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

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

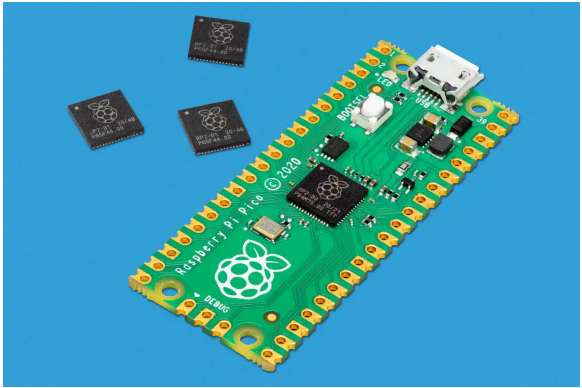
Wprowadzenie do Raspberry Pi Pico i języka MicroPython

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.

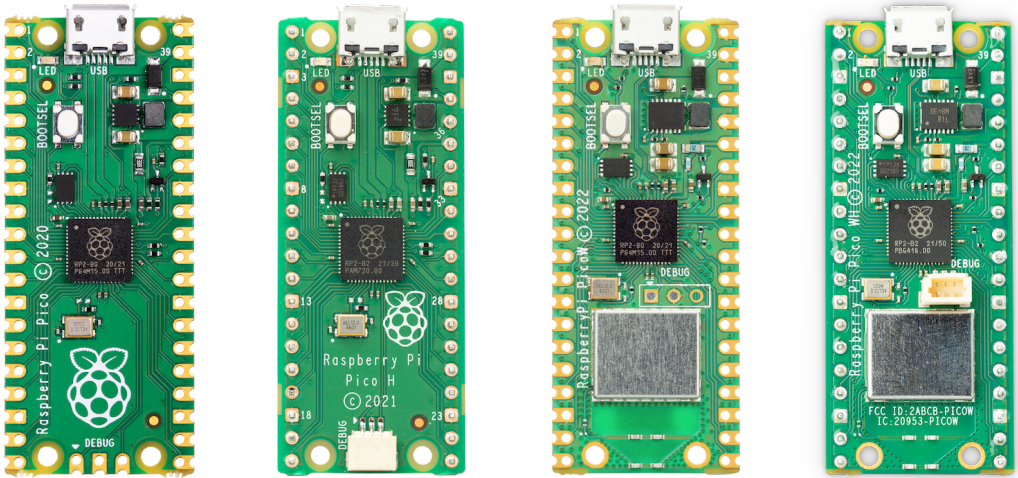
Raspberry Pi Pico

- W 2021 r. Raspberry Pi Foundation ogłosiło premierę płytki **Raspberry Pi Pico**.
- Płytka ta wyposażona jest w autorski układ **RP2040** - dwurdzeniowy mikrokontroler **ARM Cortex M0+** pracujący z częstotliwością 133 MHz.
- Płytkę można programować używając:
  - C/C++ SDK
  - MicroPython



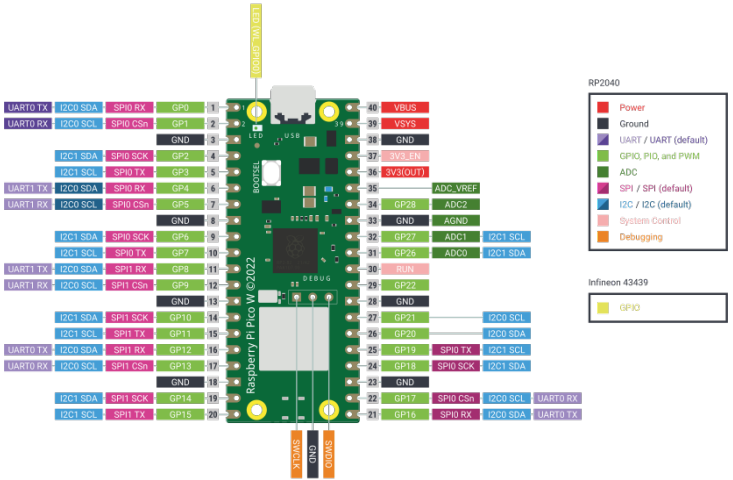
Źródło: https://www.raspberrypi.com/products/raspberry-pi-pico/

Wersje Raspberry Pi Pico



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

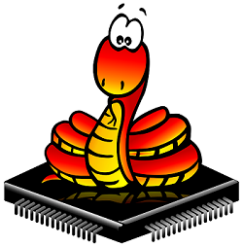
Układ płytki Raspberry Pi Pico W



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

Wstęp do języka MicroPython

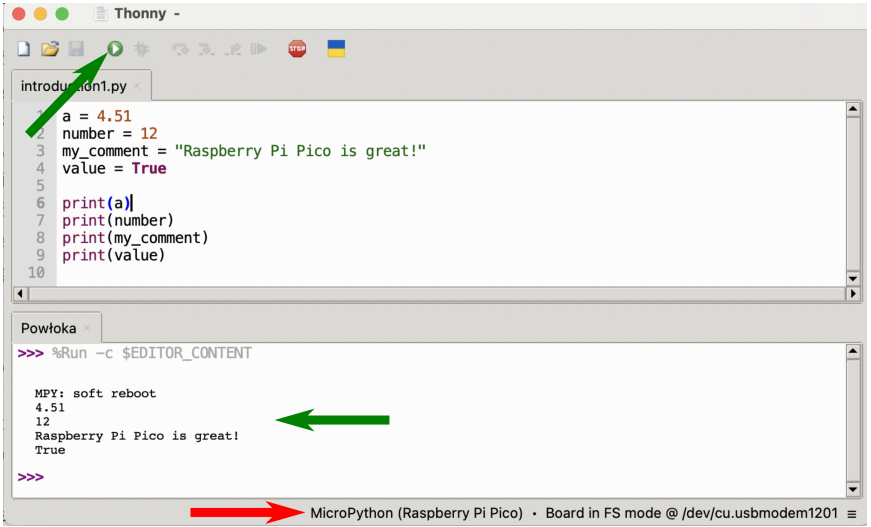
- **MicroPython** to zmodyfikowana wersja języka Python, dedykowana do programowania mikrokontrolerów.
- Najważniejsze elementy języka Python:
  - **Zmienne** - umożliwiają przechowywanie wartości pod wybraną nazwą zmiennej np. zamiast w kodzie używać kilka razy wartości 3.14 można utworzyć zmienną *pi=3.14*.
  - Rodzaje zmiennych:



Źródło: https://randomnerdtutorials.com/getting-started-raspberry-pi-pico-w/

	Typ zmiennej	Przykład definicji
Liczby całkowite	int	LED=2
Liczby zmiennoprzecinkowe	float	pi=3.14
Wartości logiczne	boolean	status=True
Ciąg znaków	string	powitanie="Witaj"

Wstęp do języka MicroPython



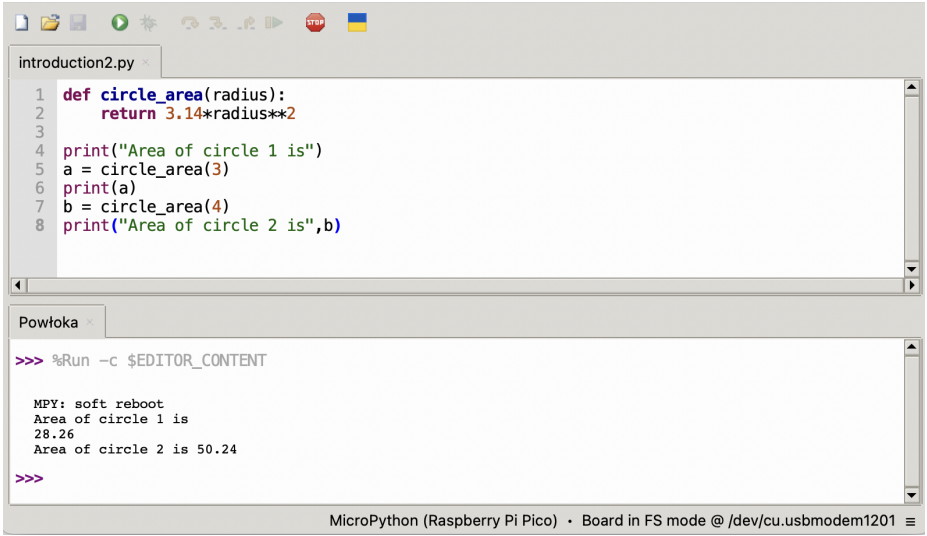
- **Funkcje** - są podprogramami, które wykonują określone operacje.
- Funkcja może przyjmować dane wejściowe zwane **argumentami** funkcji i zwracać wynik.
- Na przykład możesz napisać funkcję, która oblicza pole koła, wtedy argumentem funkcji będzie promień a rezultatem wartość pola.
- **Ważne:** Wszystko co ma być wewnątrz funkcji musi być przesunięte o tabulator (4 spacje) względem *def*.
- Potęgowanie: $x^y \rightarrow x**y$.

```
def nazwaFunkcji(argument1,
    ↪argument2):
    polecenie1
    polecenie2
    ...
    return wynik
```

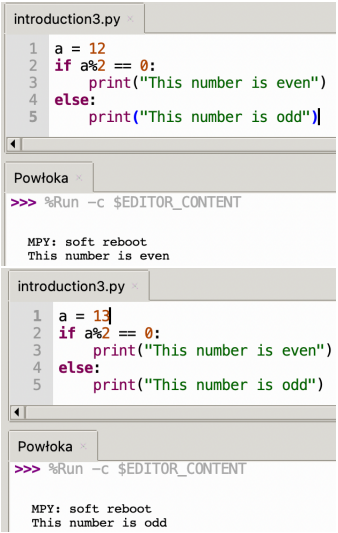
```
def circle_area(radius):
    area =      3.14*radius**2
    return area
```

```
def circle_area(radius):
    return 3.14*radius**2
```

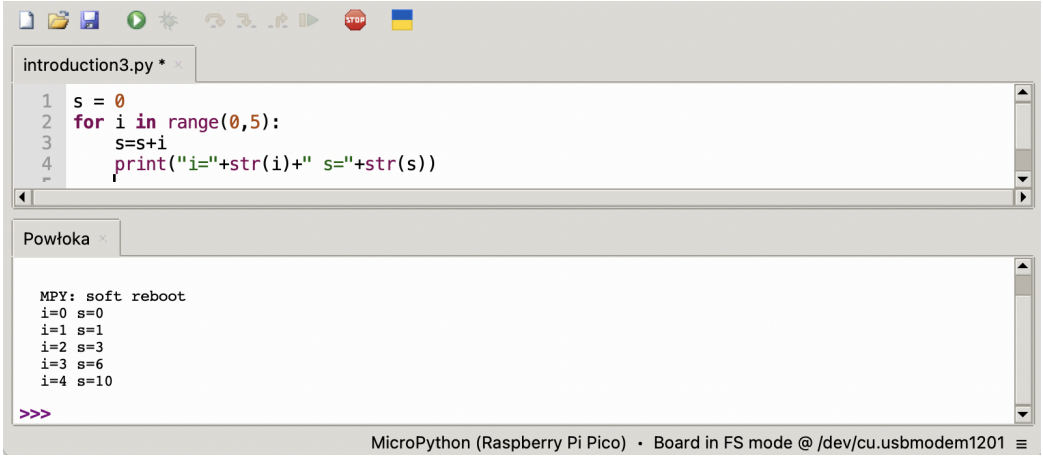
Wstęp do języka MicroPython



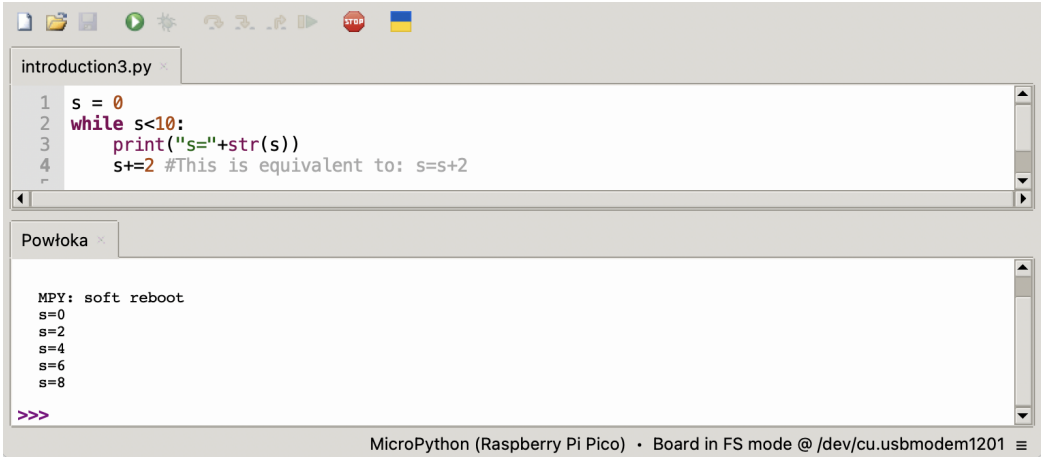
- **Struktura if...else** - umożliwia wykonywanie różnych fragmentów kodu w zależności od spełnionego warunku.
- Wszystkie polecenia, które umieścimy wewnątrz *if* (po wcięciu), zostaną wykonane tylko wtedy, gdy warunek zostanie spełniony.
- Następnie możemy, ale nie musimy, dodać blok *else*, który wykona się tylko wtedy, gdy warunek w *if* nie został spełniony.
- Podstawowe metody porównania:
  - $a == b$ - sprawdzenie czy a jest równe b,
  - $a > b$ - sprawdzenie czy a jest większe niż b,
  - $a >= b$ - sprawdzenie czy a jest większe bądź równe b,
  - $a < b$ - sprawdzenie czy a jest mniejsze niż b,
  - $a <= b$ - sprawdzenie czy a jest mniejsze bądź równe b.



- **Pętla for** - jest pętlą, która umożliwia powtórzenie danego fragmentu kodu kilka razy.
- Podczas wykonywania poniższego przykładu, zmienna *i* przyjmuje wartości z zakresu od 0 do 5.



- **Pętla while** - wykonuje dany fragment kodu tak długo, jak długo spełniony jest warunek.

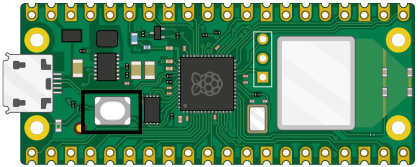


Ogólna struktura każdego programu

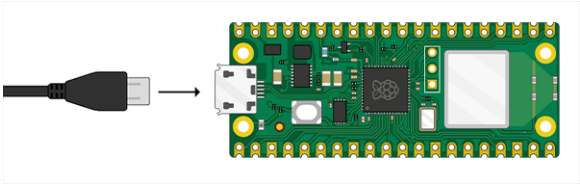
```
#Dodanie niezbędnych bibliotek, które zawierają przydatne funkcje
import nazwa_biblioteki

#Utworzenie zmiennych
#Wykonanie poleceń, które mają wykonać się tylko raz np. różnego
   ↪rodzaju konfiguracje

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

Instalacja płytki Raspberry Pi Pico na komputerze

1. Przytrzymaj przycisk BOOSTEL na płytce Raspberry Pi Pico:

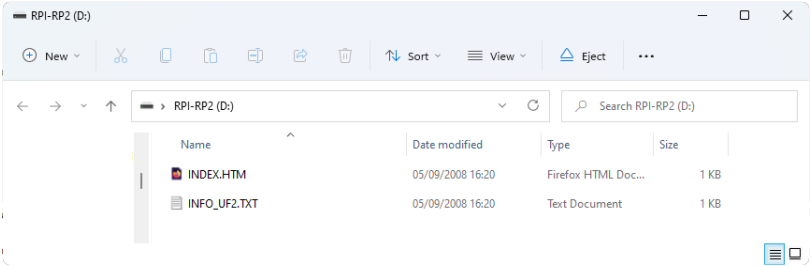


2. Podłącz płytkę Raspberry Pi Pico W do komputera trzymając przycisk BOOSTEL:



Instalacja płytki Raspberry Pi Pico na komputerze

3. Płytka Raspberry Pi Pico W będzie widoczna analogicznie jak pendrive. Po otwarciu katalogu *RPI* zobaczysz takie pliki:

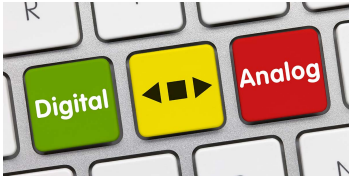


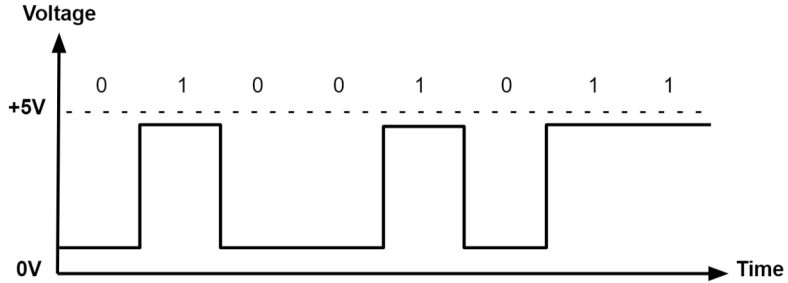
4. Pobierz plik **.UF2* ze strony `https://rpf.io/pico-w-firmware`.
5. Pobrany plik wklej do katalogu *RPI*.
6. Zainstaluj edytor *Thonny*. Instalator znajdziesz tutaj: `https://thonny.org`.

Rodzaje sygnałów

Rozpatrzmy 3 rodzaje sygnałów:

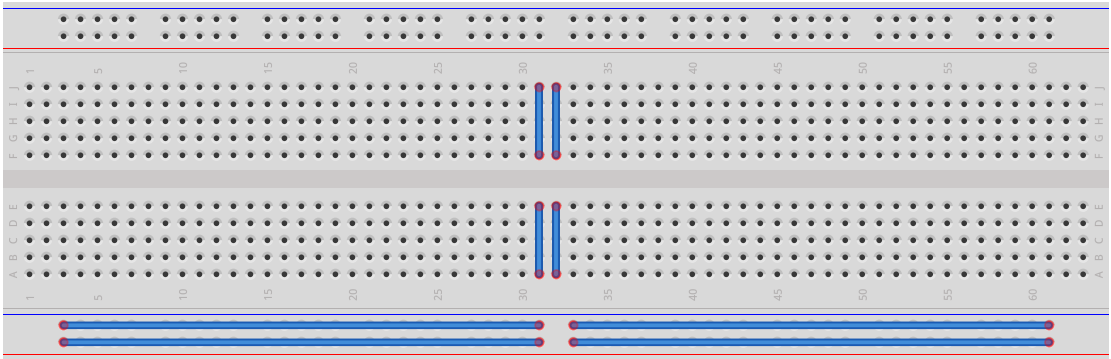
- cyfrowe,
- analogowe,
- Pulse-Width Modulation (PWM).





Źródła obrazków: https://www.nutsvolts.com, https://www.monolithicpower.com

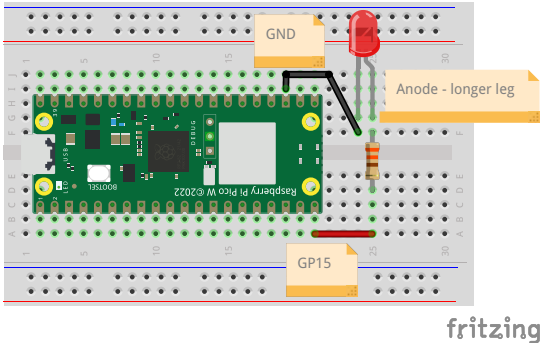
Płytka stykowa



Zadanie 1- migająca dioda LED

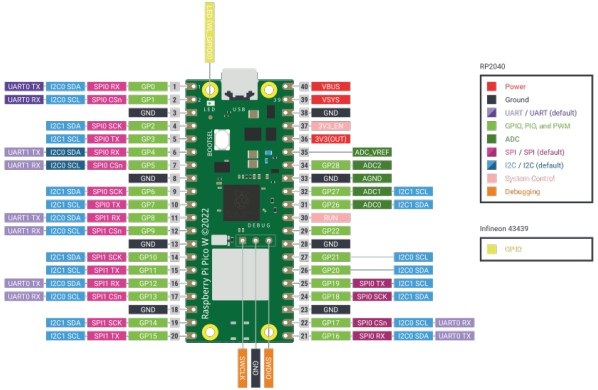
- Cel: napisz program, który będzie cyklicznie zapalał i gasił diodę LED co 1s.
- Typowa dioda LED do jasnego świecenia wymaga napięcia na poziomie ok 1,7 V (U_z) i natężenia prądu od 1 do 15 mA. Aby natężenie prądu nie przekraczało 15 mA, należy podłączyć rezystor o wartości np.:

$$R = \frac{3,3V - U_z}{I} = \frac{3,3V - 1,7V}{4,9mA} \approx 330\Omega \quad (1)$$



GPIO

- Diodę LED można podłączyć do wybranego pinu GPIO (*General Purpose Input/Output*), oznaczonego jasnym zielonym na rysunku:

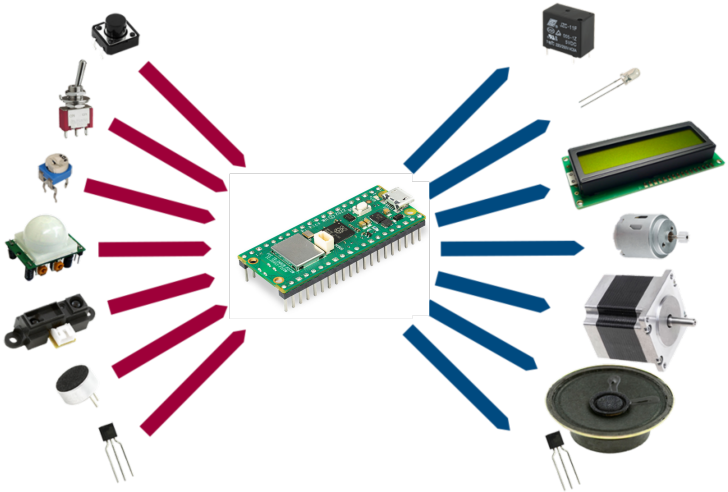


Źródło: `https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html`

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/

Zadanie 1

1. Na początku należy dodać bibliotekę, która zawiera niezbędne funkcje do komunikacji z płytką Raspberry Pi Pico:

```
1 import machine
```

2. Dioda LED jest elementem **wyjściowym** gdyż to mikrokontroler wysyła do niej sygnał wysoki, by zapalić diodę, bądź niski by ją zgasić. Zatem należy skonfigurować pin GP15 jako wyjściowy. Do tego służy funkcja: *machine.Pin()*.

Konfiguracja pinów

Funkcja **machine.Pin(numer_pinu, rodzaj_pinu)** służy do konfiguracji pinu jako wejściowy (**machine.Pin.IN**) bądź wyjściowy (**machine.Pin.OUT**).

```
1 import machine
2
3 led = machine.Pin(15, machine.Pin.OUT)
```

Zadanie 1

3. Następnie należy umieścić nieskończoną pętlę, w której znajdą się instrukcje powtarzane w kółko przez Raspberry Pi Pico W:

```
1 import machine
2
3 led = machine.Pin(15, machine.Pin.OUT)
4
5 while True:
```

4. Następnie zapalamy diodę LED wysyłając sygnał wysoki (1) na pin GP15:

```
1 import machine
2
3 led = machine.Pin(15, machine.Pin.OUT)
4
5 while True:
6     led.value(1)
```

Zadanie 1

5. Teraz program powinien odczekać 1s zanim zgasimy diodę. W tym celu należy skorzystać z funkcji *sleep(opoźnienie)*, która jest dostępna w bibliotece *utime*. Stąd najpierw należy dodać bibliotekę *utime* a potem opóźnienie 1s:

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

Zadanie 1

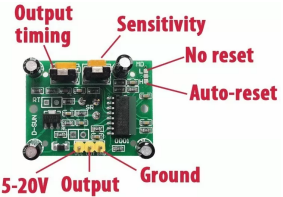
6. Ostatni krok to zgaszenie diody LED wysyłając sygnał niski (0) na pin GP15 oraz odczekanie 1s:

```
import machine
import utime

led = machine.Pin(15, machine.Pin.OUT)

while True:
        led.value(1)
        utime.sleep(1)
        led.value(0)
        utime.sleep(1)
```

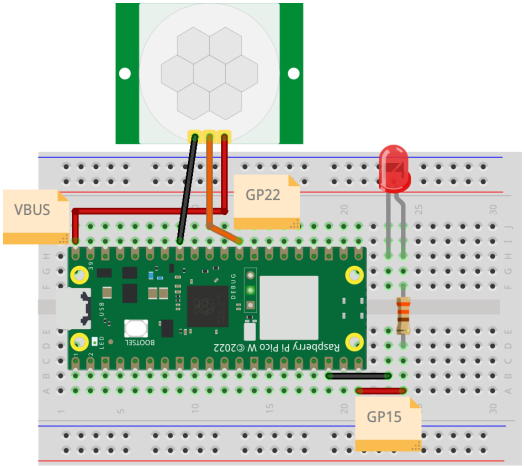
7. Program jest już gotowy. Uruchom go i przetestuj działanie!

Zadanie 2

- Cel: Zapalenie diody LED po wykryciu ruchu przez czujnik ruchu PIR.
- Uwaga: czujnik ruchu wymaga zasilania min. 5V a Raspberry Pi Pico zapewnia 3.3V. Napięcie 5V jest wyprowadzone na pinie **VBUS** i zapewnione tylko gdy Raspberry Pi Pico jest zasilana przez USB.
- Jeśli używasz czujnik zasilany napięciem 5V, upewnij się, że zwracane napięcie przez czujnik jest nie większe niż 3.3V w dokumentacji.



Źródło: https://www.unoelectro.com.ar



Zadanie 2

1. Na początku należy dołączyć niezbędne biblioteki:

```
1 import machine
2 import utime
```

2. Następnie należy skonfigurować pin GP15, do którego podłączona jest dioda LED, jako wyjściowy oraz zgasić diodę LED:

```
1 import machine
2 import utime
3
4 LED = machine.Pin(15, machine.Pin.OUT)
5 LED.value(0)
```

Zadanie 2

3. Pin GP22, do którego podłączono czujnik ruchu PIR, należy skonfigurować jako wejściowy gdyż z czujnika ruchu odczytujemy informacje czy wykryto ruch czy nie:

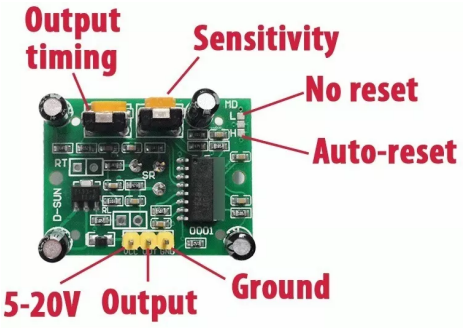
```
import machine
import utime

LED = machine.Pin(15, machine.Pin.OUT)
LED.value(0)
PIR = machine.Pin(22, machine.Pin.IN)
```

4. Następnie należy stworzyć główną pętle *while*, w której będą odczytywane wartości zwracane przez czujnik ruchu:

```
import machine
import utime

LED = machine.Pin(15, machine.Pin.OUT)
LED.value(0)
PIR = machine.Pin(22, machine.Pin.IN)
while True:
        motion = PIR.value()
        print(motion)
```

Zadanie 2

5. Teraz uruchom program i przetestuj działanie czujnika ruchu. Jeśli jest to konieczne to zmień wartości potencjometrów "Output timing" oraz "Sensitivity" tak by czujnik zachowywał się zgodnie z Twoimi oczekiwaniami.
6. Sugerujemy ustawienie "Output timing" na najniższą wartość.



Źródło: https://www.unoelectro.com.ar

Zadanie 2

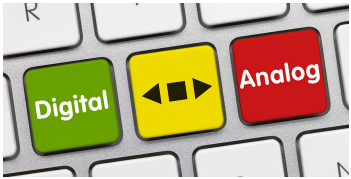
7. Wróćmy do programu i dodajmy aby dioda zapalała się gdy wykryto ruch przez $5s$:

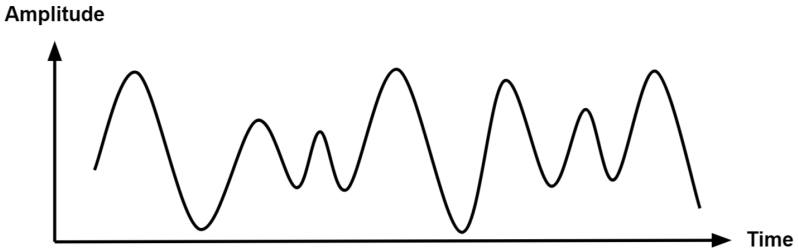
```
import machine
import utime

LED = machine.Pin(15, machine.Pin.OUT)
LED.value(0)
PIR = machine.Pin(22, machine.Pin.IN)

while True:
        motion = PIR.value()
        print(motion)

        if(motion==1):
                LED.value(1)
                utime.sleep(5)
        else:
                LED.value(0)
```

Rodzaje sygnałów

Rozpatrzmy 3 rodzaje sygnałów:

- cyfrowe,
- analogowe,
- Pulse-Width Modulation (PWM).

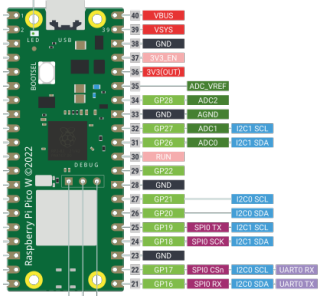




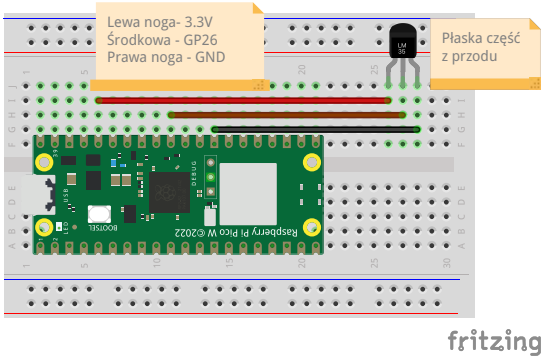
Źródła obrazków: https://www.nutsvolts.com, https://www.monolithicpower.com

Zadanie 3

- Cel: odczytanie temperatury z czujnika LM35
- Czujnik LM35 zwraca napięcie, którego wartość jest proporcjonalna do temperatury.
- Czujniki analogowe można podłączyć do pinów ADC czyli GP26, GP27 i GP28.



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



Zadanie 3

1. Na początku dodajemy niezbędne biblioteki oraz konfigurujemy pin GP26 aby pracował jako pin analogowy. Do tego służy funkcja *ADC(numer pinu)*:

```
import machine
import utime

external_temp = machine.ADC(26)
```

2. Następnie odczytujemy napięcie na pinie GP26 korzystając z funkcji *read_u16()*. Funkcja ta zwraca wartość 16bitową czyli maksymalna wartość to 65535, która odpowiada napięciu 3.3V. Stąd aby uzyskać wartość napięcia należy wynik przemnożyć przez 3.3V i podzielić przez 65535:

```
import machine
import utime

external_temp = machine.ADC(26)
while True:
        voltage = external_temp.read_u16()*3.3/65535
```

Zadanie 3

3. Następnie należy zamienić odczytane napięcie na temperaturę. Zgodnie z dokumentacją napięcie zmienia się z temperaturą o $10mV/^oC$. Zatem odczytany wynik należy pomnożyć przez 100:

```
import machine
import utime

external_temp = machine.ADC(26)

while True:
    voltage = external_temp.read_u16()*3.3/65535
    temp = voltage*100
    print("Temp="+str(temp))
    utime.sleep(1)
```

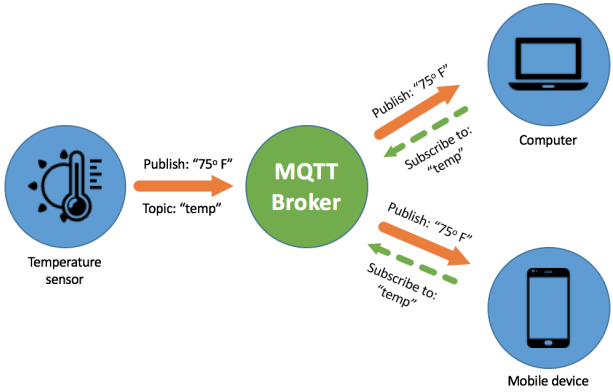
4. Program jest gotowy!

Adafruit IO

- Układy IoT wysyłają zebrane dane do chmury.
- Obecnie do dyspozycji jest wiele różnych chmur, które różnią się możliwościami, ceną oraz trudnością w używaniu.
- Na tym szkoleniu skupimy się na chmurze **Adafruit IO**, która w wersji darmowej zapewnia możliwość zbierania danych z 10 różnych czujników i utworzenie 5 różnych pulpitów do wyświetlania danych.
- Chmura Adafruit IO oparta jest o protokół MQTT (*Message Queuing Telemetry Transport*).

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/

Schemat postępowania

Zadanie 4

Cel

Przesłanie zmierzonej temperatury przez czujnik LM35 do chmury Adafruit IO.

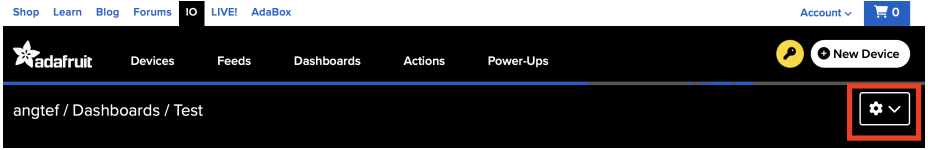
Kroki:

1. Na początku utwórz konto na platformie Adafruit IO: `https://io.adafruit.com`
2. Aby przesłać dane do chmury, należy utworzyć *feed*.
3. *Feed* to obiekt, który przechowuje dane. Aby utworzyć *feed*, należy przejść do zakładki *Feed* na platformie Adafruit IO i kliknąć na *New Feed* a następnie pojawi się okienko, w którym należy nadać nazwę *feed* np. *Temp*:

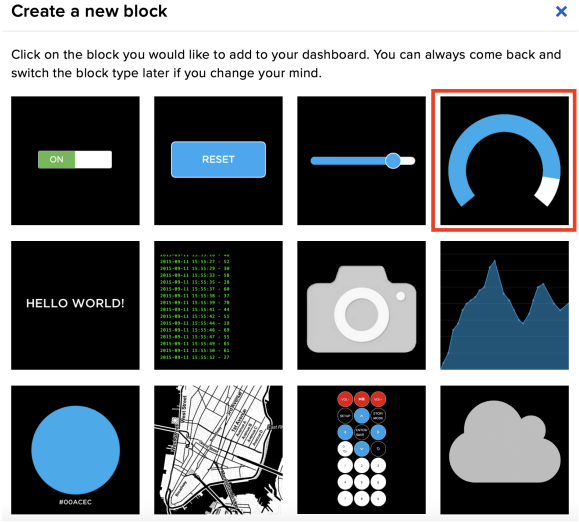
Zadanie 4

4. Następnie należy utworzyć pulpit. W tym celu należy przejść do zakładki *Dashboards* i wybrać przycisk *New Dashboard*:

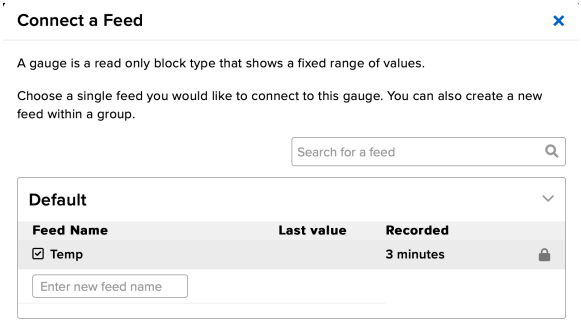
5. Następnie wyskoczy okno z wyborem nazwy pulpitu.
6. Następnie po prawej stronie kliknij na ikonę ustawień i wybierz: *Create New Block*:



Zadanie 4

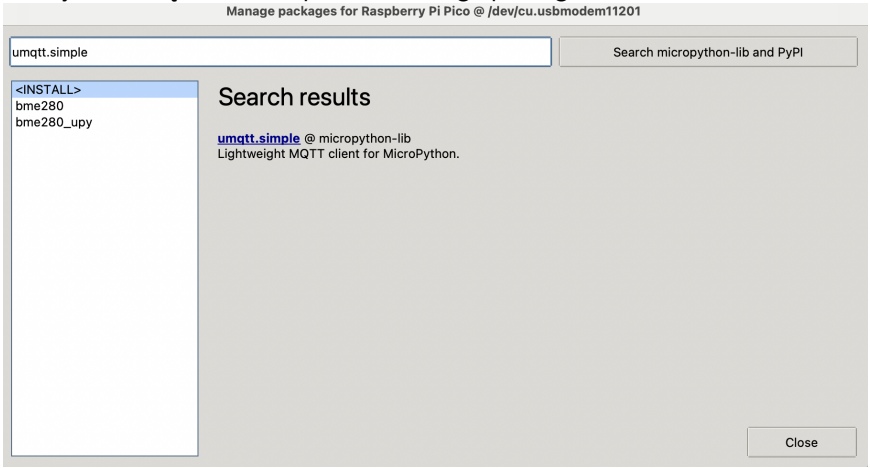

Create a new block
Click on the block you would like to add to your dashboard. You can always come back and switch the block type later if you change your mind.
ON
RESET
HELLO WORLD!
#0DACEC

Zadanie 4

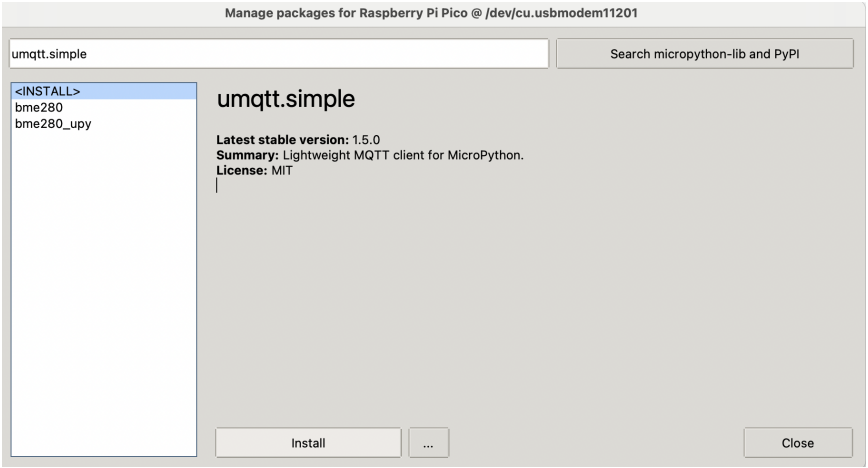

Connect a Feed
A gauge is a read only block type that shows a fixed range of values.
Choose a single feed you would like to connect to this gauge. You can also create a new feed within a group.
Search for a feed
Default
Feed Name
Last value
Recorded
Temp
3 minutes
Enter new feed name

Zadanie 4

7. Teraz powróćmy do edytora Thonny i zainstalujmy bibliotekę *umqtt.simple*. W tym celu wybierz w edytorze Thonny zakładkę "Tools" a potem "Manage packages":



Zadanie 4


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

Zadanie 4

8. Powróćmy do kodu z zadania 3. Dodajmy niezbędne biblioteki by połączyć się przez WiFi i do protokołu MQTT:

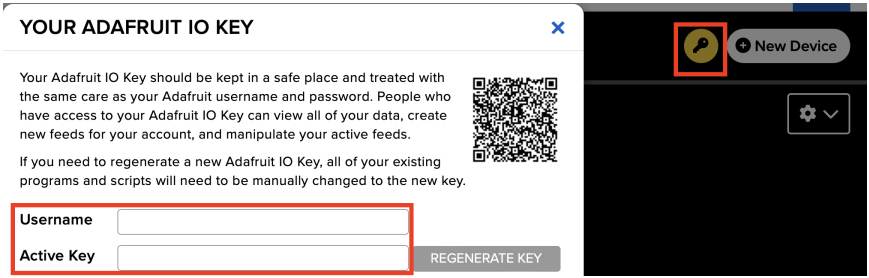
```
import machine
import utime
import network
from umqtt.simple import MQTTClient

external_temp = machine.ADC(26)

while True:
    voltage = external_temp.read_u16()*3.3/65535
    temp = voltage*100
    print("Temp="+str(temp))
    utime.sleep(1)
```

Zadanie 4

9. Utwórzmy zmienne, które będą przechowywały dane sieci WiFi oraz klucz do konta Adafruit IO:

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

external_temp = machine.ADC(26)

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

while True:
    voltage = external_temp.read_u16()*3.3/65535
    ...
```

Zadanie 4


YOUR ADAFRUIT IO KEY
Your Adafruit IO Key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your Adafruit IO Key can view all of your data, create new feeds for your account, and manipulate your active feeds.
If you need to regenerate a new Adafruit IO Key, all of your existing programs and scripts will need to be manually changed to the new key.
Username
Active Key
REGENERATE KEY
New Device

Zadanie 4

Stwórzmy funkcję do łączenia się z siecią WiFi:

```
import machine
...
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() #Wywołanie funkcji
```

Zadanie 4

- Teraz przejdźmy do protokołu MQTT. Na początku stwórzmy zmienne do przechowywania parametrów chmury i nazwę *feed*, którego stworzyliśmy:

```
...
connect_wifi() #Wywołanie funkcji

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

FEED_TEMP_LEVEL = "username/feeds/Temp"
...
```

Zadanie 4

12. Teraz należy utworzyć klienta:

```
...
connect_wifi() #Wywołanie funkcji

ADAFRUIT_IO_SERVER = "io.adafruit.com"
ADAFRUIT_IO_PORT = 1883  # Port MQTT
CLIENT_ID = "raspberry_pi_pico"
FEED_TEMP_LEVEL = "username/feeds/Temp"

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_mqtt() #Wywołanie funkcji
```

13. Teraz stwórzmy funkcję wysyłającą wartość temperatury do chmury:

```
...
connect_mqtt() #Wywołanie funkcji

def send_data(temp):
    client.publish(FEED_TEMP_LEVEL, str(temp))
    print("Temp. sent:"+str(temp))
```

14. Ostatni krok to wywołanie funkcji w głównej pętli programu by przesłać aktualną wartość temperatury:

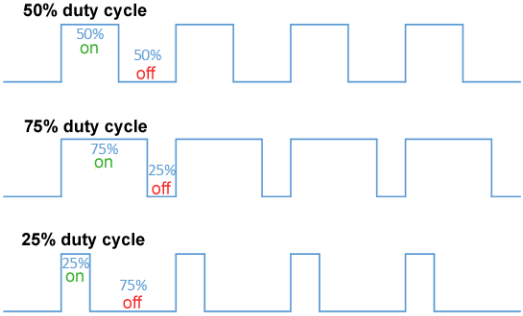
```
...
while True:
    voltage = external_temp.read_u16()*3.3/65535
    temp = voltage*100
    print("Temp="+str(temp))
    send_data(temp)
    utime.sleep(1)
```

15. Uruchom program i przejdź do pulpitu na platformie Adafruit IO.

Rodzaje sygnałów

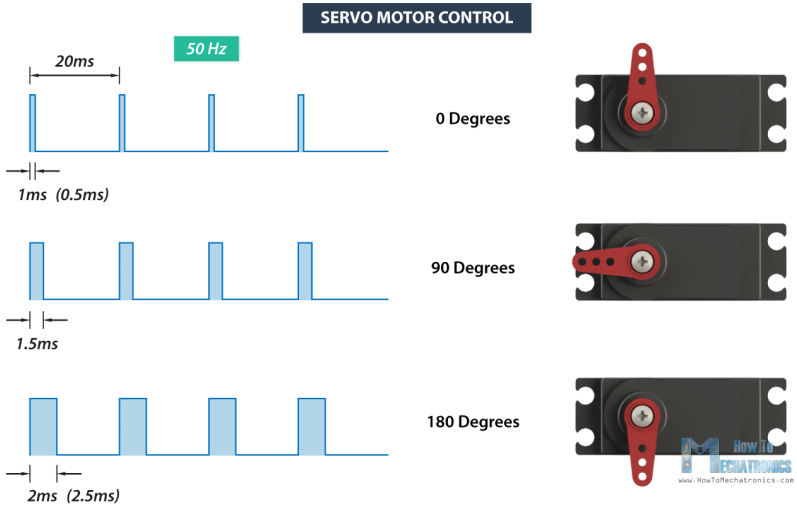
Rozpatrzmy 3 rodzaje sygnałów:

- cyfrowe,
- analogowe,
- Pulse-Width Modulation (PWM).



https://commons.wikimedia.org

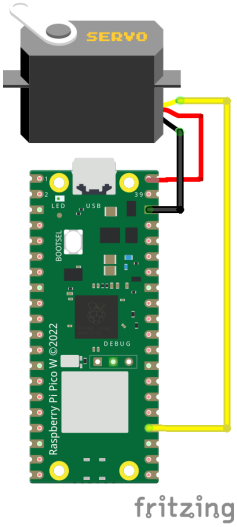
Zadanie 5 (dodatkowe) - serwomechanizm



Źródło: https://howtomechatronics.com/wp-content/uploads/2018/03/RC-Servo-Motor-Control-Signal.png.

Zadanie 5 (dodatkowe)

- Cel: obrócenie serwomechanizmu od pozycji minimalnej, przez środkową do maksymalnej i z powrotem.
- Podłączenie serwomechanizmu:
  - czerwony przewód - VBUS,
  - żółty przewód - GP18,
  - czarny przewód - GND.



Zadanie 5 (dodatkowe)

1. Najpierw należy dołączyć niezbędne biblioteki oraz utworzyć zmienne przechowujące ile czasu ma trwać sygnał wysoki aby serwomechanizm był w pozycji minimalnej (0.9ms), środkowej (1.5 ms) i maksymalnej (2.1 ms).

```
1 import machine
2 import utime
3
4 MIN = 900000
5 MID = 1500000
6 MAX = 2100000
```

Zadanie 5 (dodatkowe)

2. Następnie należy skonfigurować pin GP18 aby generował sygnał metodą PWM. Do tego służy funkcja *PWM(numer_pinu)*:

```
1 import machine
2 import utime
3
4 MIN = 900000
5 MID = 1500000
6 MAX = 2100000
7
8 pwm = machine.PWM(machine.Pin(18))
```

Zadanie 5 (dodatkowe)

3. W kolejnym kroku ustawiamy częstotliwość sygnału na 50 Hz korzystając z funkcji *freq(częstotliwość)*:

```
import machine
import utime

MIN = 900000
MID = 1500000
MAX = 2100000

pwm = machine.PWM(machine.Pin(18))
pwm.freq(50)
```

Zadanie 5 (dodatkowe)

4. Aby serwomechanizm obrócił się do danej pozycji, należy ustawić wypełnienie sygnału funkcją *duty_ns(czas trwania impulsu wysokiego)*. Początkowo ustawmy serwomechanizm w pozycji minimalnej:

```
1  import machine
2  import utime
3
4  MIN = 900000
5  MID = 1500000
6  MAX = 2100000
7
8  pwm = machine.PWM(machine.Pin(18))
9  pwm.freq(50)
10 pwm.duty_ns(MIN)
```

Alternatywnie można ustawić wypełnienie sygnału korzystając z funkcji *duty_u16(wartość)*, która przyjmuje wartości od 0 do 65535 gdzie 65535 to 100% czasu trwania sygnału wysokiego.

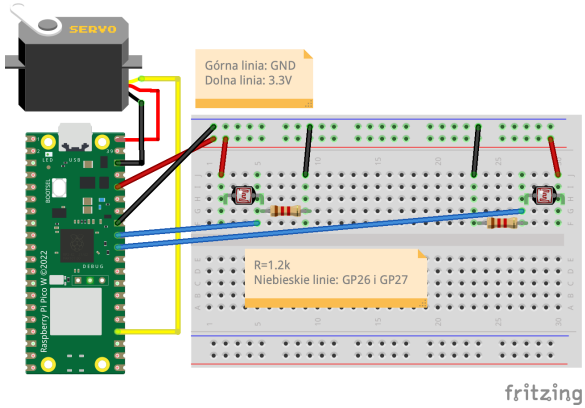
Zadanie 5 (dodatkowe)

5. Ostatni krok to stworzenie głównej pętli, w której będziemy zmieniać położenie serwomechanizmu z pozycji minimalnej do środkowej a potem do maksymalnej i z powrotem. Między każdą pozycją ustawmy opóźnienie $1s$ by zobaczyć efekt:

```
...
pwm.duty_ns(MIN)

while True:
    pwm.duty_ns(MIN)
    utime.sleep(1)
    pwm.duty_ns(MID)
    utime.sleep(1)
    pwm.duty_ns(MAX)
    utime.sleep(1)
    pwm.duty_ns(MID)
    utime.sleep(1)
```

Zadanie 6 (dodatkowe)

- Cel: stworzenie układu, który będzie obracał panele fotowoltaiczne w stronę światła.
- W tym celu zostanie wykorzystany serwomechanizm symulujący układ obracający panel fotowoltaiczny.
- Ponadto skorzystamy z dwóch fotorezystorów, których rezystancja zależy od oświetlenia. Im jaśniej, tym mniejsza rezystancja.
- Jeden fotorezystor będzie umieszczony przed panelem fotowoltaicznym (pozycja MIN serwomechanizmu) a drugi za panelem (pozycja MAX serwomechanizmu).



Zadanie 6 (dodatkowe)

1. Na początku zacznijmy od modyfikacji programu z zadania 5:

```
import machine
import utime

MIN = 900000
MID = 1500000
MAX = 2100000

servo = machine.PWM(machine.Pin(18))
servo.freq(50)
servo.duty_ns(MID)
```

Zadanie 6 (dodatkowe)

2. Teraz skonfigurujmy piny GP26 i GP27 jako analogowe i stwórzmy zmienną, która będzie określać jaka różnica musi być między fotorezystorami aby panel zmienił swoje położenie:

```
import machine
import utime

MIN = 900000
MID = 1500000
MAX = 2100000

servo = machine.PWM(machine.Pin(18))
servo.freq(50)
servo.duty_ns(MID)

fotorezystor1 = machine.ADC(26)
fotorezystor2 = machine.ADC(27)
diff = 1000
```

Zadanie 6 (dodatkowe)

3. Dodajmy główną pętlę while, a w niej odczytajmy wartości zwracane przez fotorezystory:

```
...
fotorezystor1 = machine.ADC(26)
fotorezystor2 = machine.ADC(27)
diff = 1000

while True:
    wartosc1 = fotorezystor1.read_u16()
    wartosc2 = fotorezystor2.read_u16()
    print("Foto1="+str(wartosc1)+" Foto2="+str(wartosc2))
    utime.sleep(1)
```

Zadanie 6 (dodatkowe)

3. Ostatni krok to określenie położenia serwomechanizmu. Jeśli wartość z fotorezystora nr 1 jest większa od wartości z fotorezystora 2 o więcej niż *diff* to niech serwo obróci się do pozycji minimalnej. Gdy wartość z fotorezystora 2 jest większa od *diff* od pierwszego to niech serwo obróci się do pozycji maksymalnej. Jeśli różnica jest w granicach *diff* to niech nic się nie dzieje:

```
...
while True:
    wartosc1 = fotorezystor1.read_u16()
    wartosc2 = fotorezystor2.read_u16()
    print("Foto1="+str(wartosc1)+" Foto2="+str(wartosc2))
    if wartosc1>(wartosc2+diff):
        servo.duty_ns(MIN)
    elif wartosc2>(wartosc1+diff):
        servo.duty_ns(MAX)
    utime.sleep(1)
```

Zadanie 7 (dodatkowe)

- Cel: stworzenie inteligentnej wentylacji.
- W tym celu skorzystamy z silnika DC podłączonego do sterownika DRV8835 oraz czujnika temperatury LM35.
- W zależności od odczytanej wartości temperatury, będziemy regulować prędkość obrotu silnika DC.
- Podłącz czujnik LM35 (patrz zadanie nr 3) + silnik ze sterownikiem:

DRV8835	Raspberry Pi Pico	DC motor
GND	GND	-
VIN, VCC i MD	VBUS	-
O1 i O2	-	dwie nogi
VM	-	-
IN1 PH	GP16	-
IN2 EN	GP17	-

Zadanie 7 (dodatkowe)

1. Zacznijmy od programu z zadania 3 do pomiaru wartości temperatury z czujnika LM35:

```
import machine
import utime

external_temp = machine.ADC(26)

while True:
    voltage = external_temp.read_u16()*3.3/65535
    temp = voltage*100
    print("Temp="+str(temp))
```

Zadanie 7 (dodatkowe)

2. Silnik DC jest kontrolowany za pomocą sygnałów PWM. Stąd skonfigurujmy pin GP17 jako PWM a GP16 jako wyjściowy (do sterowania kierunkiem obrotów silnika). Ponadto ustawmy częstotliwość sygnału PWM na 1 kHz:

```
1  import machine
2  import utime
3
4  external_temp = machine.ADC(26)
5
6  DCmotor = machine.PWM(machine.Pin(17))
7  direction = machine.Pin(16, machine.Pin.OUT)
8
9  DCmotor.freq(1000)
10 direction.value(0)
11
12 while True:
13     voltage = external_temp.read_u16()*3.3/65535
14     temp = voltage*100
15     print("Temp="+str(temp))
```

Zadanie 7 (dodatkowe)

3. Stwórzmy teraz zmienne, które będą przechowywać minimalną i maksymalną temperaturę w pomieszczeniu oraz minimalną i maksymalną prędkość obrotów silnika (wypełnienie sygnału). Ponadto stwórzmy funkcję, która będzie konwertować temperaturę na prędkość obrotów:

```
...
DCmotor.freq(1000)
direction.value(0)

min_speed = 20000
max_speed = 65535
Tmin=22
Tmax=50

def map_value(x, in_min, in_max, out_min, out_max):
    return int((x - in_min) * (out_max - out_min) / (in_max - in_min) +
        ↪out_min)

while True:
```

Zadanie 7 (dodatkowe)

3. Ostatni krok to ustawienie prędkości obrotów silnika w głównej pętli:

```
...
while True:
    voltage = external_temp.read_u16()*3.3/65535
    temp = voltage*100
    print("Temp="+str(temp))

    speed = map_value(temp, Tmin, Tmax, min_speed, max_speed)
    DCmotor.duty_u16(speed)
```

Więcej przykładów w przewodniku o Raspberry Pi Pico:

https://www.iot.fizyka.pw.edu.pl

Dziękuję za uwagę