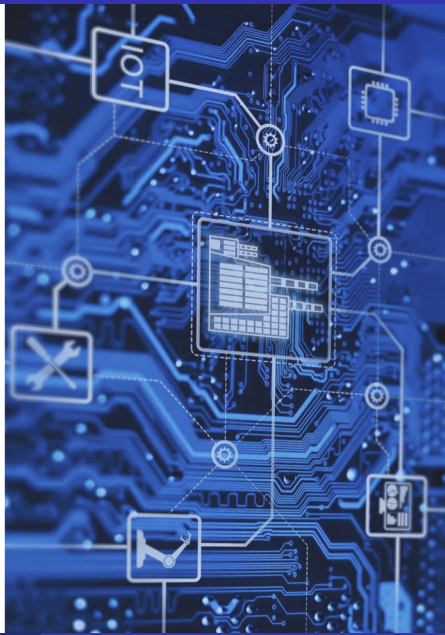
PROJECT NUMBER: 2023-1-PL01-KA220-SCH-000154043

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?"

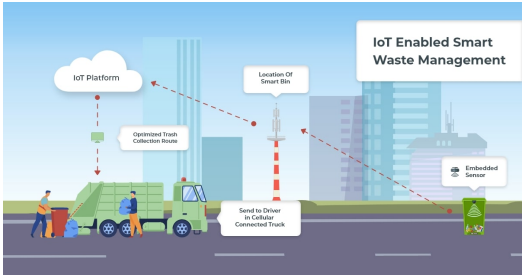


Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Cel projektu

- Celem tego projektu jest zapoznanie uczniów z technologią Internetu Rzeczy (IoT) w kontekście inteligentnych miast, w szczególności zagadnieniem zarządzania odpadami.
- Jednym z ważnych problemów współczesnych miast jest wywóz śmieci. Obecnie większość miast odbiera śmieci według ustalonego harmonogramu, przejeżdżając obok wszystkich posesji niezależnie od poziomu zapełnienia pojemników.
- Taki system jest nieefektywny i powoduje znaczne zużycie paliwa.
- Projekt koncentruje się na zaprojektowaniu inteligentnych pojemników na śmieci, które będą informować o ich zapełnieniu.



Źródło: https://www.excellentwebworld.com

Cele edukacyjne

Dzięki temu projektowi uczniowie będą potrafili:

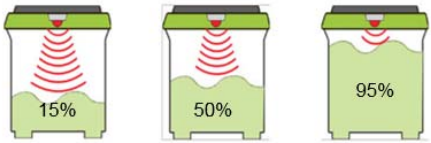
- Stworzyć prosty program oparty na języku programowania MicroPython.
- Zbudować obwód elektroniczny przy użyciu płytki Raspberry Pi Pico oraz korzystać z podstawowych czujników.
- Zbudować i zaprogramować urządzenie, które może mierzyć zapełnienie pojemników na śmieci.
- Korzystać z platformy chmurowej *Adafruit IO* i przesyłać do niej dane z czujników.

Ponadto uczniowie:

- Zrozumieją, w jaki sposób urządzenia IoT mogą zbierać i przesyłać dane do chmury.
- Będą w stanie wskazać rozwiązania mające na celu usprawnienie systemu odbioru odpadów tak by był on bardziej ekologiczny i efektywny.
- Wskazać zalety recyklingu i obecne trudności w przetwarzaniu odpadów.
- Pogłębią umiejętności współpracy i przełamią bariery międzykulturowe.

Zawartość projektu

- Dwa ćwiczenia wstępne:
  - z wykorzystaniem diod LED,
  - z wykorzystaniem ultradźwiękowego czujnika odległości.
- Poziom 1: budowa inteligentnego kosza na śmieci, który mierzy stopień zapełnienia kosza na podstawie ultradźwiękowego czujnika odległości i wskazuje poziom napełnienia za pomocą 3 diod LED (zielonej, żółtej i czerwonej)
- Poziom 2: modyfikacja kosza na śmieci z poziomu 1 poprzez dodanie wysyłania danych do chmury (poziom zapełnienia, lokalizacja wprowadzona jako stała w programie)
- Zalecane rozszerzenia:
  - dodanie modułu GPS w celu faktycznego określenia położenia kosza - rozwiązanie w poradniku technicznym dla nauczycieli;
  - analizowanie danych z różnych pojemników na odpady i określanie optymalnej trasy dla śmieciarki, która dociera tylko do pełnych pojemników - rozwiązanie w *Collab Research* dodane jako załącznik.



Source of image: https://www.researchgate.net

Ścieżka edukacyjna

1. **Formowanie grup:** Podziel uczniów na grupy 2-3 osobowe.
2. **Burza mózgów:** czas na poszukiwanie informacji przez uczniów na temat możliwych sposobów usprawnienia systemu odbioru odpadów z posesji.
3. **Dyskusja:** dzielenie się wnioskami i pomysłami na temat tego, jak udoskonalić system gospodarowania odpadami, aby uczynić go bardziej efektywnym.
4. **Planowanie:** czas na przygotowanie planu stworzenia inteligentnego kosza na śmieci.
5. **Tworzenie:** przygotowanie inteligentnego kosza na śmieci bazując na arkuszu pracy.
6. **Testowanie - optymalizacja**
7. **Prezentacja**: przedstawienie projektów na forum klasy.

Niezbędne wyposażenie

- Sprzęt:
  - Raspberry Pi Pico W board,
  - płytka stykowa,
  - przewody męsko-żeńskie i męsko-męskie,
  - 3 diody LED (czerwona, zielona i żółta),
  - 3 rezystory 330Ω,
  - Ultradźwiękowy czujnik odległości HC-SR04
- Oprogramowanie:
  - edytor Thonny,
  - Chmura Adafruit IO.

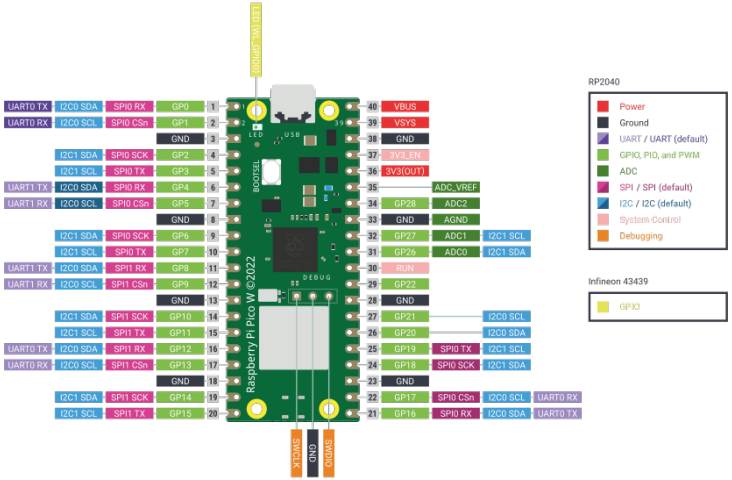
Przewidywany czas realizacji projektu w klasie:

- 45-90 minut - wprowadzenie projektu wraz z burzą mózgów, dyskusją, planowaniem i ćwiczeniami rozgrzewającymi,
- 45 minut - poziom 1,
- 45 minut - poziom 2,
- 45-90 minut - rozszerzenia projektu,
- 30 minut - dyskusja i prezentacja projektów.

Ćwiczenia wstępne

Ćwiczenie wstępne nr 1 - układ płytki

- *General Purporse Input/Output* to piny wejścia/wyjścia ogólnego przeznaczenia (GPIO), które służą do generowania lub odczytywania sygnałów cyfrowych, czyli takich, które mają tylko dwie możliwe wartości: 0 lub 1.
- Piny pełniące funkcję GPIO są skrótowo oznaczane GPx (zielone prostokąty), gdzie x jest numerem porządkowym pinu GPIO, zaczynającym się od 0 do 28.

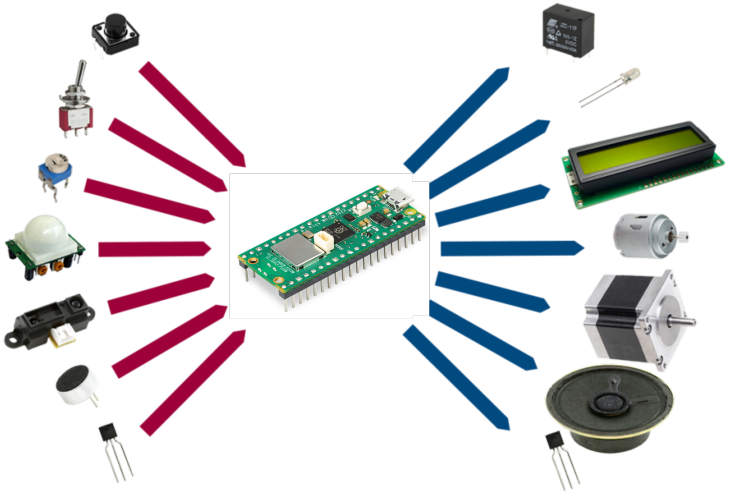


Źródło: https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html

Ćwiczenie wstępne nr 1 - konfiguracja pinów

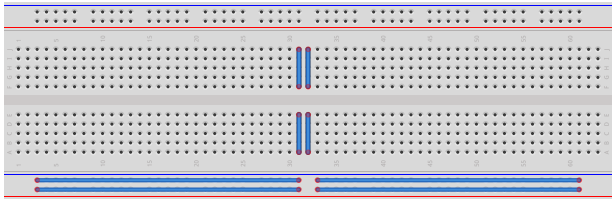
Piny, do których podłączane są elementy, mogą być:

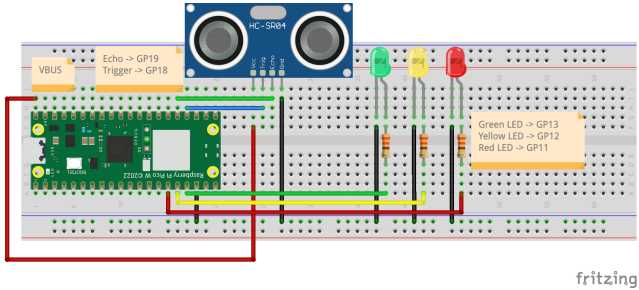
- wejściowe (*input*),
- wyjściowe (*output*).



Źródło oryginalnego obrazka: https://blog.codebender.cc/2014/03/07/lesson-1-inputs-and-outputs/

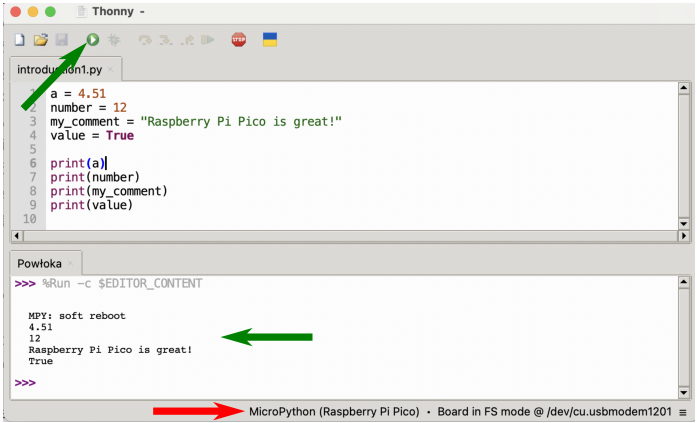
Ćwiczenie wstępne nr 1 - podłączenie





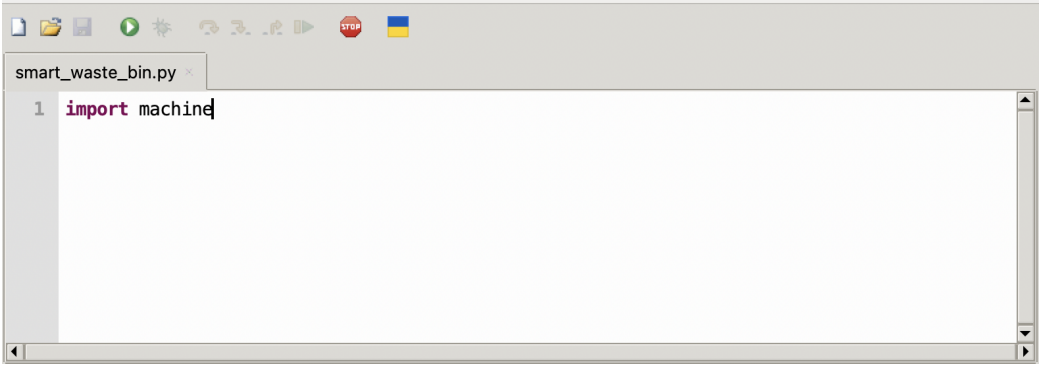
Ćwiczenie wstępne nr 1 - edytor Thonny

- Celem pierwszego ćwiczenia jest stworzenie programu, który będzie symulował sygnalizację świetlną.
- Aby to zrobić, otwórz edytor Thonny, a następnie wybierz "MicroPython (Raspberry Pi Pico)" (czerwona strzałka)
- Po poprawnym podłączeniu do płytki zielony przycisk Run powinien być aktywny (nie wyszarzony). Jeśli jest wyszarzony, wybierz ponownie "MicroPython (Raspberry Pi Pico)".



Ćwiczenie wstępne nr 1

1. Zacznijmy od dodania biblioteki, zawierającej funkcje niezbędne do komunikacji z płytką Raspberry Pi Pico. Komenda *import* dodaje wskazaną bibliotekę:



Ćwiczenie wstępne nr 1

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 funkcje 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*).

smart_waste_bin.py

```
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)
```

Dygresja: Ogólna struktura każdego programu

- Pętla *while* to taki rodzaj pętli, który będzie powtarzać wszystko co w niej umieścimy tak długo aż będzie spełniony warunek. Pętlę while tworzy się następująco:

```
while warunek:
        polecenia
```

- W przypadku programowania mikrokontrolerów tworzy się nieskończoną pętlę, czyli pętlę która będzie wykonywać się cały czas aż do wyłączenia zasilania płytki. Stąd jako warunek podana jest wartość *True*.

```
#Dodanie niezbędnych bibliotek, które zawierają przydatne funkcje
import nazwa_biblioteki

#Utworzenie zmiennych
#Wykonanie poleceń, które mają wykonać się tylko raz

#Nieskończona pętla
while True:
    #Polecenia, które mają się ciągle wykonywać
```

Ćwiczenie wstępne nr 1

3. Następnie tworzymy nieskończoną pętlę *while*:

smart_waste_bin.py

```
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:
```

Ćwiczenie wstępne nr 1

4. Następnie 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 *

```
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
```

Ćwiczenie wstępne nr 1

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

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
```

Ćwiczenie wstępne nr 1

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
```

Ćwiczenie wstępne nr 1

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
```

Czas na przetestowanie układu!

Ćwiczenie wstępne nr 2

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

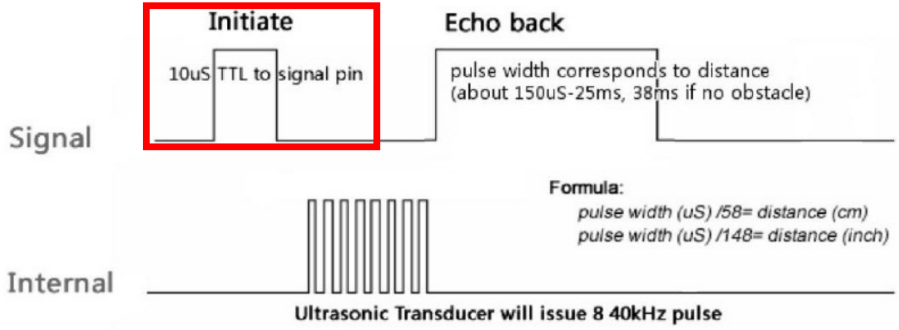
1. Najpierw dodajmy niezbędne biblioteki (*machine* i *utime*), i skonfigurujmy piny: GP18 (trigger) jako wyjście i GP19 (echo) jako wejście.

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
```

Ćwiczenie wstępne nr 2

2. Następnie w pętli głównej dodajmy fragment kodu, aby zmierzyć odległość korzystając z ultradźwiękowego czujnika odległości.
   - Zgodnie z dokumentacją czujnika HC-SR04, najpierw należy ustawić niski sygnał na *trigger* na krótki czas, np. $2\mu s$. Następnie należy ustawić wysoki sygnał na $10\mu s$.



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

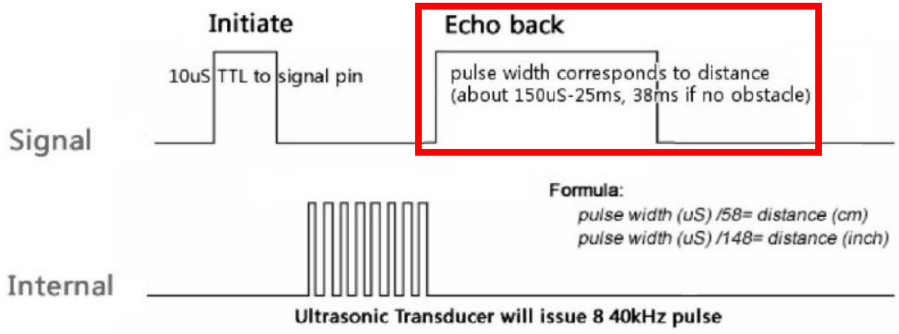
Ćwiczenie wstępne nr 2

- Aby wygenerować opóźnienia w mikrosekundach, użyj funkcji: *utime.sleep_us()*.

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)
```

Ćwiczenie wstępne nr 2



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

Ćwiczenie wstępne nr 2

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** i zmierzy w jakiej chwili ostatni raz był stan niski na pinie echo.

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()
```

Ćwiczenie wstępne nr 2

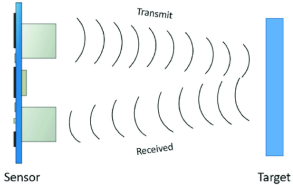
4. Teraz utworzymy pętlę while, która będzie wykonywana tak długo, jak długo sygnał będzie **wysoki** i zmierzy w jakiej chwili ostatni raz był stan wysoki na pinie echo.

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()
```

Ćwiczenie wstępne nr 2

- Różnica między czasem ostatniego wystąpienia sygnału wysokiego i niskiego to czas trwania impulsu wysokiego na pinie echa.
- Jak przełożyć czas trwania impulsu na odległość?



Źródło: https://www.researchgate.net/figure/A-block-diagram-of-Ultrasonic-sensor-working-principles_fig5_344385811

- 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):

$$v = \frac{2d}{t} \qquad \rightarrow \qquad d = v \cdot \frac{t}{2} = 0.034 \frac{cm}{\mu s} \cdot \frac{t}{2} \approx \frac{t}{58} cm \tag{1}$$

Ćwiczenie wstępne nr 2

5. 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

```
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))
```

Czas na przetestowanie układu!

Poziom 1

Cel: Budowa inteligentnego kosza na śmieci, który mierzy stopień zapełnienia kosza na podstawie ultradźwiękowego czujnika odległości i wskazuje poziom napełnienia za pomocą 3 diod LED (zielonej, żółtej i czerwonej).

1. Zaczniemy od modyfikacji programu z ćwiczenia wstępnego nr 2. W tym celu dodamy 3 diody:

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 while True:
12     trigger.value(0)
13     utime.sleep_us(2)
14     trigger.value(1)
15     utime.sleep_us(10)
16     trigger.value(0)
```

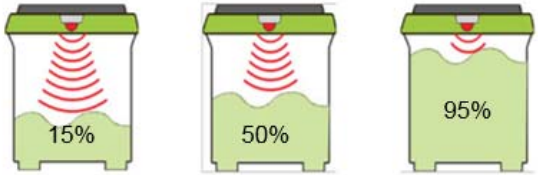
2. Następnie utwórzmy zmienną *depth*, która będzie odzwierciedlać głębokość kosza na śmieci. Ustawmy tymczasowo wartość na 100cm, a w późniejszym kroku dostosujemy ją do rzeczywistej głębokości kosza na śmieci, który utworzymy:

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)
```

Poziom 1

- W zależności od tego, jak pełny jest kosz, zapalimy inną diodę LED:
  - (0,50)% zapełnienia - dioda zielona,
  - (50,80)% zapełnienia - dioda żółta,
  - $>$ 80% zapełnienia - dioda czerwona.
- Pamiętaj, że 0% zapełnienia odpowiada wartości głębokości kosza, czyli w tym przykładzie 100 cm, a 50% zapełnienia będzie wtedy, gdy odczytamy odległość 50 cm itd.



Source of image: https://www.researchgate.net

Poziom 1

```
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)
```

Poziom 1

- Teraz czas na stworzenie własnego inteligentnego pojemnika na śmieci.
- Aby to zrobić, użyj dowolnego pudełka do wysyłki/na buty 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.
- Pamiętaj, że po skonstruowaniu pojemnika na śmieci powinieneś odczytać, jaką odległość (głębokość) czujnik zwraca, gdy pojemnik na śmieci jest pusty, i **poprawić wartość w zmiennej *depth***!

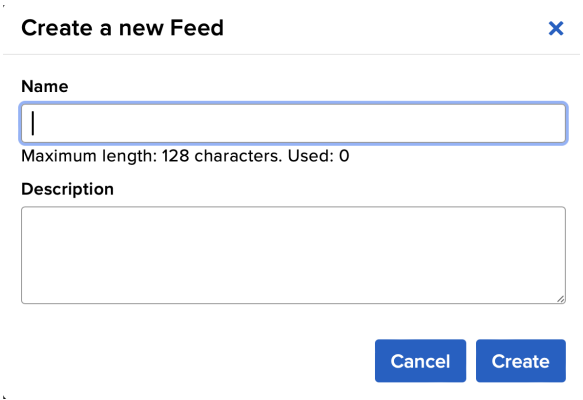
Poziom 2

Cel

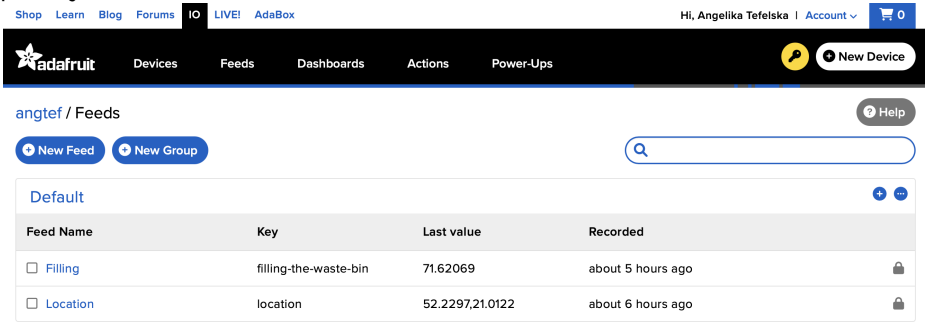
Inteligentny kosz na śmieci przesyła dane o zapełnieniu i swojej lokalizacji do chmury.

1. Najpierw należy założyć bezpłatne konto na `https://io.adafruit.com`.
2. Następnie będziemy potrzebowali wysłać dwie wartości: zapełnienie kosza na śmieci i lokalizację kosza do chmury. Aby to zrobić, musisz utworzyć dwa *feed'y* - obiekty, które przechowują dane:

2. Następnie pojawi się okienko z wyborem nazwy *feed'a*. Wpisz wybraną nazwę np. *Filling* czy *Location*:

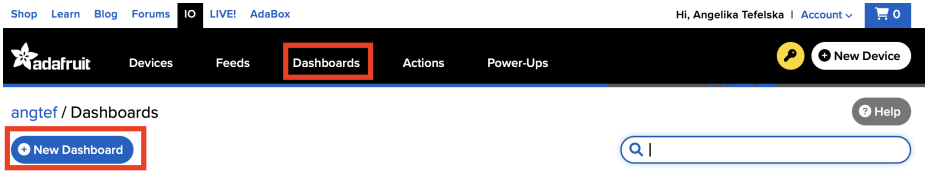


3. Gdy utworzysz dwa *feed'y* to powinieneś je zobaczyć na swojej stronie, podobnie jak na zrzucie ekranu poniżej:

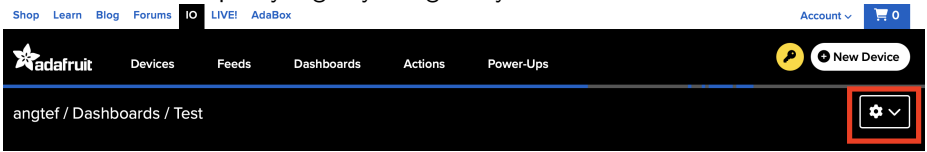


Poziom 2

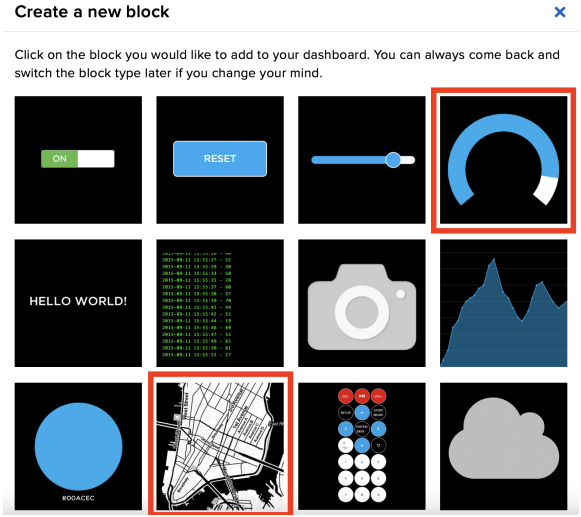
4. Następnie należy utworzyć pulpit:



5. Po nadaniu nazwy pulpitowi. Należy dodać bloki (elementy wyświetlające dane). W tym celu kliknij na znak ustawień w prawym górnym rogu i wybierz *Create New Block*.

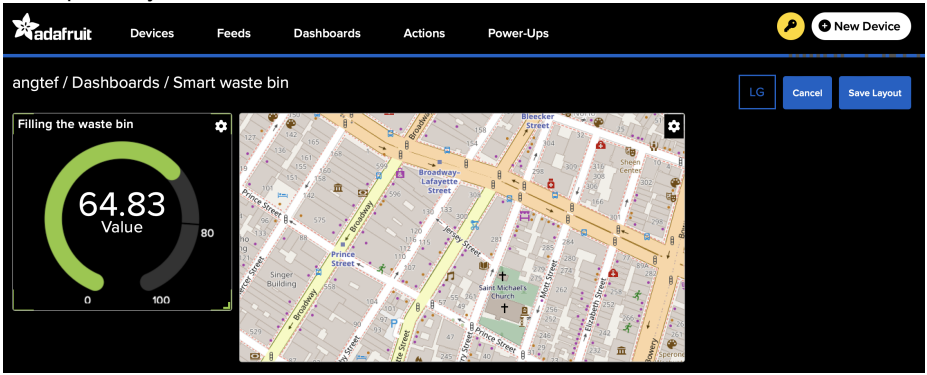


Możliwe bloki

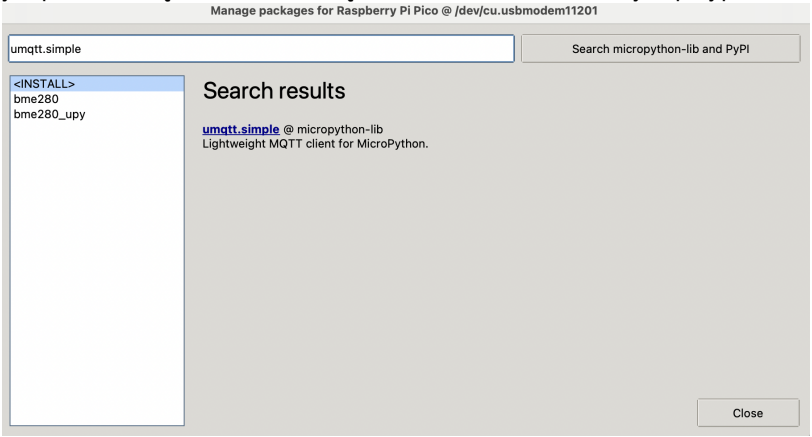


Poziom 2

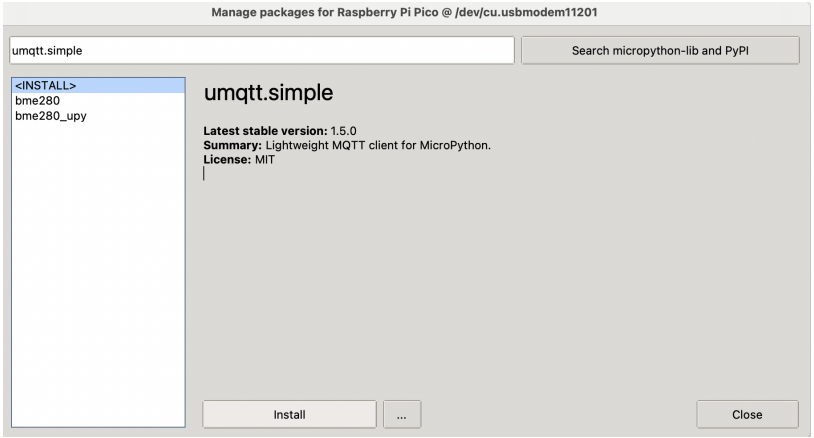
5. Następnie ponownie wybierz ikonę ustawień i wybierz „Edytuj układ". Ułóż bloki na ekranie tak, jak uważasz za stosowne. Możesz również powiększać i zmniejszać bloki. Przykład wyglądu pokazano na poniższym zrzucie ekranu:



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



Poziom 2


Manage packages for Raspberry Pi Pico @ /dev/cu.usbmodem11201
umqtt.simple
Search micropython-lib and PyPI
<INSTALL>
bme280
bme280_upy
umqtt.simple
Latest stable version: 1.5.0
Summary: Lightweight MQTT client for MicroPython.
License: MIT
Install
...
Close

7. Teraz możemy wrócić do naszego kodu i dodać części niezbędne do wysyłania danych do chmury. Zacznijmy od dodania niezbędnych bibliotek:

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)
```

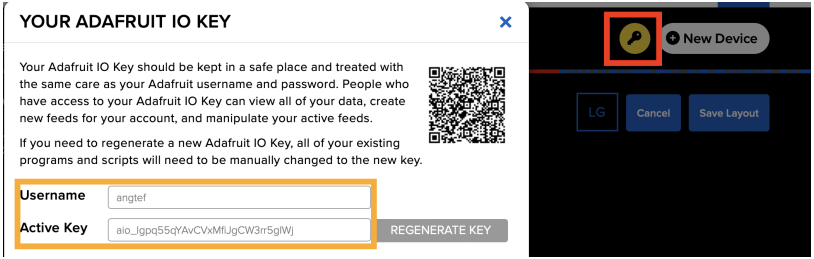
Poziom 2

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

smart_waste_bin.py | backup.py

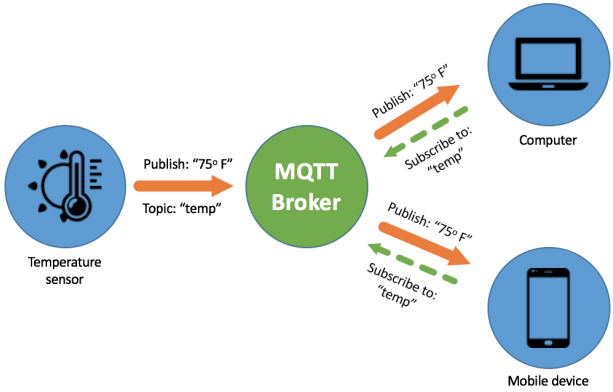
```
1  import machine
2  import utime
3  import network
4  from umqtt.simple import MQTTClient
5
6  trigger = machine.Pin(18, machine.Pin.OUT)
7  echo = machine.Pin(19, machine.Pin.IN)
8
9  led_green = machine.Pin(13, machine.Pin.OUT)
10 led_yellow = machine.Pin(12, machine.Pin.OUT)
11 led_red = machine.Pin(11, 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:
```

9. Ostatnie dwa parametry (linia 17-18) to login i klucz do konta Adafruit IO. Wróć na stronę Adafruit IO i kliknij na symbol klucza. Następnie pojawi się okno z niezbędnymi danymi do skopiowania:



MQTT

- Protokół MQTT opiera się na modelu publikuj/subskrybuj.
- Klient "A" publikuje informacje w danym temacie, a każdy inny klient, który zasubskrybował ten temat, odczyta dane opublikowane przez klienta "A".



Źródło: https://www.akcp.com/blog/scaling-mqtt-network-for-better-operational-output/

Poziom 2

- Teraz użyjemy protokołu MQTT. Aby to zrobić, użyj kodu obok, zmieniając tylko nazwy kanałów w wierszach 33-34. Pamiętaj tylko, aby zostawić /csv w przypadku *Location*!

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 = "raspberry_pi_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,
37
38  def connect_mqtt():
39      client.connect()
40      print("Connected to Adafruit IO!")
41
42  connect_wifi()
43  connect_mqtt()
```

Poziom 2

1. Teraz dodajmy funkcję do wysyłania danych 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
```

Poziom 2

2. Ostatni krok to wywołanie funkcji do wysyłania danych do chmury i przekazanie do niej zmierzonego zapełnienia i lokalizacji:

```
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)
```

Czas na przetestowanie układu!

Ankieta

Proszę o wypełnienie ankiet:

1. Obowiązkowa (anonimowa): `https://forms.gle/5rUfHGSQV4WK5Dx97`
2. Opcjonalna (nie anonimowa dt. współpracy): `https://forms.gle/uYhZwqKkh4HXHWRW7`

Dziękuję za uwagę