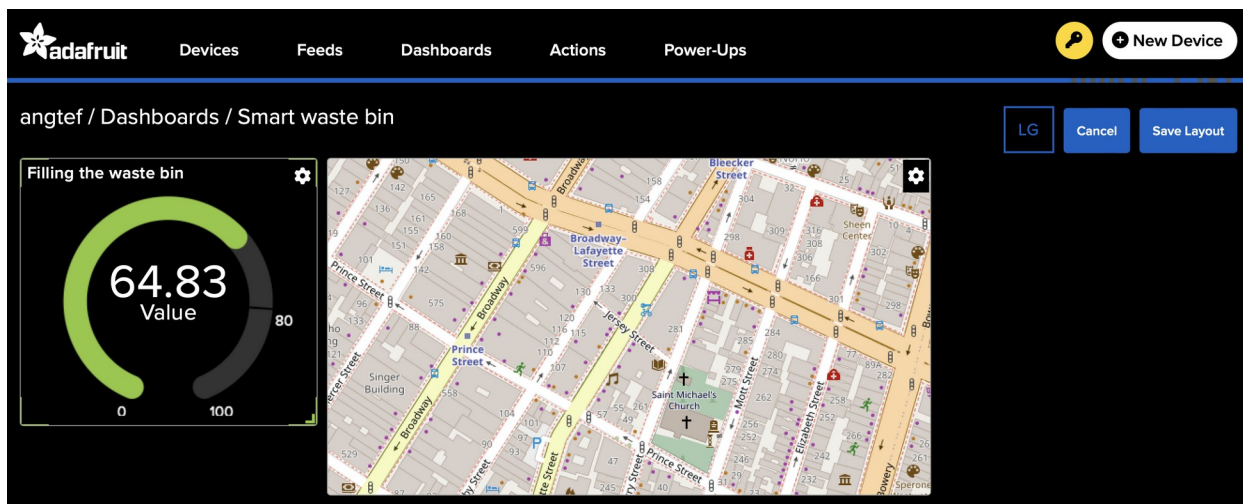
Create a new block



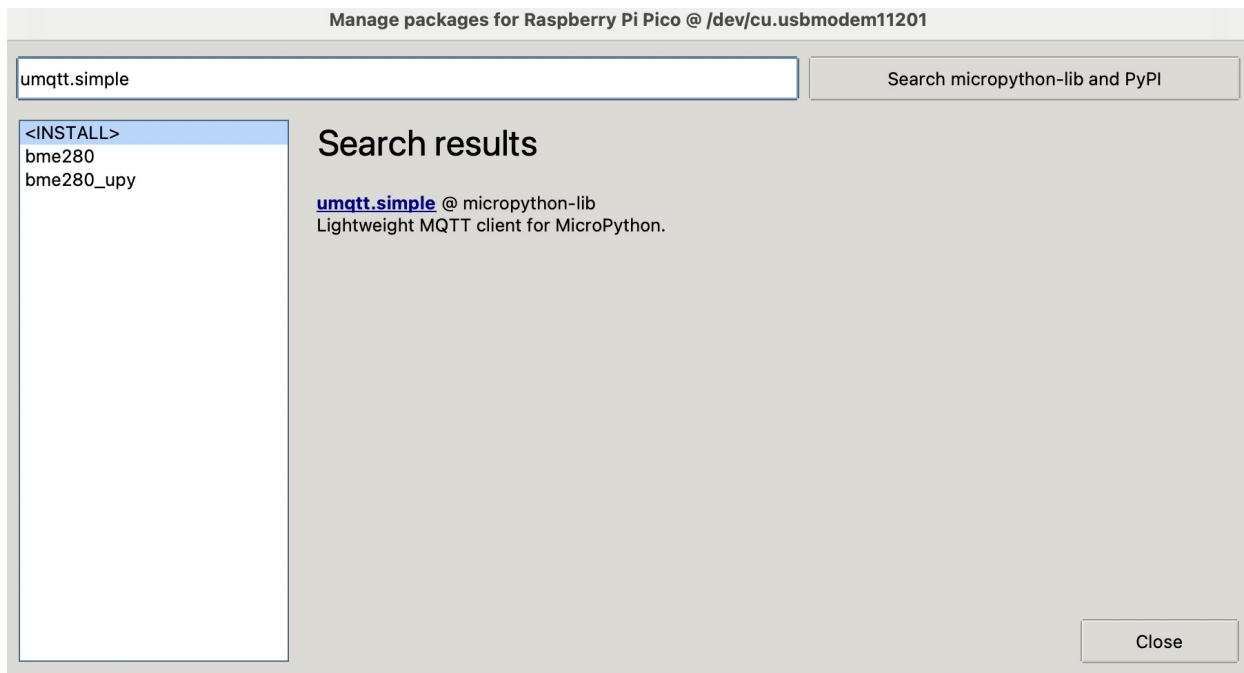
Click on the block you would like to add to your dashboard. You can always come back and switch the block type later if you change your mind.



Następnie pojawi się okno z pytaniem, do którego kanału chcemy podłączyć blok. Wybierz kanał definiujący wypełnienie kosza na śmieci w przypadku wskaźnika i kanał definiujący lokalizację w przypadku mapy. Następnie ponownie wybierz ikonę ustawień i wybierz „Edit Layout”. Ułóż bloki na ekranie tak, jak uważasz za stosowne. Możesz również powiększać i zmniejszać bloki. Przykład wyglądu pokazano na poniższym zrzucie ekranu:



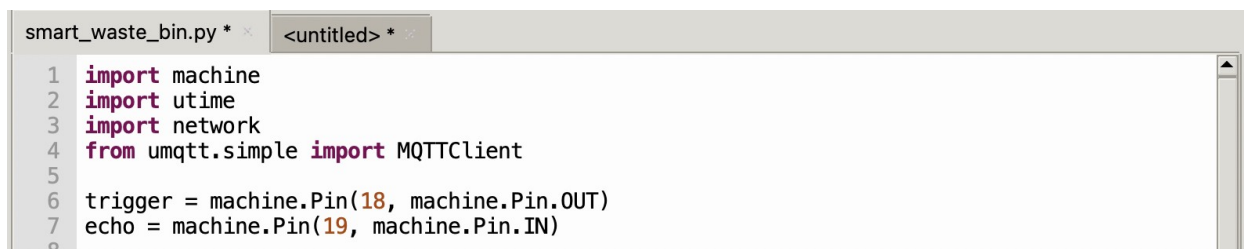
Przejdźmy teraz do edytora Thonny i zainstalujmy bibliotekę ***umqtt.simple***. Aby to zrobić, wybierz „Narzędzia”, a następnie „Zarządzaj pakietami”. Pojawi się okno, w którym należy wpisać nazwę biblioteki, którą chcesz zainstalować, w tym przypadku ***umqtt.simple***:



Po kliknięciu nazwy znalezionej biblioteki ***umqtt.simple*** otworzy się okno z danymi biblioteki. Będziesz mógł/a zainstalować bibliotekę, klikając przycisk Instaluj.

Teraz możemy wrócić do naszego kodu i dodać części niezbędne do wysłania danych do chmury. Aby to zrobić, wykonaj następujące kroki:

1. Oodaj niezbędne biblioteki:



2. Dodaj fragment kodu, który pozwoli Ci połączyć się z Twoją siecią WIFI. Tutaj należy wypełnić wiersze 15-18. Najpierw wpisz nazwę swojej sieci WIFI, a następnie hasło.

```
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:
```

Ostatnie dwa parametry to dane z platformy Adafruit IO. Wróć do witryny Adafruit i kliknij symbol klucza. Kiedy to zrobisz, pojawi się Twój login i klucz. Skopiuj te dane do programu do wierszy 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.

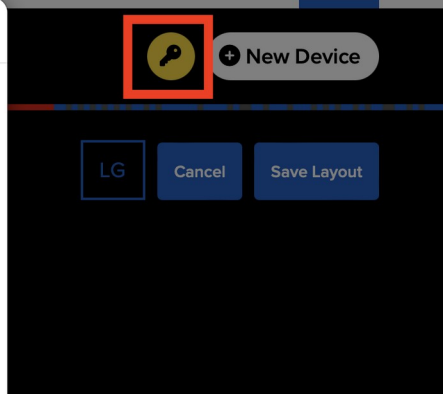
If you need to regenerate a new Adafruit IO Key, all of your existing programs and scripts will need to be manually changed to the new key.

Username

Active Key

[REGENERATE KEY](#)





3. Teraz użyjemy MQTT (Message Queuing Telemetry Transport), lekkiego protokołu komunikacyjnego opartego na modelu publish/subscribe. Został on zaprojektowany specjalnie do przesyłania danych w środowiskach o ograniczonych zasobach, takich jak urządzenia IoT (Internet of Things). Więcej szczegółów w „Technical guide about Raspberry Pi Pico and MicroPython”. Aby to zrobić, użyj poniższego kodu, zmieniając tylko nazwy kanałów w wierszach 33-34. Przykład używa kanałów o nazwach: „Filling” i „Location”. Zastąp je własnymi. Pamiętaj tylko, aby zostawić /csv w przypadku Location, ponieważ podczas korzystania z map musisz podać dane w formacie csv.

```
smart_waste_bin.py * 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. Teraz dodajmy funkcje wysyłające dane do chmury:

```
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
```

Ostatnim krokiem jest przekazanie zmierzonych danych o zapełnieniu i lokalizacji do funkcji:

```
68 if distance>=0.5*depth:
69     led_red.value(0)
70     led_yellow.value(0)
71     led_green.value(1)
72 elif (distance<0.5*depth) and (distance>0.2*depth):
73     led_red.value(0)
74     led_yellow.value(1)
75     led_green.value(0)
76 else:
77     led_red.value(1)
78     led_yellow.value(0)
79     led_green.value(0)
80
81 location = "0, 52.2297,21.0122,0" #value, latitude, longitude, altitude
82 fill_level = ((depth - distance)/depth)*100
83 send_data(fill_level, location)
84
85 utime.sleep(10)
```

Program jest gotowy. Uruchom go i przejdź do utworzonego pulpitu na stronie Adafruit IO.

Cały kod wygląda następująco:

```
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 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"
```

```
smart_waste_bin.py * backup.py
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()
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
51 while True:
52     trigger.value(0)
53     utime.sleep_us(2)
54     trigger.value(1)
55     utime.sleep_us(10)
56     trigger.value(0)
57
58     while echo.value() == 0:
59         signal_off = utime.ticks_us()
60
61     while echo.value() == 1:
62         signal_on = utime.ticks_us()
63
64     diff = signal_on - signal_off
65     distance = diff / 58.0
```

```

64     diff = signal_on - signal_off
65     distance = diff/58.0
66     print("d="+str(distance))
67
68     if distance>=0.5*depth:
69         led_red.value(0)
70         led_yellow.value(0)
71         led_green.value(1)
72     elif (distance<0.5*depth) and (distance>0.2*depth):
73         led_red.value(0)
74         led_yellow.value(1)
75         led_green.value(0)
76     else:
77         led_red.value(1)
78         led_yellow.value(0)
79         led_green.value(0)
80
81     location = "0, 52.2297,21.0122,0" #value, latitude, longitude, altitude
82     fill_level = ((depth - distance)/depth)*100
83     send_data(fill_level, location)
84
85     utime.sleep(10)
86

```

3 Porady i rozszerzenia

Projekt można rozszerzyć w dwóch kierunkach:

1. Poprzez dodanie GPS do pomiaru rzeczywistej lokalizacji inteligentnych pojemników na śmieci. Przykład odczytu danych z GPS można znaleźć w „Technical guide about Raspberry Pi Pico and MicroPython”.
2. Poprzez odczyt danych z chmury i napisanie kodu, np. w Pythonie, w celu określenia optymalnej trasy dla śmieciarki. Przykładowe kody, których można użyć w tym celu, znajdują się tutaj:

<https://colab.research.google.com/drive/1aOq9jRh6c6fhaw1ahe0a1-yKVdMnO613?usp=sharing/>