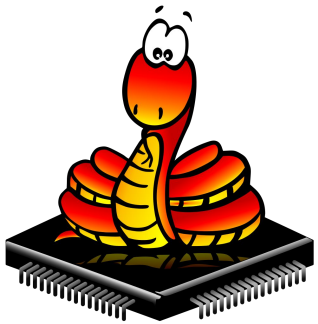


Sparrow
MicroPython

Thursday, September the 28th 2023

Antoine Chauvin & Hugo Reimeringer

Summary

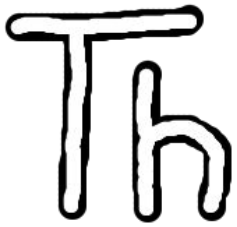
I - Install Thonny and MicroPython (20 min)

II - Get the basics

- a) Blink the integrated LED (10 min)
- b) Read an analog value (30 min)

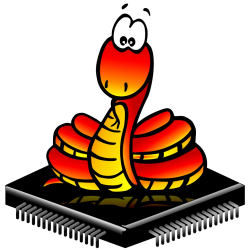
III - Get everything ready

- a) Use the servomotor (20 min)
- b) Use the IMU if you have one (5 min)
- c) Import files (5 min)



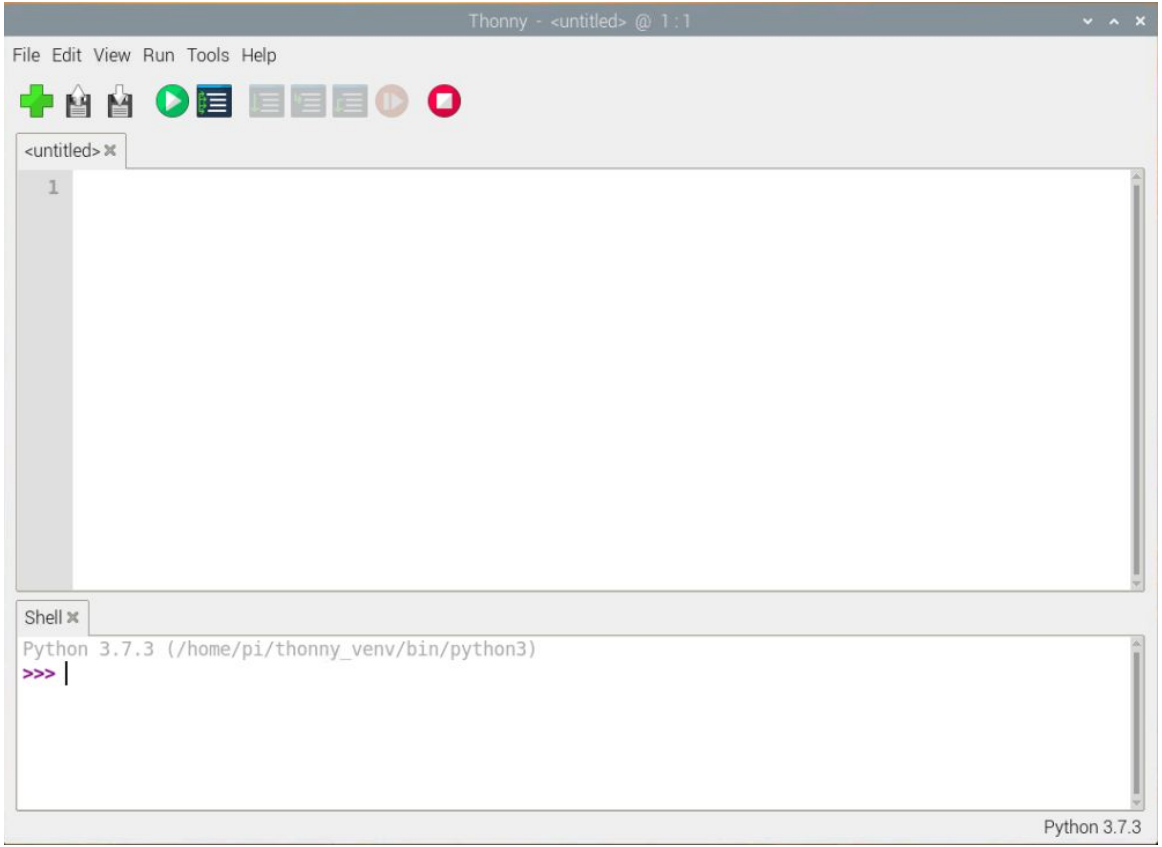
MicroPython

I - Install Thonny and MicroPython

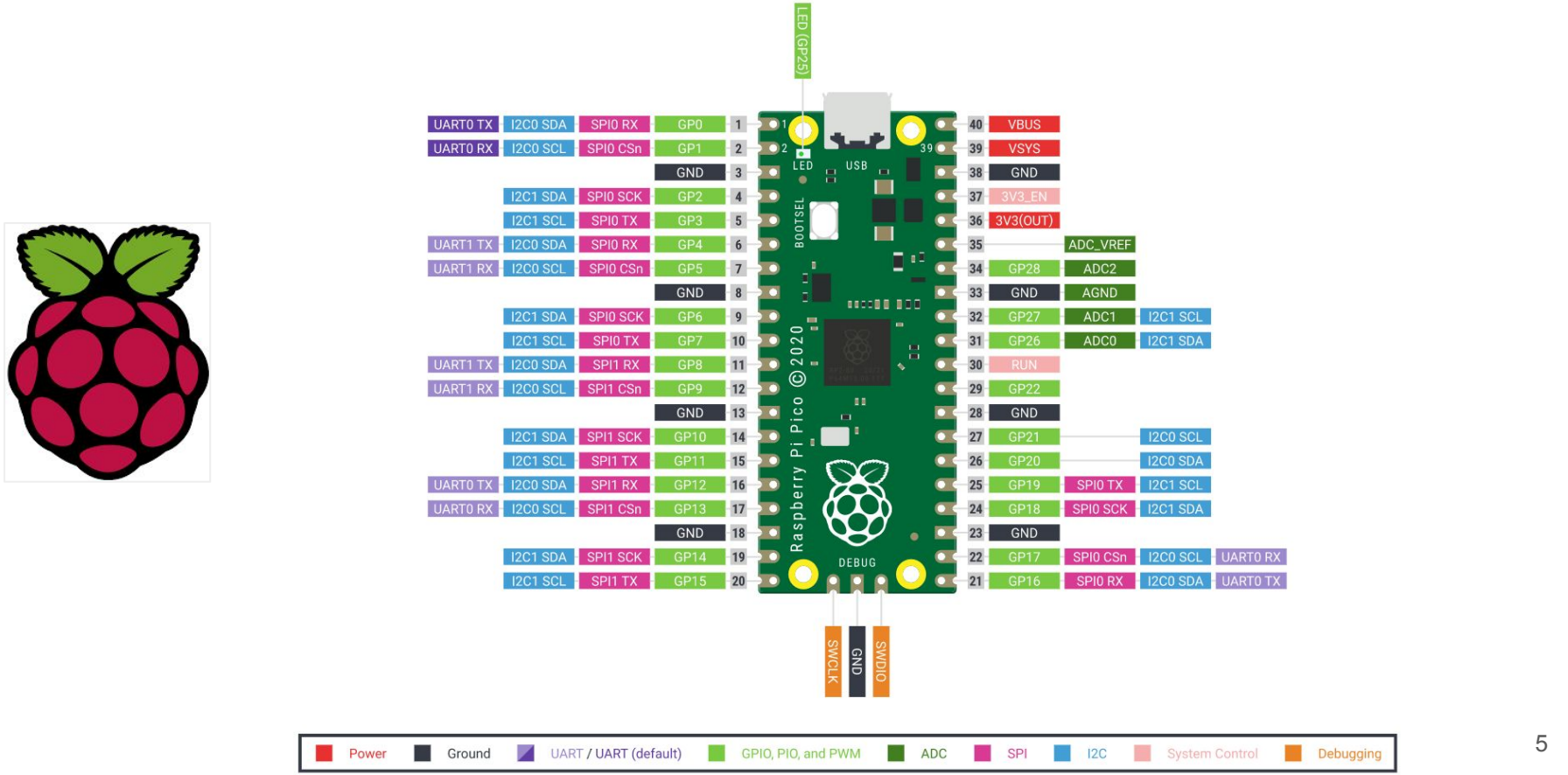


Install Thonny

Download it on :
https://thonny.org/

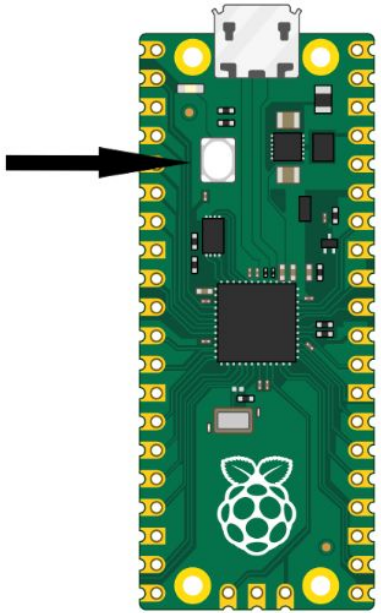


View of the *Raspberry Pi Pico* card



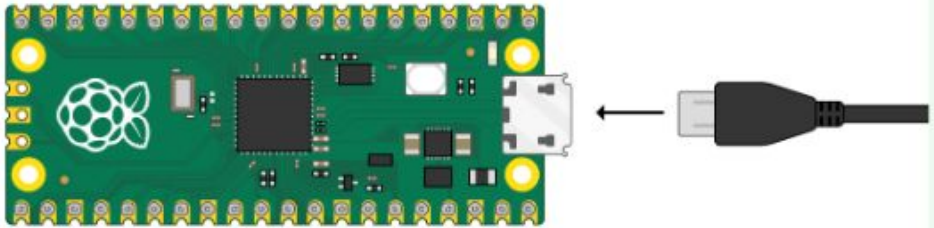
Install Micropython on the card

Find the BOOTSEL button on your Raspberry Pi Pico.



Install Micropython on the card

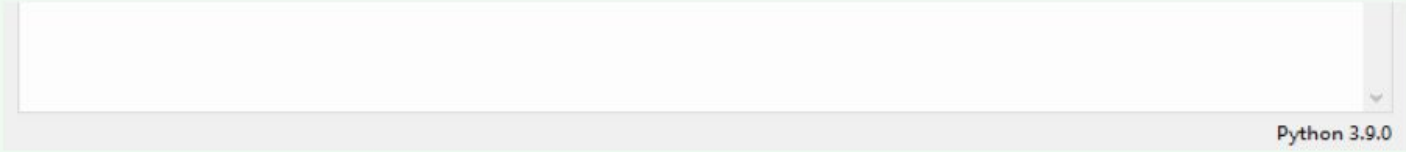
Press the BOOTSEL button and hold it while you connect the other end of the micro USB cable to your computer.



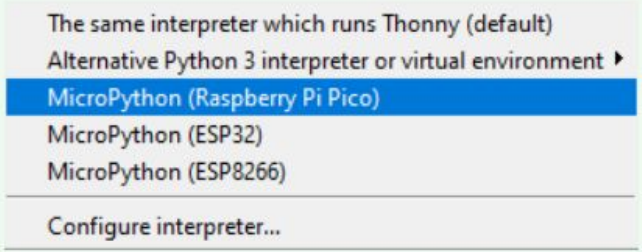
This puts your Raspberry Pi Pico into USB mass storage device mode.

Install Micropython on the card

In the bottom right-hand corner of the Thonny window, you will see the version of Python that you are currently using.



Click on the Python version and choose 'MicroPython (Raspberry Pi Pico)':



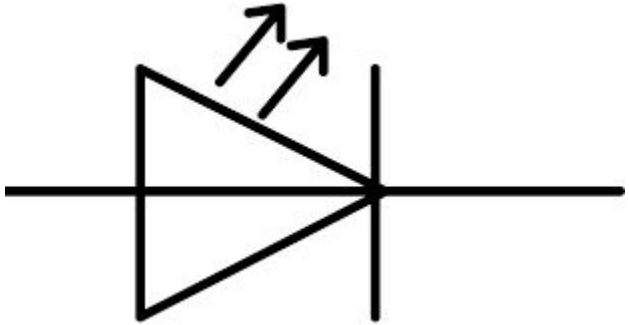
If you don't see this option, then check that you have plugged in your Raspberry Pi Pico.

Install Micropython on the card

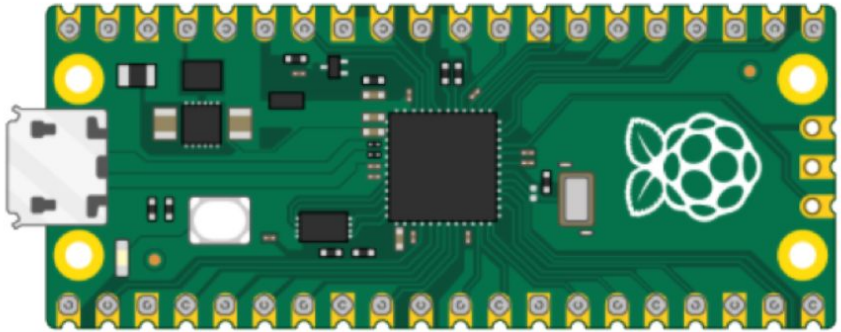
Look at the Shell panel at the bottom of the Thonny editor.

You should see something like this:

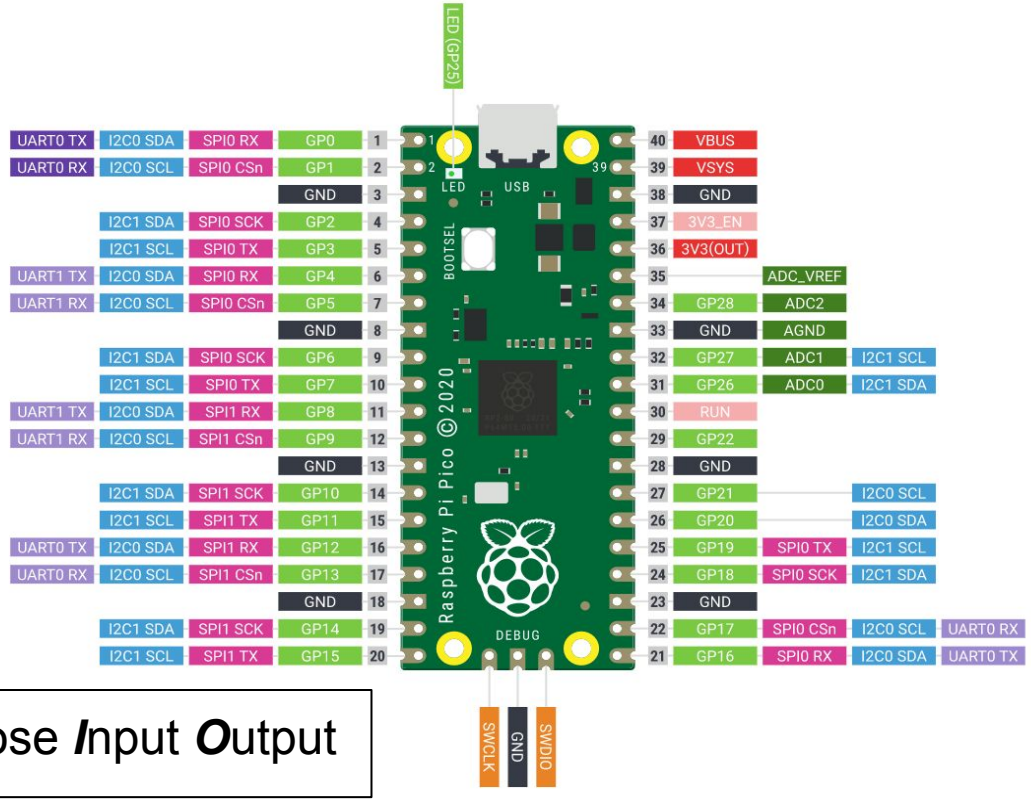
Thonny is now able to communicate with your Raspberry Pi Pico using the REPL (read–eval–print loop), which allows you to type Python code into the Shell and see the output.



II - Get the basics



Reminder of the card characteristics



GPIO : ***G***eneral ***P***urpose ***I***nput ***O***utput

Use the integrated LED

MicroPython adds hardware-specific modules, such as `machine`, that you can use to program your Raspberry Pi Pico.

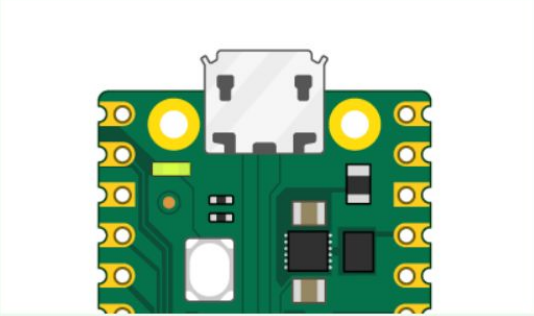
Let's create a `machine.Pin` object to correspond with the onboard LED, which can be accessed using GPIO pin 25.

If you set the value of the LED to `1`, it turns on.

Enter the following code, make sure you tap Enter after each line.

```
from machine import Pin
led = Pin(25, Pin.OUT)
led.value(1)
```

You should see the onboard LED light up.



```
import time

time.sleep(1)
```

The function to wait 1 sec so that you can blink the LED

Read an analog value

ADC = ***A***nalog to ***D***igital ***C***onverter

4.9.5. Temperature Sensor

The temperature sensor measures the Vbe voltage of a biased bipolar diode, connected to the fifth ADC channel (AINSEL=4). Typically, Vbe = 0.706V at 27 degrees C, with a slope of -1.721mV per degree. Therefore the temperature can be approximated as follows:

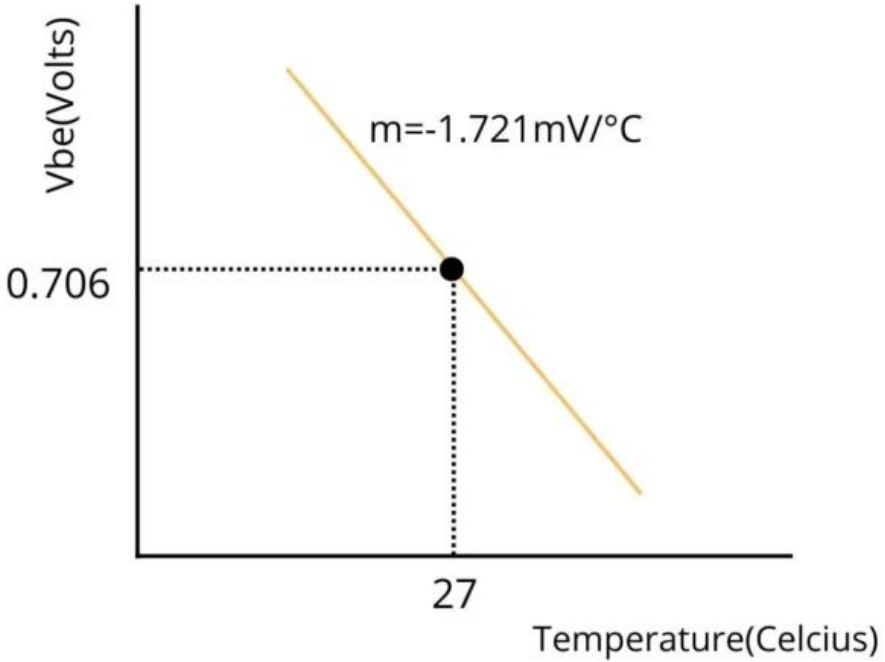
T = 27 - (ADC_voltage - 0.706)/0.001721

As the Vbe and the Vbe slope can vary over the temperature range, and from device to device, some user calibration may be required if accurate measurements are required.

extracted from the datasheet :

https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

Read an analog value



diode voltage (V) as a function of temperature (°C)

Read an analog value

The ADCs have a sampling rate of 500kS/s and use an independent 48Mhz clock for this purpose. Each sample takes 96 clock cycles. So the sampling time taken per sample is (96/48Mhz)=2µS.

The 12-bit ADC register will give us a value ranging from 0 to 4095 (2^12-1). But specific functions in MicroPython can scale the ADC values to give us a 16-bit range. So we can get sensor readings ranging from 0 to 65535 (2^16-1).

Max voltage possible is 3.3V.

Read an analog value

Exercise : Print the temperature of the processor

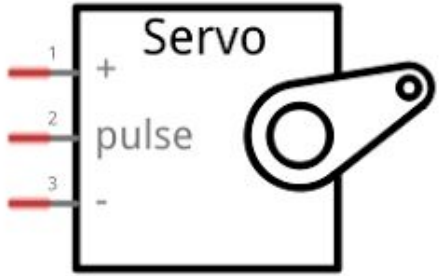
Use the datasheet's content about the sensor…

Given functions :

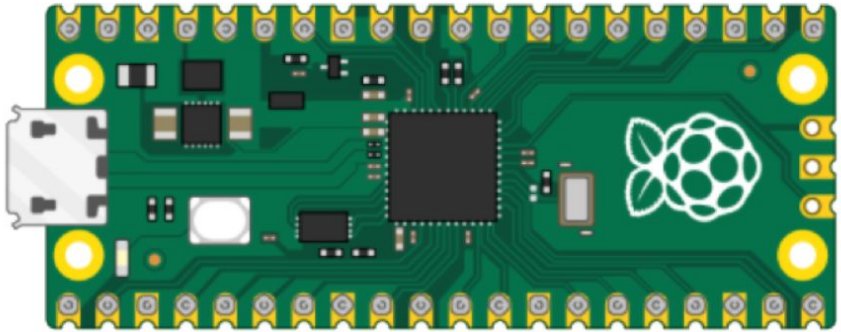
```
1  adc=machine.ADC(XX) #initializes the XX ADC pin in the adc variable
2  adc.read_u16()      #reads the adc's initialized pin (16-bit value)
```

Solution

```
1  from machine import ADC #library for ADC function
2  import time             #library for sleep_ms function
3
4  adc = machine.ADC(4) #correct ADC pin initialized
5  while True: #main loop
6      ADC_voltage = adc.read_u16() * (3.3 / (2**16-1))
7      #converts u16 (16bits) range in Volts
8
9      temperature_celcius = 27 - (ADC_voltage - 0.706)/(1.721e-3)
10     #converts Volts in temperature (diode's characteristics)
11
12     print("Temperature: {}°C".format(temperature_celcius))
13
14     time.sleep_ms(100) #waits for the newt measure (100 ms)
```



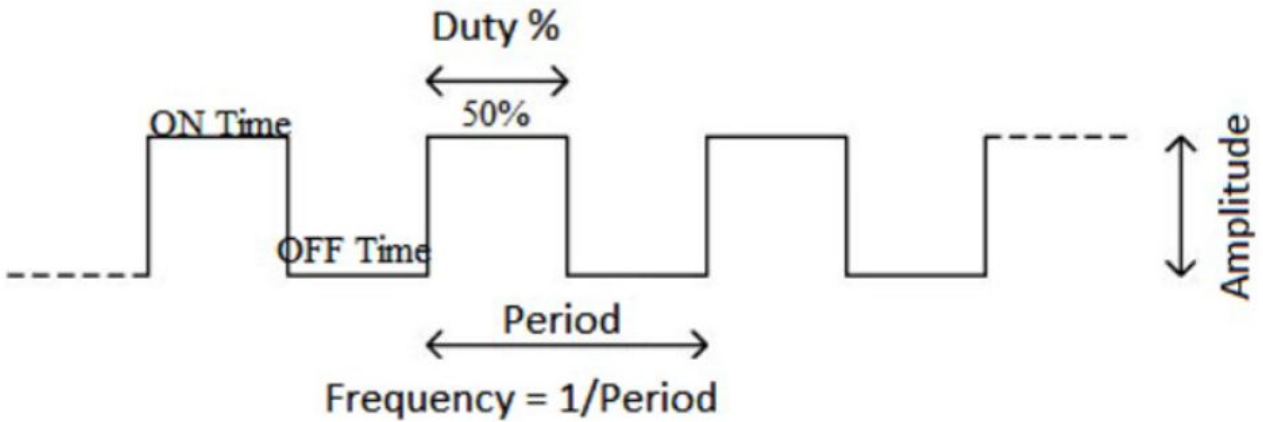
III - Get everything ready



PWM description

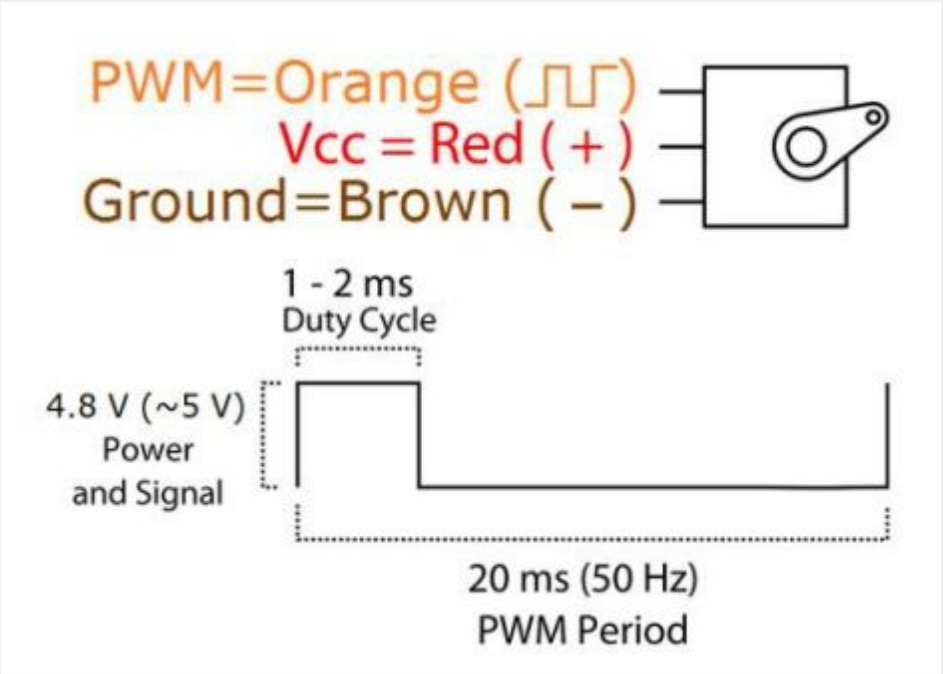
Pulse Width Modulation

Pulse-width modulation (PWM) is a technique for changing the amplitude of digital signals to control devices and applications that require power or electricity. It effectively regulates the amount of power delivered to a device from the standpoint of the voltage component by rapidly cycling the on-and-off phases of a digital signal and changing the breadth of the "on" phase or duty cycle. To the device, this would appear as a constant power input with an average voltage value determined by the proportion of on time. The duty cycle is represented as a percentage of being totally (100%) on.



Use a servomotor

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.



Extract from SG90's Datasheet

Use a servomotor

Exercise : Use the servomotor

servomotor’s datasheet will be useful

Given functions :

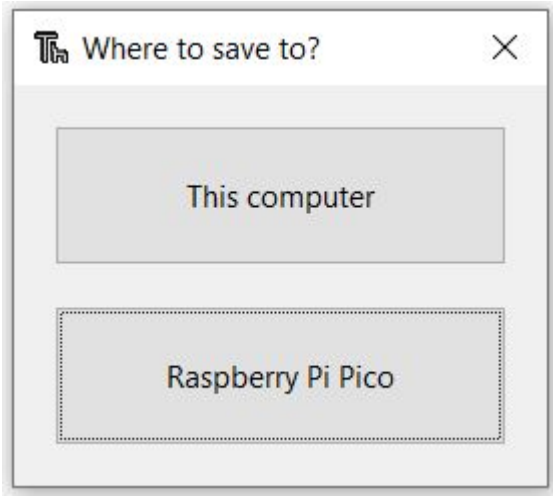
```
pwm = machine.PWM(Pin(XX)) #initialize the XX pin for PWM in pwm
pwm.freq(XX) #set the PWM's frequency to XX Hz
pwm.duty_ns(XX) #set the PWM's duty to XX ns
```

Solution

```
from machine import Pin,PWM
import time

MID = 1500000
MIN = 0600000
MAX = 2200000
#duties(ns) for max, min and mid angles

pwm = PWM(Pin(15)) #initializes PIN 15
pwm.freq(50) #set the correct frequency

angle = 30 #we choose a 30° angle as example
duty = 1.5*10^6 + 0.5*10^6*angle/90
#conversion angle to duty in ns
pwm.duty_ns(duty) #use the servomotor

time.sleep(1)
pwm.deinit()
```

Import files

On big projects, it's easier to separate the code you write



Save the code you want in the raspberry pi Pico
Ex : Name it Xxxx.py

```
1 import Xxxx
```

```
1 from Xxxx import function
```

In a new file, you can import the code with import Xxxx
OR
from Xxxx import function
(with function the name of the function you want to import)

And you can call the function with Xxxx.function() in the first case
OR
function()