CanSat Raspberry Pi Pico Workbook

Table of content


CANSAT

Introduction

The following labs have been designed to introduce you to some of the electronics and programming skills required to undertake the CanSat primary mission.

This document is to the point without being a complete technical guide.
References to full technical guides are given where necessary.

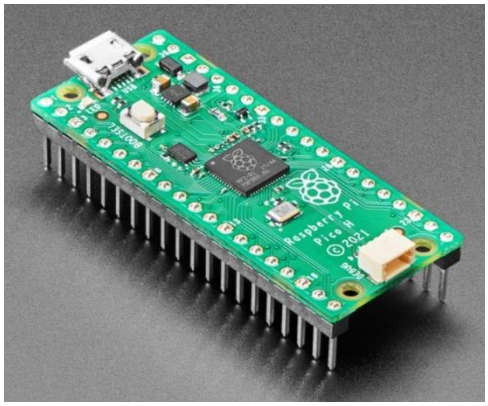
The difficulty of the labs is progressive, starting with wiring and programming steps without any soldering.

If you have never coded in Python, we advise you to look at the Learning Python section

The CanSat Kit

The kit comprises of off-the-shelf hardware that is cheap and easy to buy online. This allows teams to easily replace broken components and to also find support and ideas from the wealth of online teaching tutorials and technical resources related to Raspberry Pi Pico and MicroPython. The kit we use in the labs contains the following items:

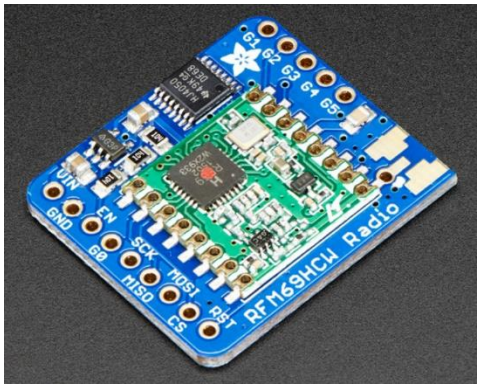
2x Raspberry Pi Pico with headers



The Pico is sold with or without headers pre-soldered. We give you 2 Pico with headers to help you get up to speed and focus early on programming.

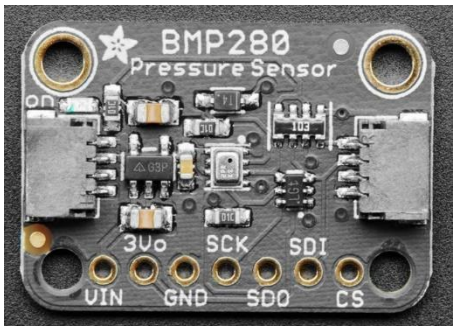
https://www.adafruit.com/product/5525

2x radio transceivers RFM69HCW



https://www.adafruit.com/product/3071

1x BMP280 Pressure/Temperature Sensor



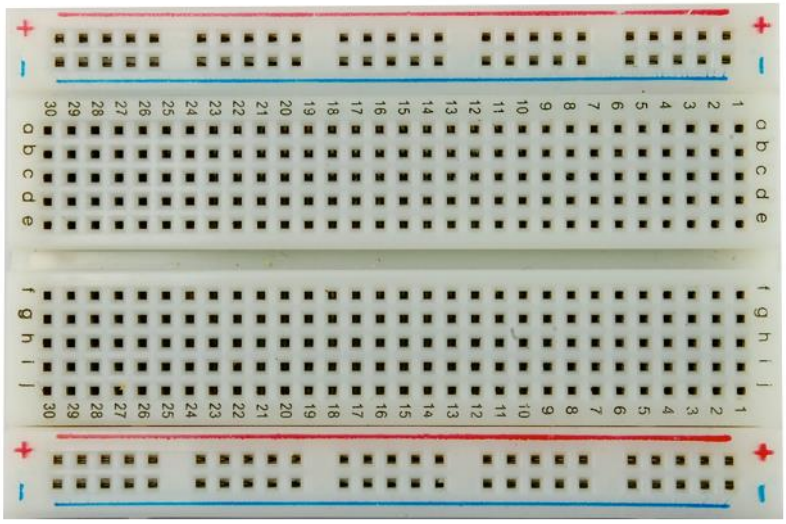
https://www.adafruit.com/product/2651

1x TMP36 sensor

Alternative to the BMP280, the TMP36 is a basic analogue temperature sensor that outputs a voltage based on the ambient temperature around the sensor.

https://learn.adafruit.com/tmp36-temperature-sensor

1x breadboard for prototyping circuits



A breadboard is a construction base used to build semi-permanent prototypes of electronic circuits without any soldering.

1x USB cable

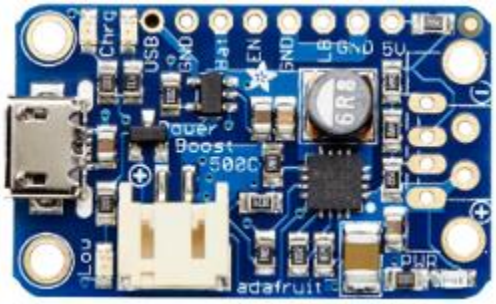
Cable used to plug your Raspberry Pico to a computer to program it or to charge the Lipo lithium battery.

1x lithium battery

3.7V 1300mAh battery to power the Raspberry Pico inside your CanSat, without the USB cable.

https://cdn-shop.adafruit.com/datasheets/Li-poly+603562-1300mAh.pdf

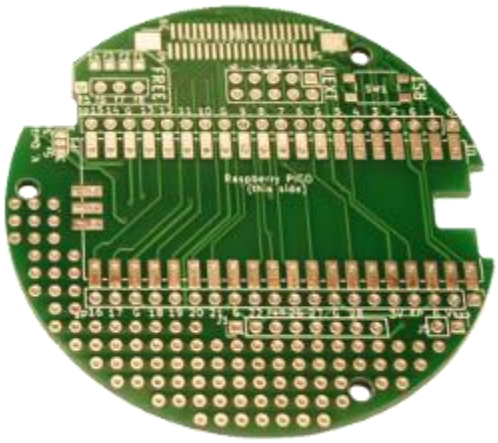
1x 5 volts converter and lithium battery charger



Board that converts the lithium battery power to a 5 volts power source suitable for the Raspberry Pico. It can recharge the lithium battery when the kit is powered via USB.

https://www.adafruit.com/product/1944

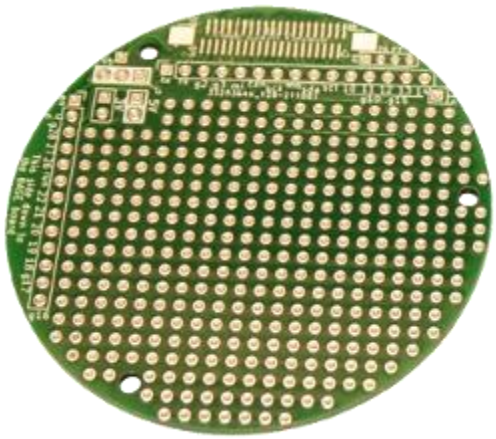
1x Cansat base board for Pico



After having tested the wiring of your components and their related software on the breadboard, you can solder the Raspberry Pico, the 5V converter board and the radio transmitter to this board. Its diameter fits within the maximal CanSat diameter. The BMP280 board can be plugged in using the provided JST cable.

https://shop.mchobby.be/fr/pico-rp2040/2275-carte-de-base-cansat-pour-pico-3232100022751.html

1x Cansat extension board



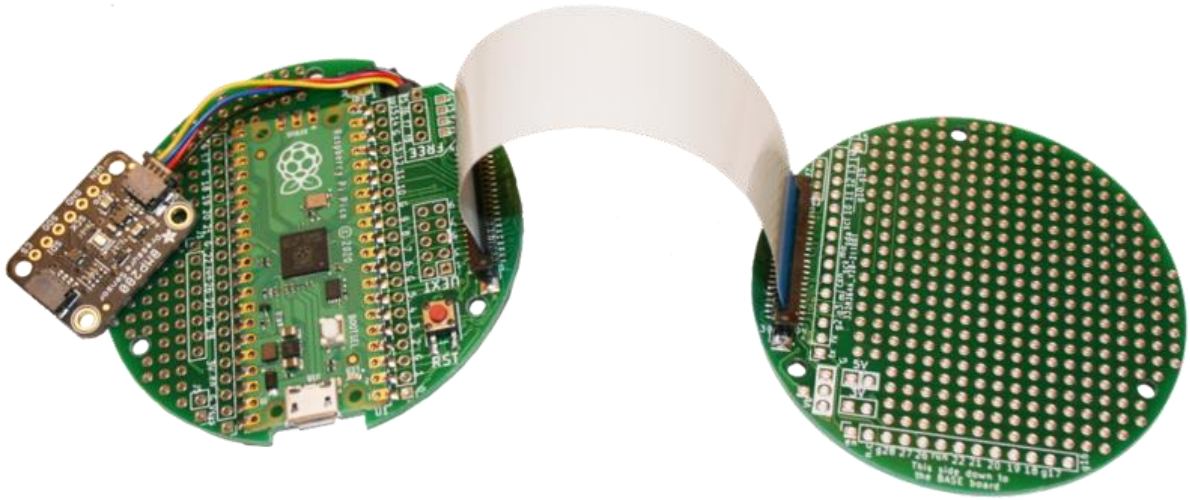
This CanSat extension board is useful to add components needed for the secondary mission like a GPS or other sensors.

https://shop.mchobby.be/fr/pico-rp2040/2272-carte-de-prototypage-cansat-pour-pico-3232100022720.html

1x FPC ribbon

This ribbon is used to connect the base board to the extension board.

https://shop.mchobby.be/en/wire-cables/2278-fpc-ribbon-40-pos-p05-1016mm-3232100022782.html



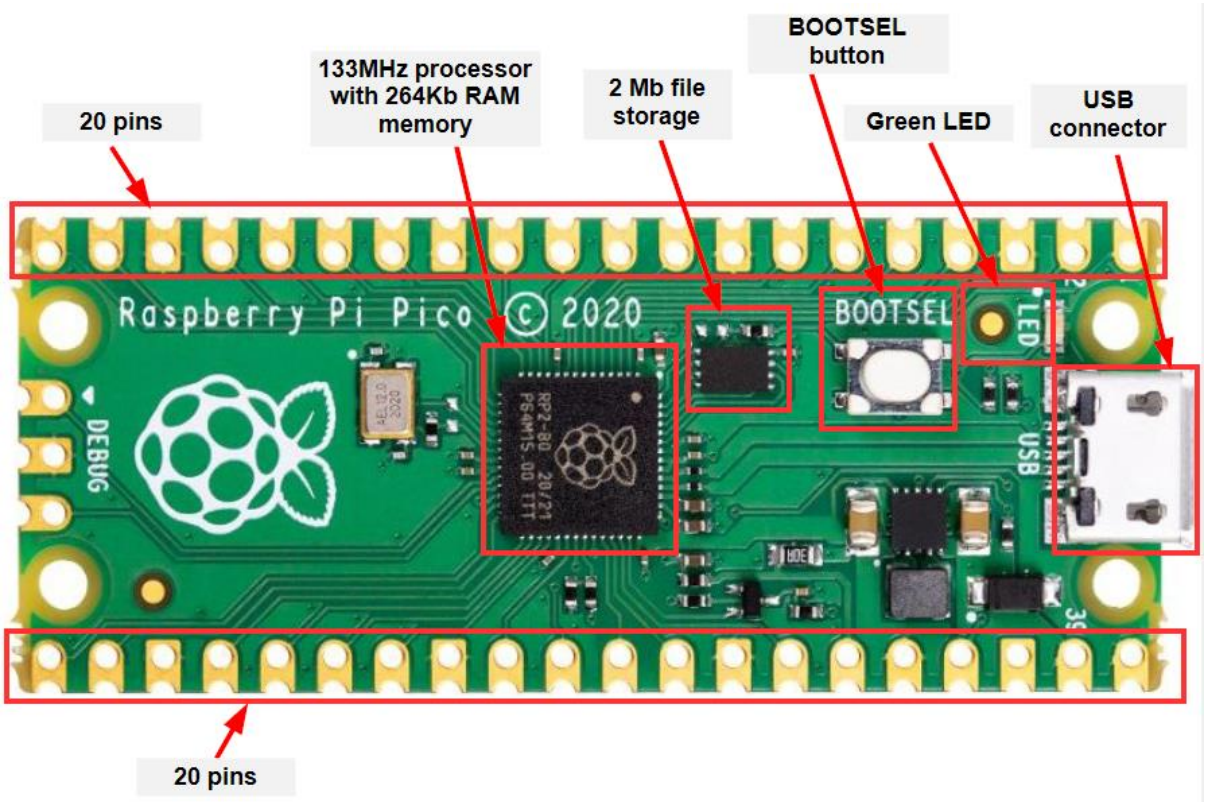
ESERO provides you with a complete kit for free but the various components can be bought separately at McHobby

Meet Raspberry Pi Pico

A Raspberry Pi Pico is a low-cost microcontroller device. Microcontrollers are tiny computers, but they only have a small file storage (unlike a hard drive on a typical computer) and lack peripheral devices that you can plug in (for example, keyboards or monitors).

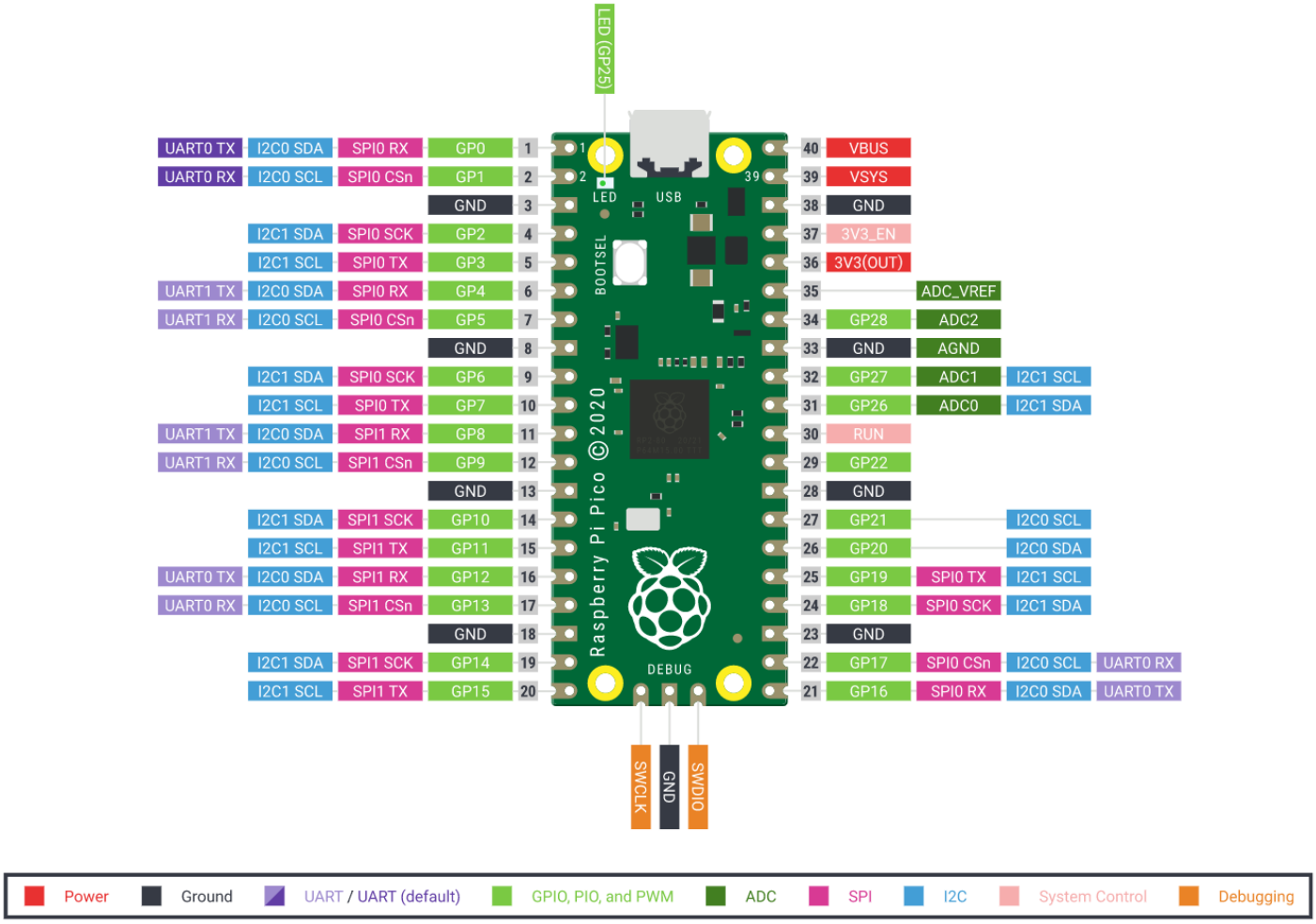
A Raspberry Pi Pico has

1- A 133MHz processor with 264 kilobytes of RAM memory
2- 2 megabytes of file storage
3- A BOOTSEL button used to install MycroPython on the Pico
4- A green LED
5- A USB connector to power the Pico and transfer software or data.
6- 2 x 20 pins used to power the Pico as well as control and receive input from a variety of electronic devices.



Each of the 40 pins has its own function.

If you need to know the pin numbers for a Raspberry Pi Pico, you can refer to the following diagram or [this interactive website]



While working with the Pico, you will only need to work with:

1- Red and black pins = pins related to power and ground connection
2- Purple pins = pins related to the [UART communication]
3- Rose pins = pins related to the [SPI communication protocol]
4- Blue pins = pins related to the [I2C communication protocol]

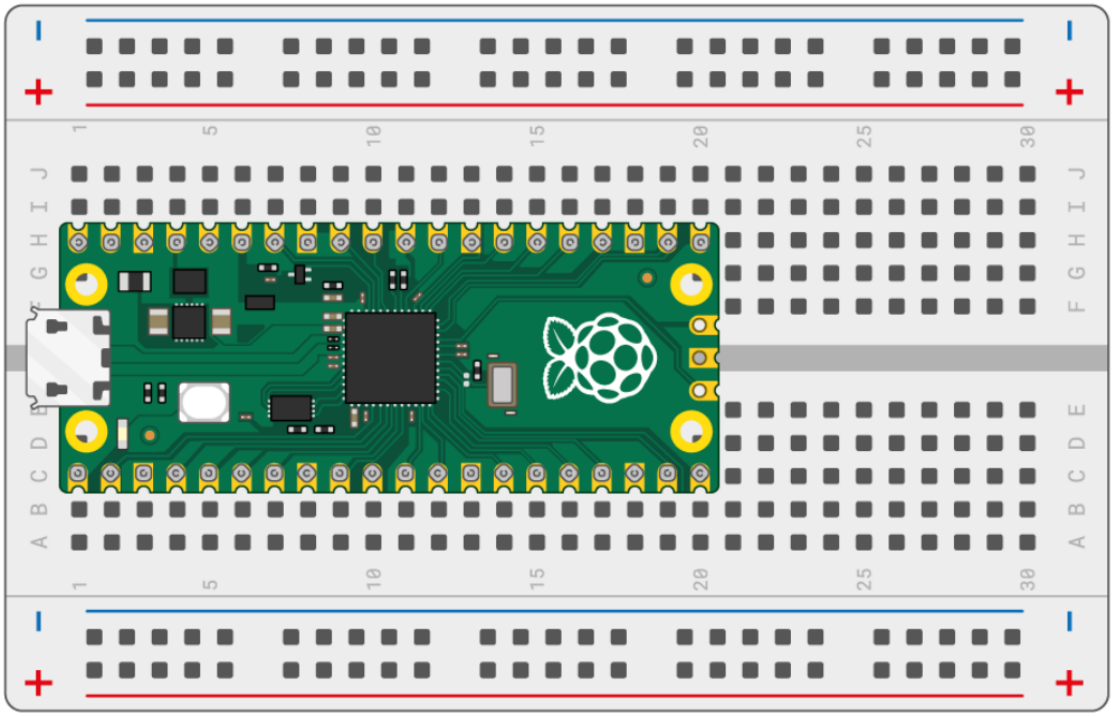
There are several ways to power your Pico, either using

1- the micro-USB cable (in the development phase)
2- the provided 5 volts converter and lithium battery (when development is done)

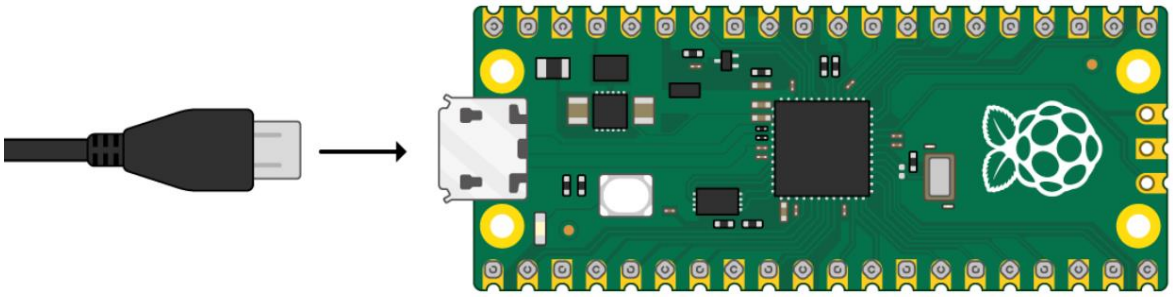
Connect your Raspberry Pi Pico to your computer

In this section, you will connect a Raspberry Pi Pico to another computer and learn how to program it using MicroPython.

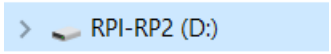
Firmly plug your Raspberry Pi Pico on the provided breadboard like shown in the following picture. Place it so that it is separated by the breadboard's ravine in the middle.



Plug the provided micro-USB cable into the port on the left-hand side of the Pico.



Your Pico should appear like an USB storage on your file system

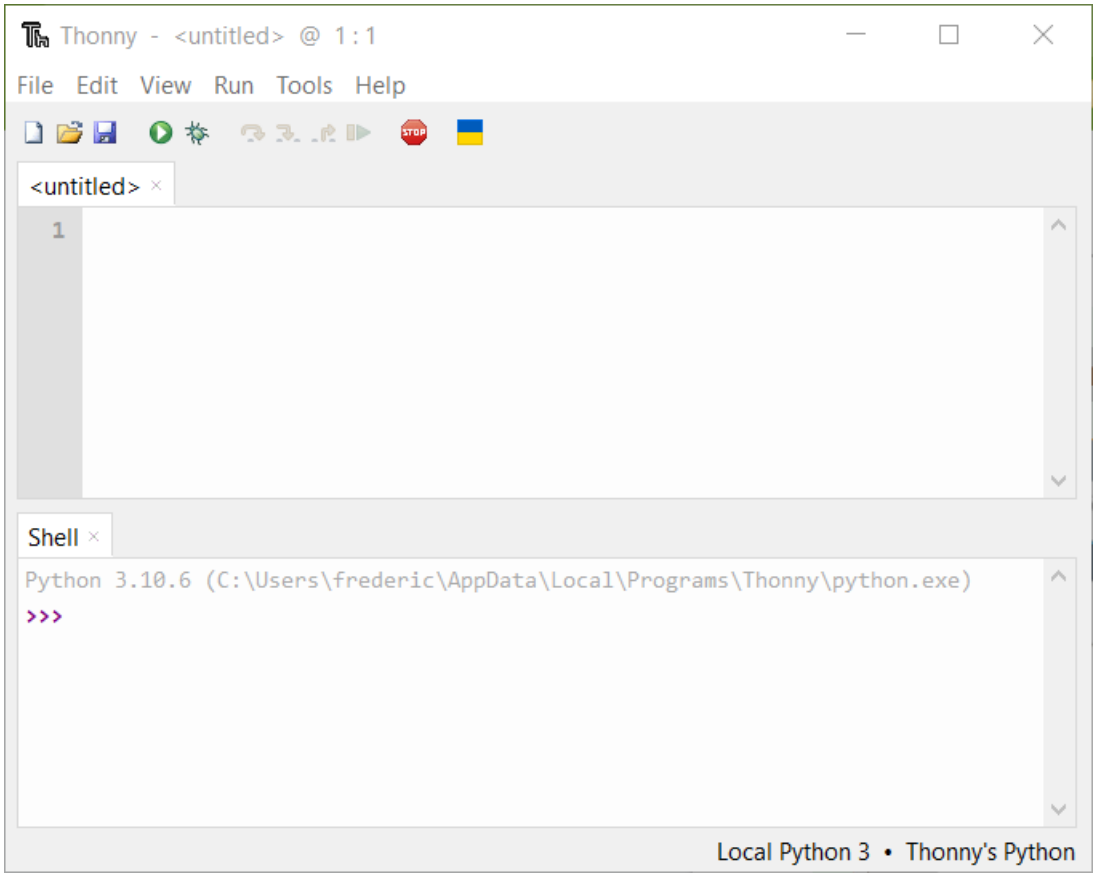


Install Thonny on your computer

Python is a general purpose language used in a large variety of applications (data sciences, Artificial Intelligence, statistics, …) while MicroPython is specifically designed for microcontrollers like the Raspberry Pi Pico used in our project.

To edit, run and debug our code in MicroPython language we will install an Integrated Development Environment (IDE) called Thonny, available at https://thonny.org

Open Thonny from your application launcher. It should look something like this:



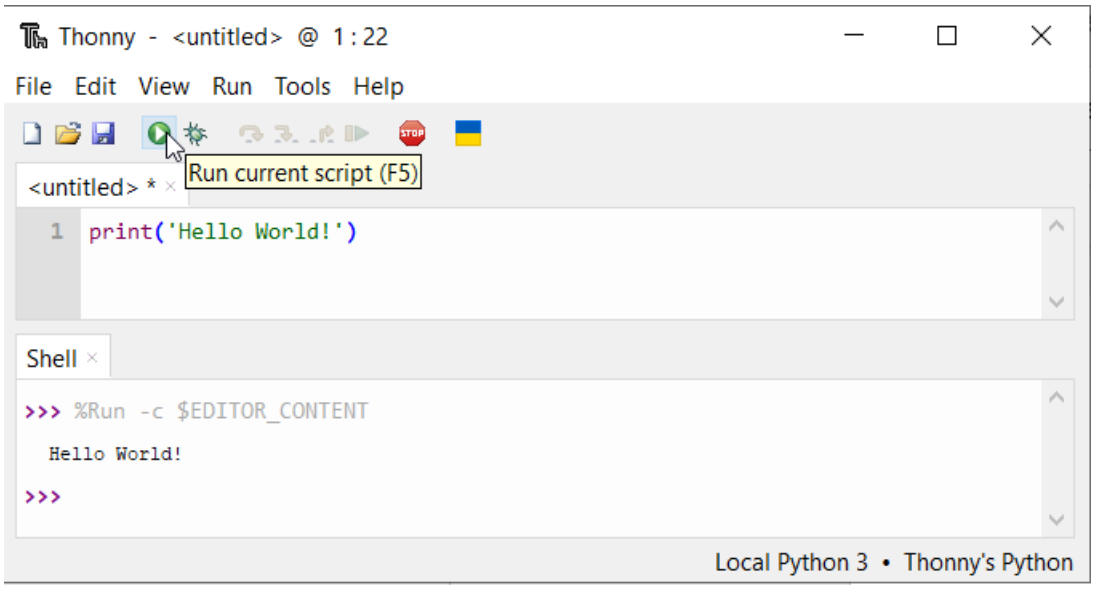
At some point we will need Thonny to be opened twice.

- Go to the menu “Tools -> Options…->General”
- Uncheck the checkbox “Allow only single Thonny instance”
- Press “OK”

You can use Thonny to write standard Python code. Type the following in the top window, and then click the Run button.

```
print('Hello World!')
```

The result is shown in the “Shell” window



Install MicroPython on your Pico

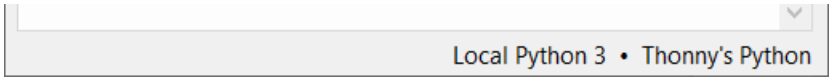
Your new Raspberry Pi Pico needs MicroPython to run your software.

MicroPython is a programming language largely compatible with Python that is optimized to run on a microcontroller like the Raspberry Pico.
Its documentation can be found on the MicroPython official website.

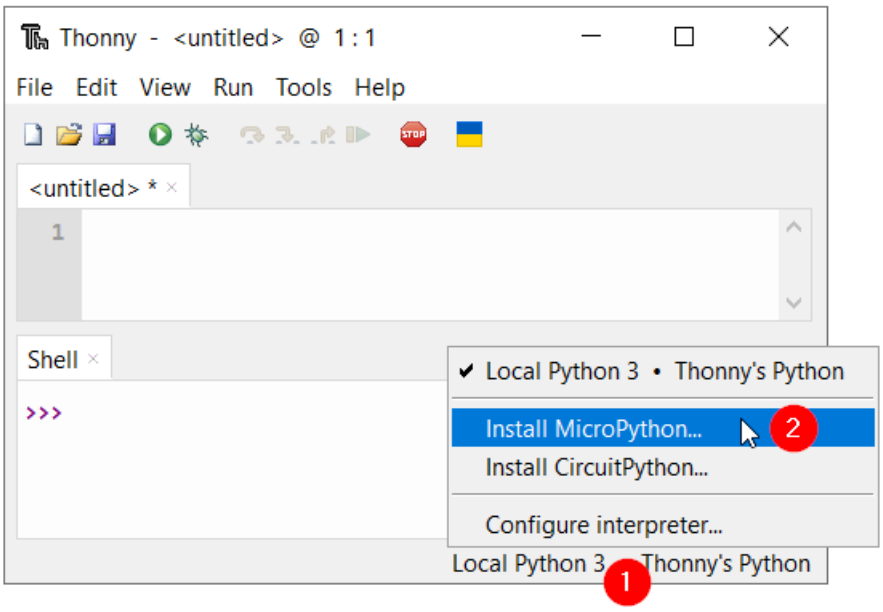
Start by unplugging the micro-USB cable from your computer but leave it connected to your Pico.

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

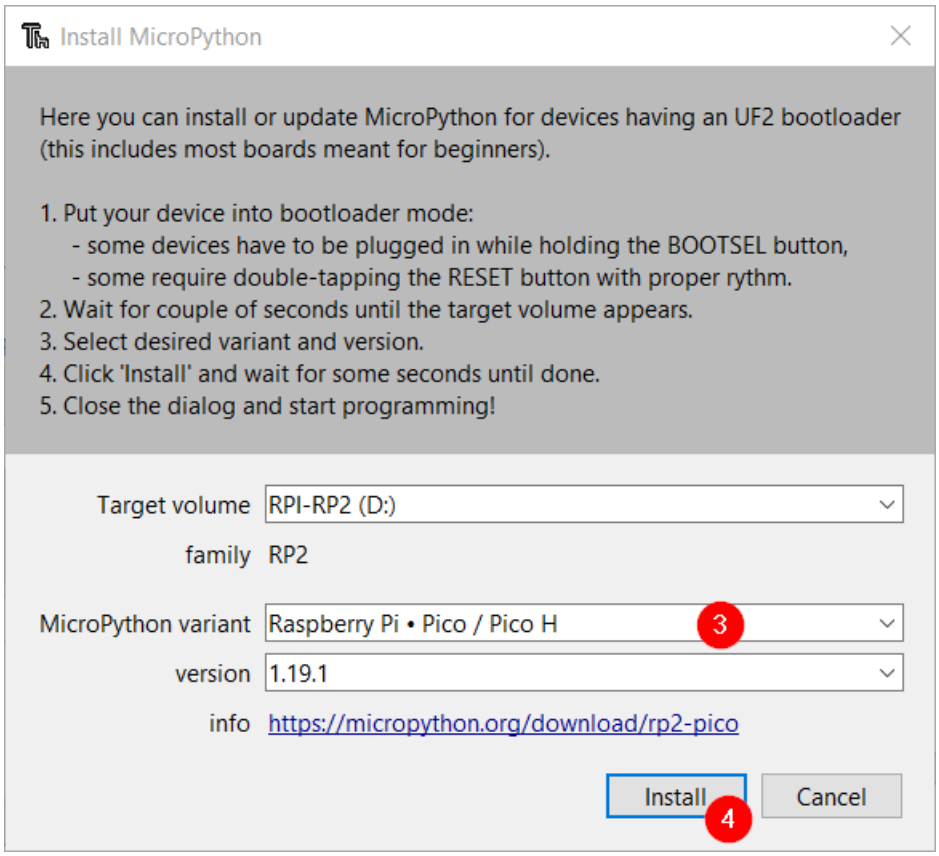
In the bottom right-hand corner of the Thonny window, you will see the version of Python that you are currently using.



Left-click on the Python version and choose ‘Install MicroPython…’



A dialog box will pop up to install the MicroPython firmware on your Pico.
Select the correct MicroPython variant and click on the Install button.



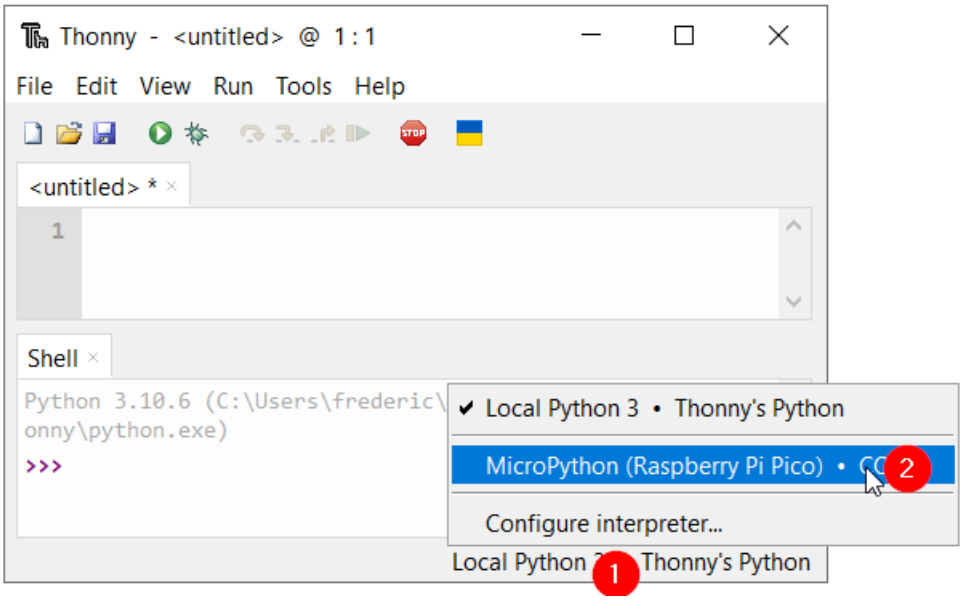
Wait for the installation to complete and click on the Close button.

You don’t need to update the firmware every time you use your Pico.

Next time, you can just plug it into your computer without pressing the BOOTSEL button.

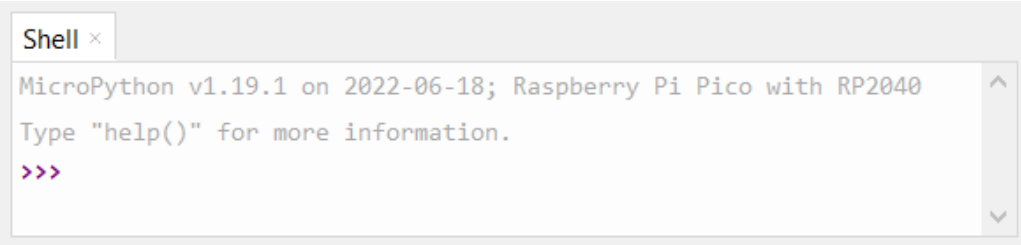
Run your first MicroPython code on your Pico

Make sure that your Raspberry Pi Pico is still connected to your computer.
Select the MicroPython (Raspberry Pi Pico) interpreter on the bottom right.



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 Pico using the REPL (read–eval–print loop), which allows you to type Python code into the Shell and directly see the result.

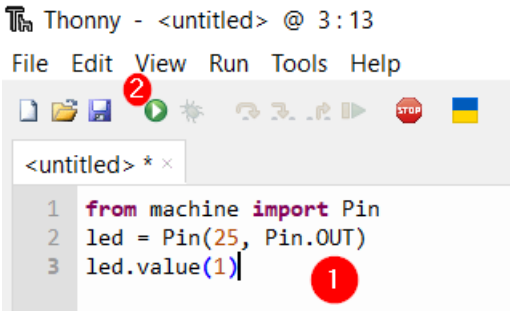
MicroPython adds hardware-specific modules, such as `machine`, that you can use to program your Raspberry Pi Pico.

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

If you set the value of the LED to `1`, the onboard LED 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)
```



After pressing the green “Run” button , you should see the onboard LED light up.

Change the code and set the LED value to 0 to turn the LED off.

```
led.value(0)
```

Turn the LED on and off as many times as you like.

Tip: You can use the up arrow on the keyboard to quickly access previous lines.

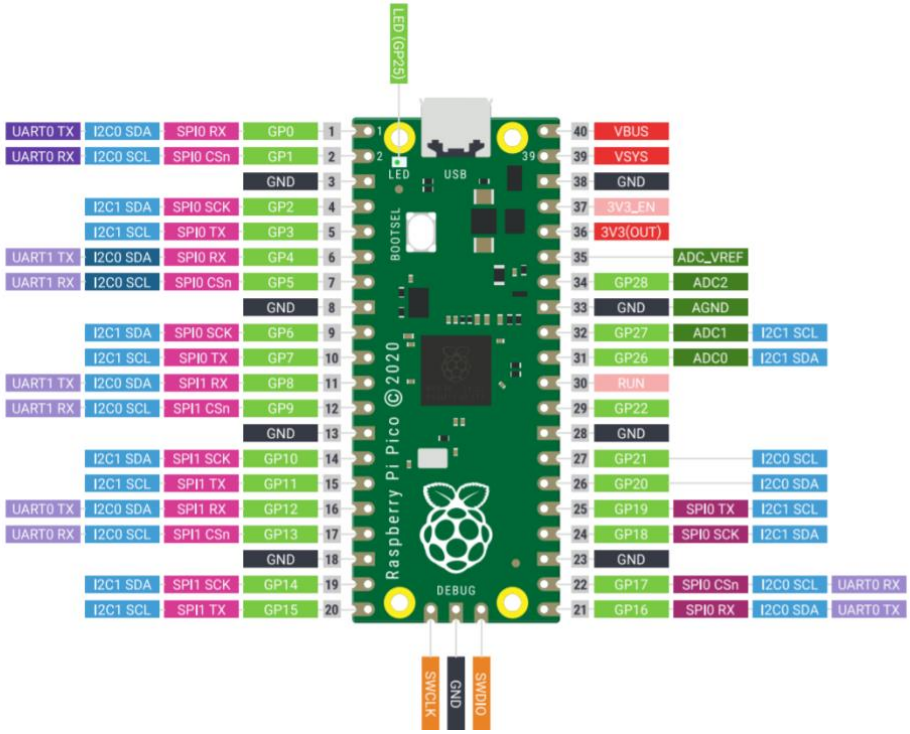
We said the onboard LEP is connected to GPIO pin 25, but what is GPIO?

General Purpose Input/Output (GPIO)

GPIO pins on the Raspberry Pi allow external voltages to be read from the software and they also allow external voltages to be set from software. GPIOs can be either digital or analog. Analog pins can be set as a digital pin but digital pins remain digital. Analog pins are only input pins.
For our 3.3V Raspberry Pi, any voltage under 2.5V is interpreted as "False" and conversely any voltage over 2.5V is interpreted as true (up to 3.3V). This is similar for output signals. A "True" output will set the pin's voltage to 3.3V and the "False" output will set the pin's voltage to 0V.

GPIO pins can be used as either input or output ports and this set by software.

The Pi Pico has 28 GPIO ports as seen in the green boxes in the following diagram. Many pins are multi-purpose and can also be used for other interfaces (UART, SPI, I2C), these are represented by the multi-coloured boxes to the side of the green boxes in the diagram. The following link contains the pinout: https://datasheets.raspberrypi.org/pico/Pico-R3-A4-Pinout.pdf



https://datasheets.raspberrypi.org/pico/Pico-R3-A4-Pinout.pdf

Typical GPIO CanSat Uses:
Inputs: On/Off based sensors, switches, buttons, deployment sensors
Outputs: Status LEDs, basic servos (PWM is better), turning sensors on, resetting sensors connected by other signals

If you want to write a longer program, then it is best to save it in a file. You will do this in the next section.

Write your first microPython file to control the onboard LED

The Shell is useful to try out quick commands. However, it is better to put longer programs in a file.

Thonny can save and run MicroPython programs directly on your Raspberry Pi Pico.

In this step, you will create a MicroPython program to blink the onboard LED on and off in a loop, using GPIO pins

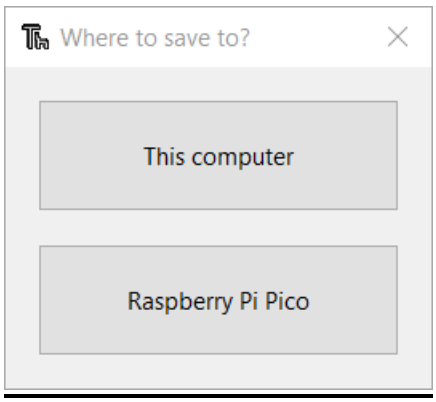
Go back to Thonny and click in the main editor pane of.

Enter the following code to toggle the LED.

```
# import necessary pre-existing libraries
from machine import Pin
# Declare a variable named “led”, link it to pin number 25 and define it as output
led = Pin(25, Pin.OUT)
# Change the led state from led.value(0) to led.value(1) and vice versa
led.toggle()
```

The complete “Pin” library documentation can be found in the official MicroPython documentation

Click the **Save** button to save your code and the following screen will show up:



Choose “Raspberry Pi Pico” and name the file “blink.py”

Tip: You need to enter the .py file extension so that Thonny recognises the file as a Python file. Thonny can save your program to your Raspberry Pi Pico and run it.

You should see the onboard LED switch between on and off each time you click the Run button.

But what if you want to see the LED blinking without having to click the Run button over and over ?

To achieve this we will use a “while” loop and the “sleep” function

Be careful to indent the code with 4 spaces within the while loop to let MicroPython know that these lines are part of the while loop.

Update your code so it looks like this:

```
# import necessary pre-existing libraries
from machine import Pin
from time import sleep
# Declare a variable named “led”, link it to pin number 25 and define it as output
led = Pin(25, Pin.OUT)

# Be careful to indent the code with 4 spaces within the while loop to let
MicroPython know which lines are part of the while loop.
while True:
    # Change the led state from led.value(0) to led.value(1) and vice versa
    led.toggle()
    # Halt the program execution for half a second
    sleep(0.5)
```

You can also use the Timer module (first line below) to set a timer that runs a function at regular intervals. Update your code so it looks like this:

```
# import necessary pre-existing libraries
from machine import Pin, Timer

# Declare a variable named “led”, link it to pin number 25 and define it as output
led = Pin(25, Pin.OUT)
# Declare a timer variable to deal with timing of periods and events
timer = Timer()

# Declare a function named “blink” that toggle the LED state
def blink(timer):
    # # Change the led state from led.value(0) to led.value(1) and vice versa
    led.toggle()

# Configure the timer to call the pre-defined blink function every 2.5 seconds
timer.init(freq=2.5, mode=Timer.PERIODIC, callback=blink)
```

The complete “Timer” library documentation can be found in the official MicroPython documentation

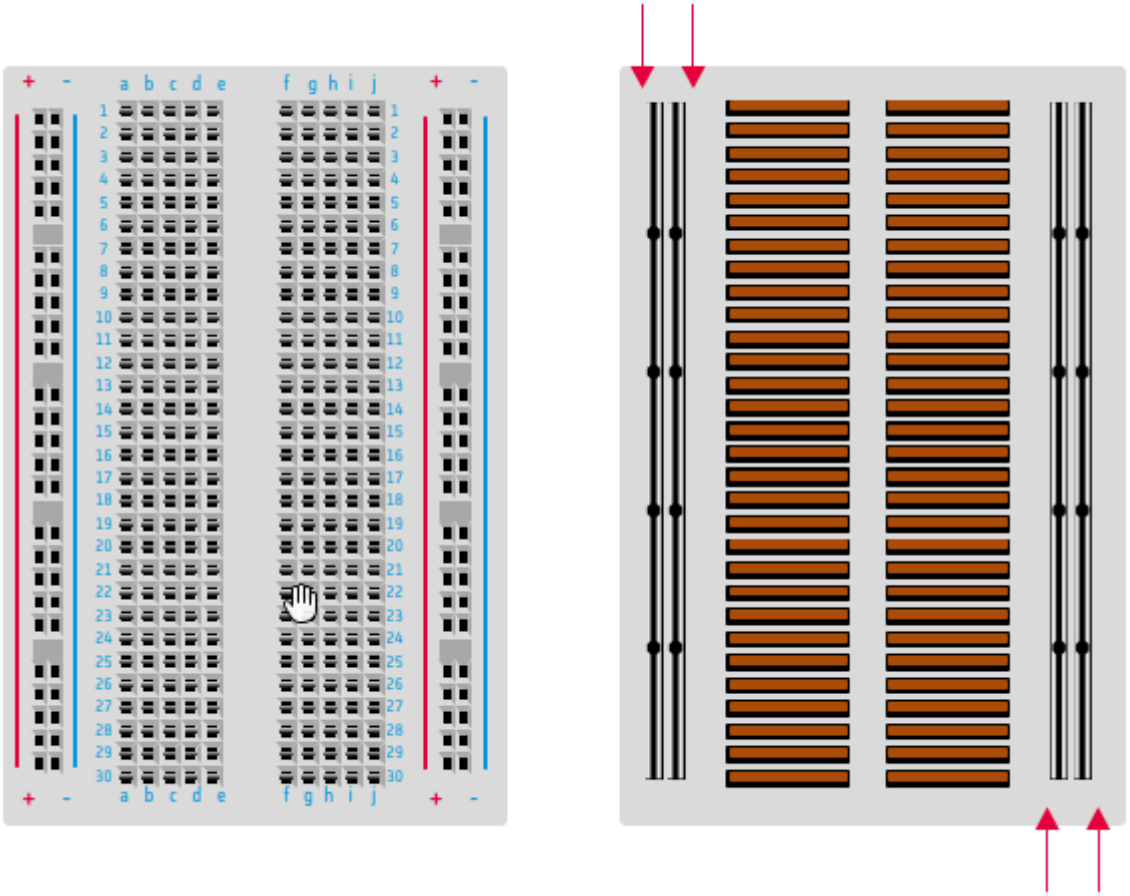
Click **Run** and your program will blink the LED on and off until you click the **Stop** button.

Tip: If you unplug/re-plug your Pico, you will have to press again the **Run** button to run the program. But if you name your program “**main.py**” on your Pico it will run automatically when the Pico powers up.

Software development & wiring test

Meet a breadboard

Whilst you are learning the basics of Pico and sensors it is best to use a solderless breadboard, as any mistakes you make building your circuit can be easily changed.
A breadboard is a simple tool that can be used to wire electrical components together.



Pins of electrical components can be placed into the terminals on the board. Centrally, rows are horizontally connected. This means for example, that the two pins of a resistor should be placed in different rows, otherwise it will form a closed circuit with itself.

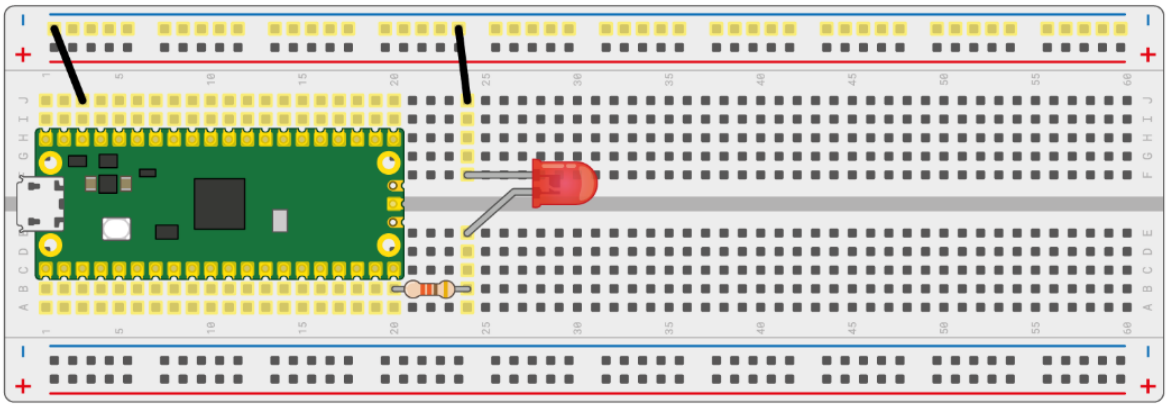
It is very important to make a sketch of your circuit before connecting and powering the circuit, because you will risk breaking the components. The outer columns of the board are connected in columns, rather than rows. Typically, these are used to provide ground and voltage connections.

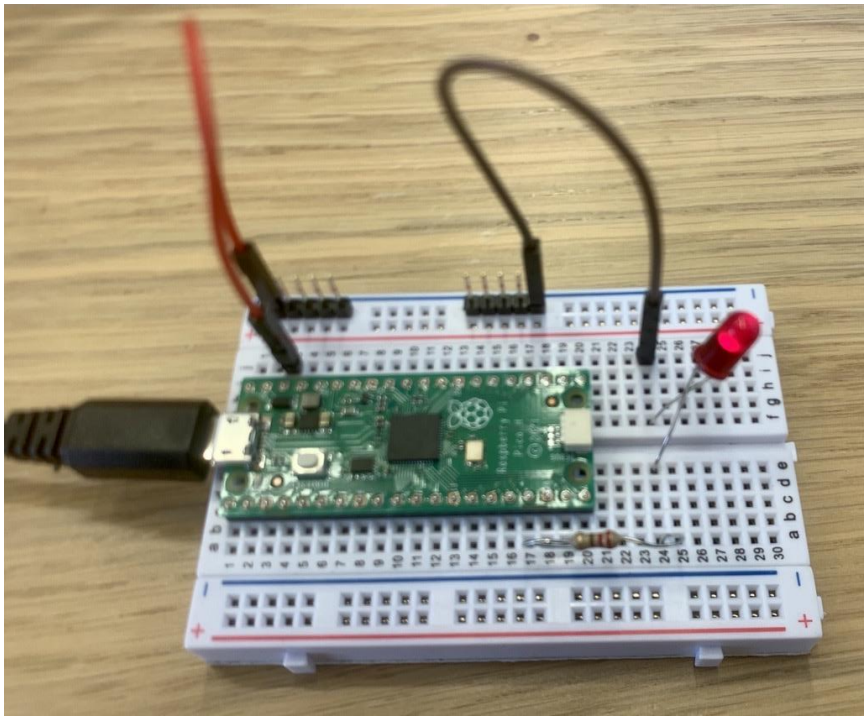
Blink a LED on the breadboard

You will know learn how to control an external LED.

Use a 220 ohms resistor, an LED, some female jumpers and a few headers to connect up your Raspberry Pi Pico on your breadboard as shown in the image below.

Note how the LED is connected on GPIO 15 on one side (the last one on the bottom left as you can see in the Pico pinout diagram) and to the Pico's ground pin on the other side.





In Thonny, reuse the code from the previous section, but instead of GPIO 25, use GPIO 15

```
….
# Declare a variable named “led”, link it to pin number 15 and define it as output
led = Pin(15, Pin.OUT)
….
```

In this example, we chose to connect the LED to GPIO 15 but you can use another GPIO if you want.

Test your jumpers

Your kit contains several male and female jumpers to test your components wirings on your breadboard before soldering your components on the CanSat base board.

Sometimes, because of factory issues, it happens that some if these jumpers are broken.

To save you some headaches and time, we strongly advise you to test all your jumpers with the previous “blink an external LED” exercise by replacing the 2 jumpers with the other jumpers provided with the kit.

Alternatively you can also test the jumpers with the continuity tester of a multimeter device

Measuring temperature and pressure

The BMP280 chip provided with the kit measures the atmospheric pressure as well as the temperature.

Install the BMP280 python library

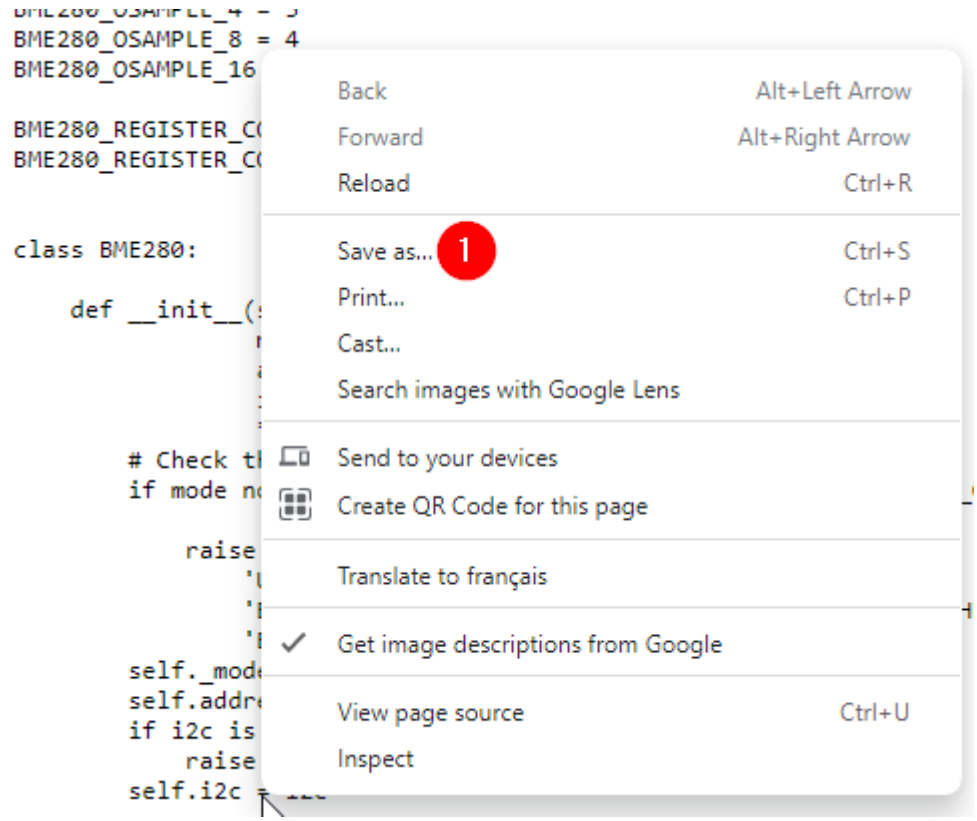
Micro Python provides some built-in functionality for managing the Pi Pico; however this can be extended through the use of third-party libraries. These are libraries produced by manufacturers, suppliers, and the Micro Python community for the purpose of using extra devices with the Pi Pico. These libraries reduce the complexity of using external devices by providing high-level functions to interact with the devices they support

The kit is made of several sensors for which we will use specific Micro Python libraries we can find on the internet.

To interact with the BMP280 we need a dedicated python library that can be downloaded via this link:

https://raw.githubusercontent.com/mchobby/esp8266-upy/master/bme280-bmp280/bme280.py

Right-click on the file, and choose “Save as…” as shown in the screenshot below:

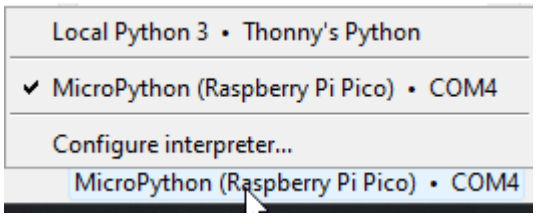


Save the file on your PC and keep the name “bme280.py” in lowercase.

Using Thonny, the library must then be transferred from your PC to the Raspberry Pi Pico /lib directory.

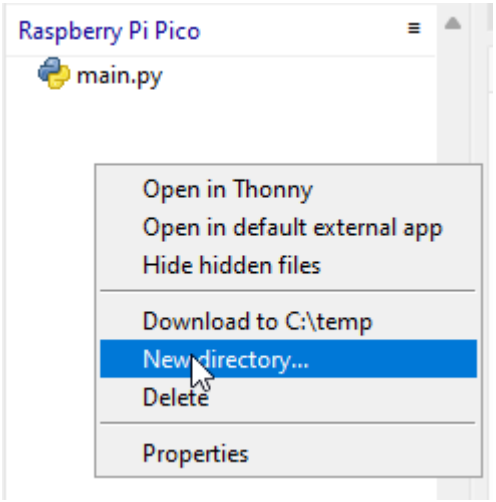
Ensure the Pico is connected to your PC and turned on.

In Thonny, ensure you are connected to the Pico by clicking on the bottom right menu

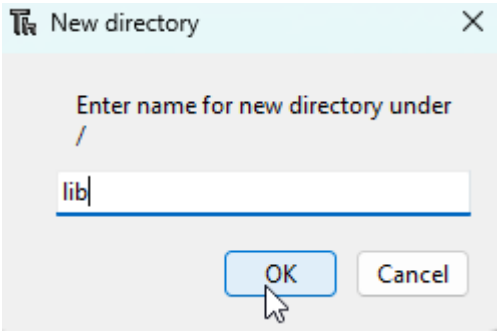


In the “View” menu, ensure the “Files” option is checked.

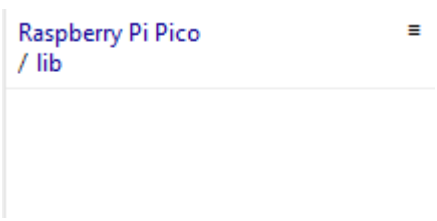
Then in the “Raspberry Pi Pico” window on the bottom right, right click and select “new directory…”.



Enter “lib” (**in lowercase**) and press ok

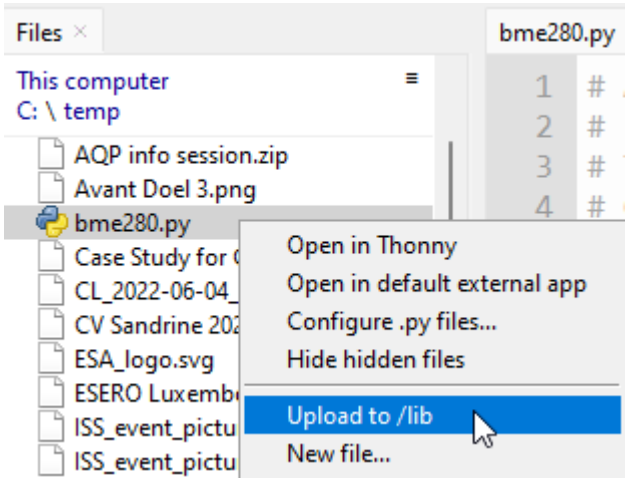


Then double-click on the “lib” directory which is empty for the moment



In the top left “Files” window, browse to the location where you saved the bme280.py file.

Right-click on the bme280.py file and press “upload to /lib” to install the library on the Pico.

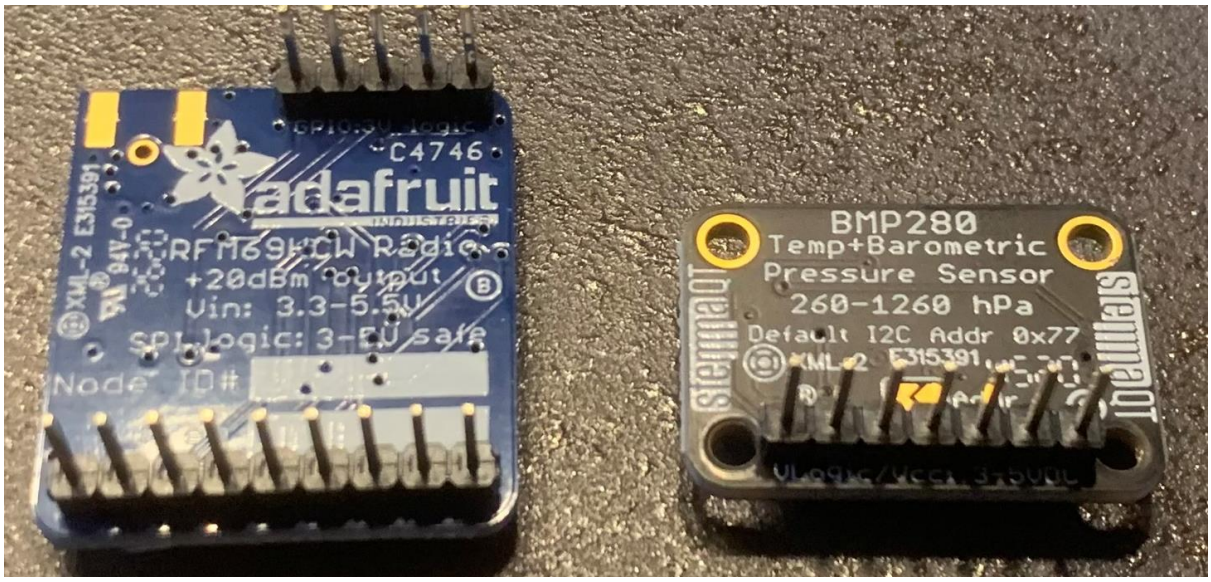


Soldering the components pins

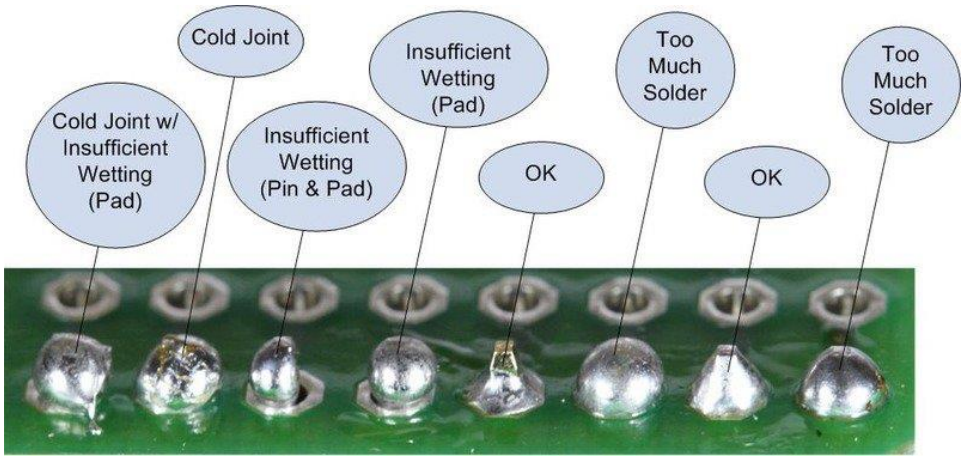
WARNING: Please first carefully read the guide to Excellent Soldering from Adafruit

The BMP280 as well as the RFM96W radio module for the ground station need header pins soldered to them so that they can be connected to the breadboards.

The pins of the BMP280 board and the RFM96W radio module that will be used for the ground station need to be soldered with header pins.



For the second RFM96W radio module, solder **female jumpers cables** on its pins, **not headers**, so that you can easily connect the module to the breadboard as well as on the CanSat base board later on.



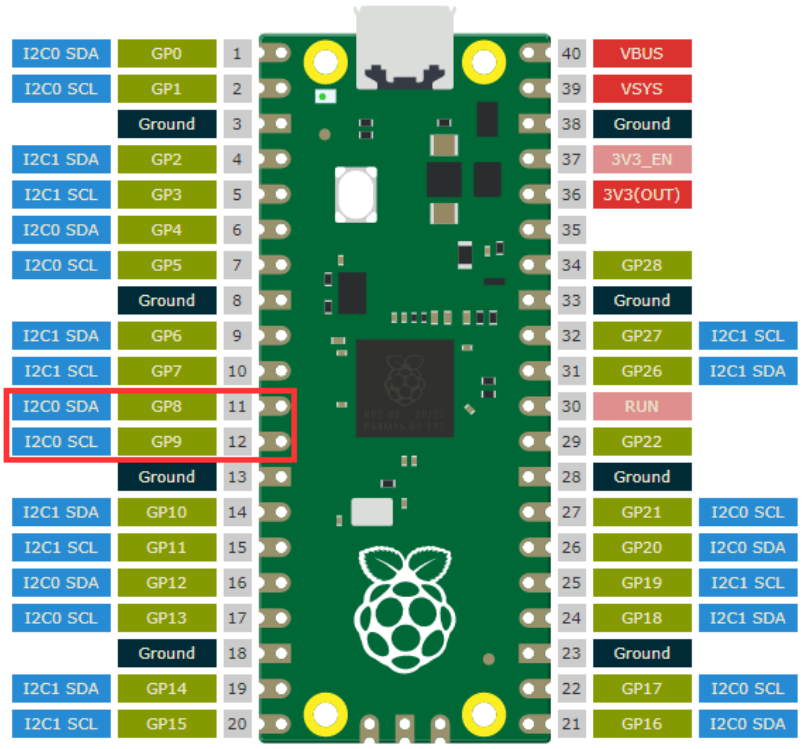
https://learn.adafruit.com/adafruit-guide-excellent-soldering/common-problems

Connecting the BMP280 Pressure/Temperature Sensor using I2C

Before we can read data from the sensor, we need to connect it to the Pi.
To do that, we first need to solder headers to its pin as explained on page 8 of the BMP280 documentation.

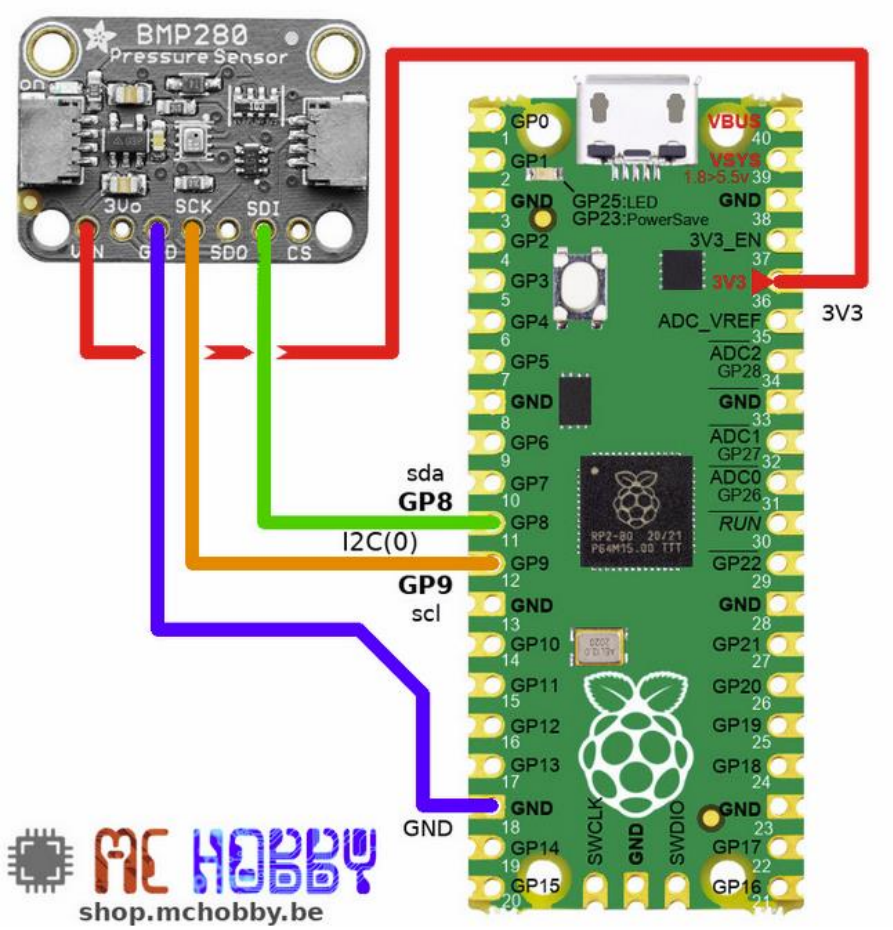
The documentation explains we need a 3.3V power supply to power it. We can use the Pico 3V3 and GND (ground) pins for that. The documentation also explains that we can connect to it using the I2C bus. I2C requires us to connect two data cables and, thus we need to connect four cables in total.

1. Using the interactive pinout website, locate the 3V3 power pin on the Pico
2. Still using the interactive pinout website, locate I2C (0) pins on the Pico (see below)



3. Connect the 2-6V input pin on the BMP280 to the pin 3V3 on the Pi (3.3 volts 300 mA output)
4. Connect the GND pin on the BMP280 to a GND pin on the Pi.
5. Connect the I2C 0 SDA (pin 11) to the BMP280 SDI pin.
6. Connect the I2C 0 SCL (pin 11) to the BMP280 SCK pin.

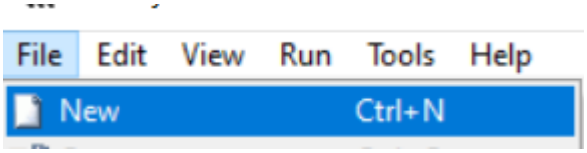
See the connection diagram below:



Reading the Temperature and Pressure

Now that the BMP280 is connected and set up we can read data from it.

1. Create a new file



2. Save this file under the name you want, **terminated by .py**

3. Fill the file with the following code

```
from machine import I2C
# The BME280 library also work for the BMP280 sensor
from bme280 import BME280, BMP280_I2CADDR
from time import sleep
# Initiate a new I2C connector on bus 0, sda=GP8, scl=GP9 @ 400 KHz
(default)
i2c = I2C(0)
# Create a new BME280 variable connected to i2c bus 0 communicating with
address BMP280_I2CADDR
bmp = BME280( i2c=i2c, address=BMP280_I2CADDR )
while True:
    # print a tuple with (temperature, pressure and humidity)
    print( bmp.raw_values )
    sleep(1)
```

4. Pressing the “Run” button should print the temperature, pressure, and the humidity every second

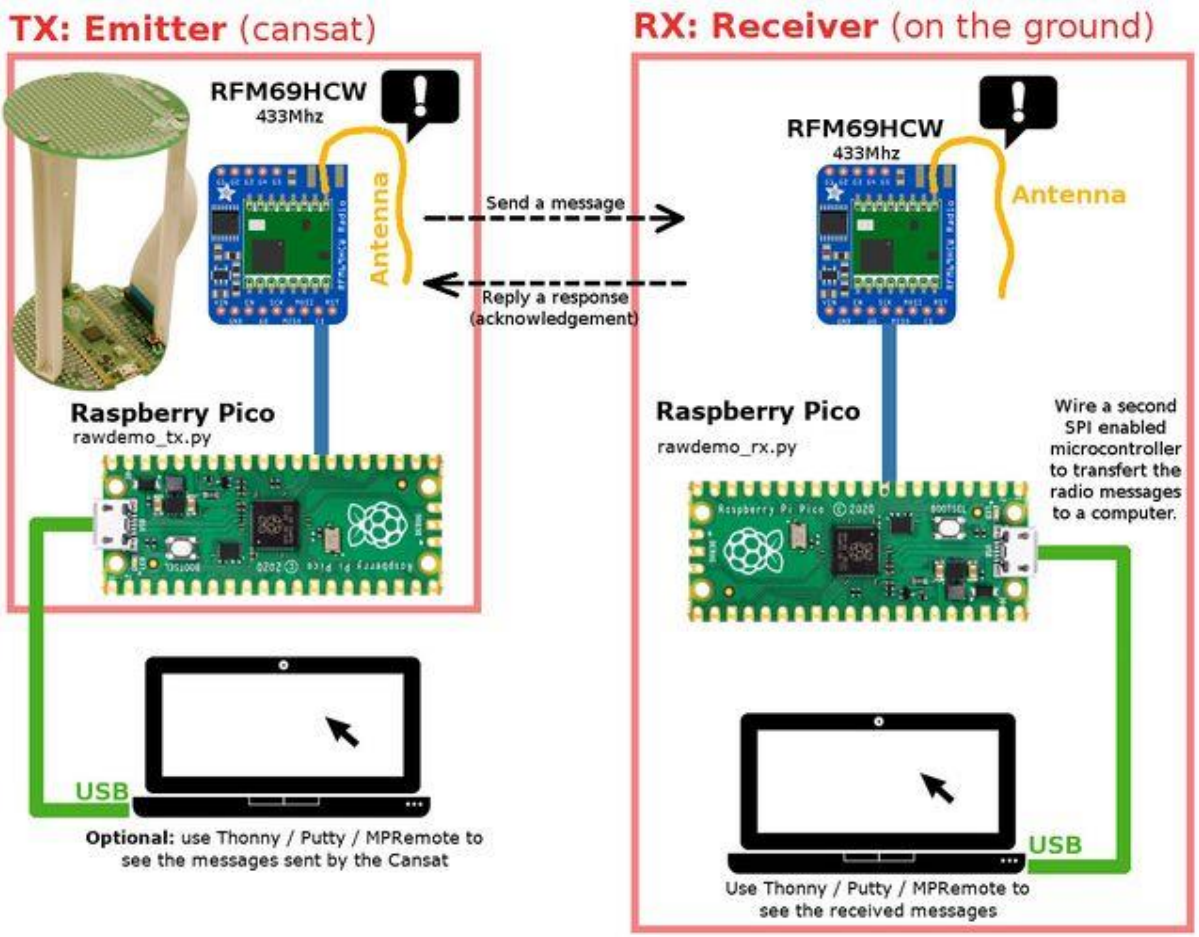
```
(22.28, 1017.68, 0.0)
(22.27, 1017.66, 0.0)
(21.87, 1017.67, 0.0)
(21.83, 1017.73, 0.0)
(21.83, 1017.68, 0.0)
(21.81, 1017.68, 0.0)
(21.81, 1017.68, 0.0)
```

The altitude can be computed using the air pressure as explained in this document.

Receiving radio data

The kit contains 2 Pico microcontrollers and 2 RFM69HCW modules to create a "Data Emitter" (the CanSat) as well as the “Data Receiver” (the ground station).

In this section, we will setup the ground station (on the right below)



A successful communication requires on both sides:

- Identical frequencies (eg: 433.1 MHz).
- Identical encryption keys.
- Well-designed antennas, except during tests on breadboards where the 2 RFM69HCW are really close from each other.

Install the RFM69HCW python library

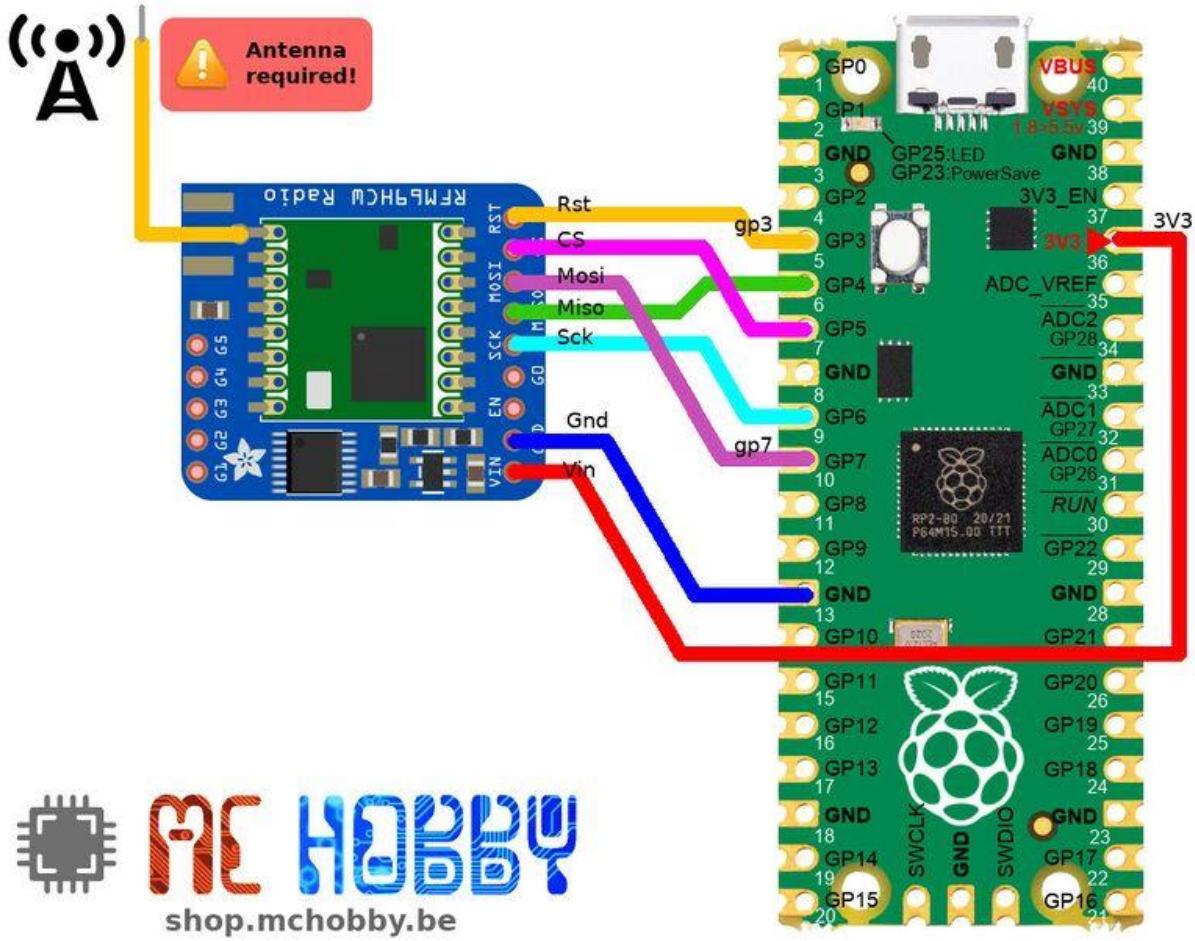
Using Thonny, follow the same process than with the installation of the BMP280 library to upload the following rfm69.py file into the /lib folder of your 2 Pico.

https://raw.githubusercontent.com/mchobby/esp8266-upy/master/rfm69/lib/rfm69.py

Connecting the RFM69HCW using SPI

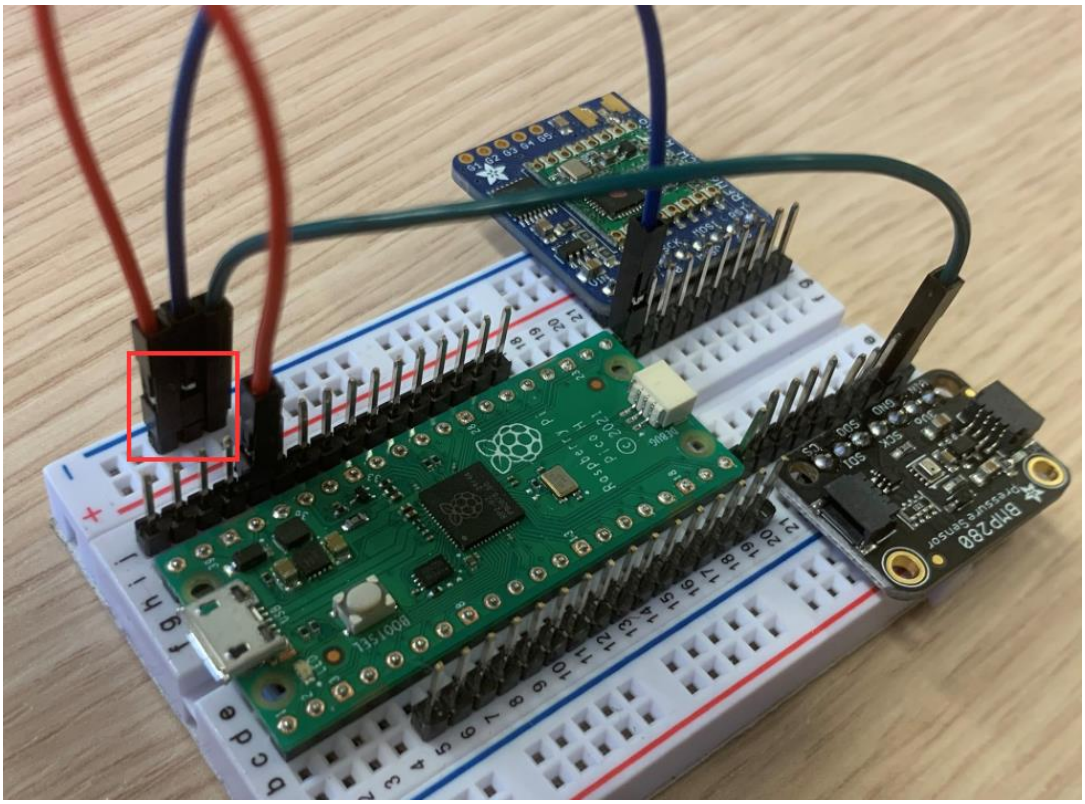
Using headers and jumper cables, we need to connect the ground station Pico and the RFM69HCW together on a breadboard

The RFM69HCW documentation explains that we can connect to it using the SPI bus. SPI requires us to connect 4 data cables, a reset connection plus 2 power cables and, thus we need to connect seven cables in total.

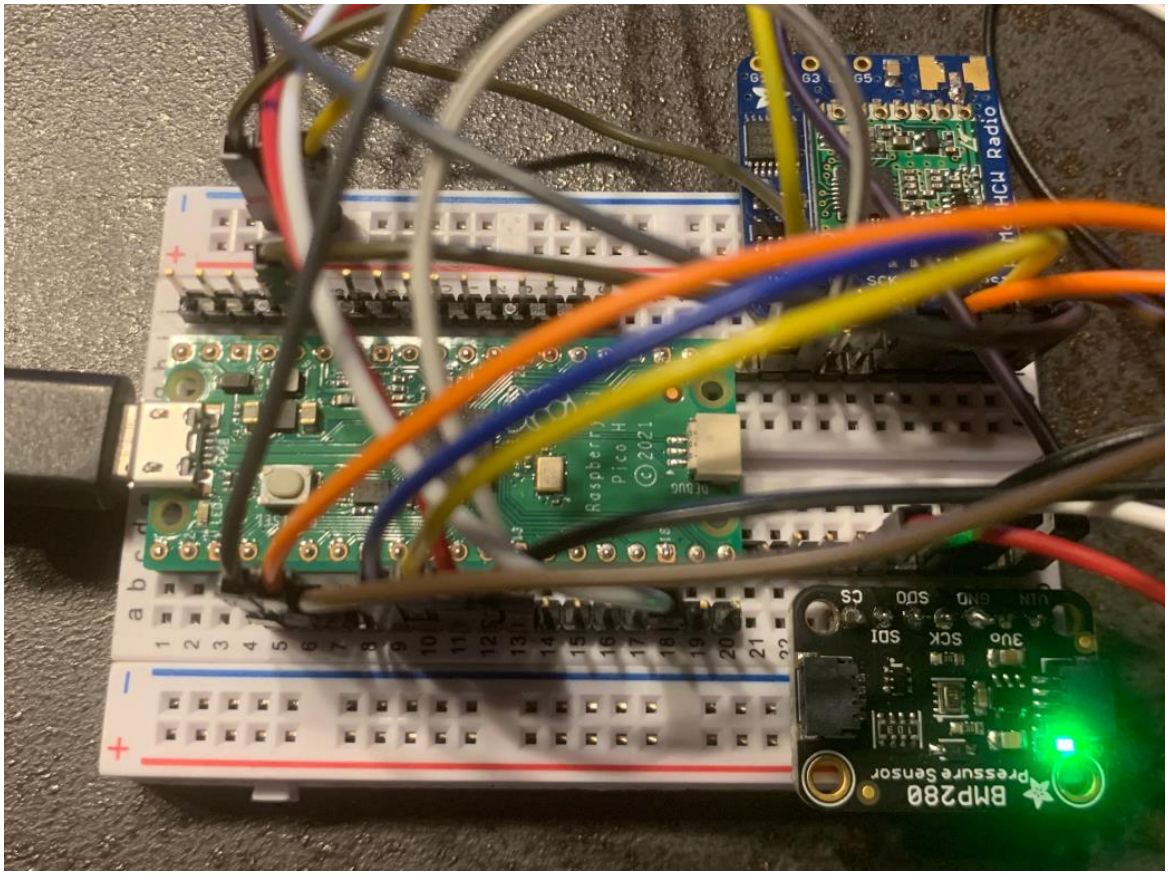


RFM69HCW	PICO
RST	GP3
CS	GP5 (Slave Select)
MOSI	GP7 (Miso)
MISO	GP4 (Mosi)
SCK	GP6 (Clock)
GND	GND
VIN	3V3

Note that since the 3V3 power pin is already used to power the BMP280, you can share it to power the radio module like show in red below



After the radio wiring the breadboard should look like this



Waiting for radio data

1- Download and open the receiver script in Thonny

2- Update the frequency and the encryption key on lines 17 and 30

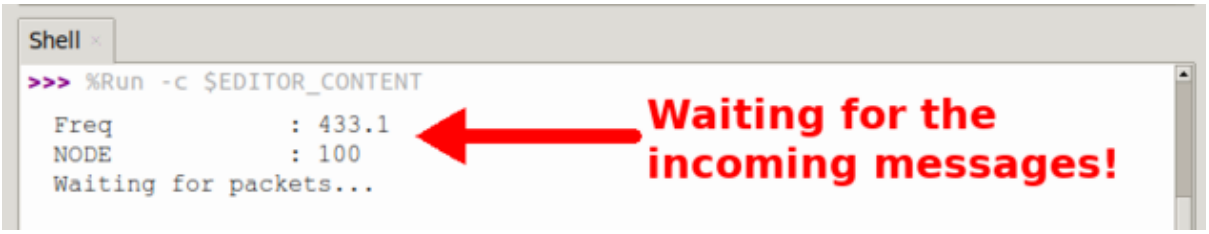
   a. The frequency is defined on line 17, comprised between 430 and 440Mhz,

   b. The encryption key can be defined on line 30 using the following code

```
rfm.encryption_key = bytes( [1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8] )
```

   Each team will receive its own frequency and encryption key before the rocket launch.

3- Run the script



Sending radio data

In this section, we will build the data emitter that will be onboard your CanSat.

Leave Thonny open with the receiver script running and open a second time Thonny.

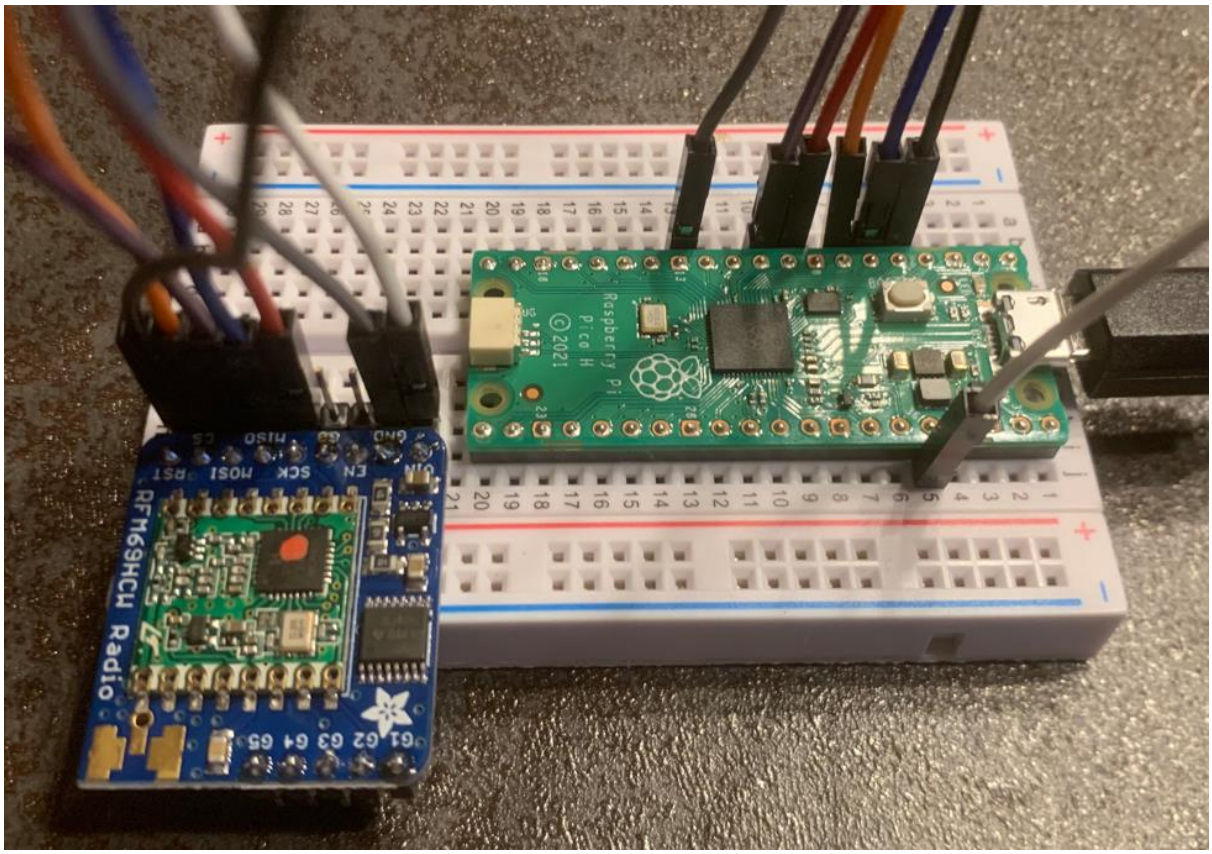
Using headers and jumper cables, we need to connect the CanSat Pico and the RFM69HCW together on another breadboard

On the second Pico:

1- Install MicroPython

2- Install the RFM69HCW python library if not already done (see previous section)

3- Connect the RFM69HCW using SPI (see previous section)

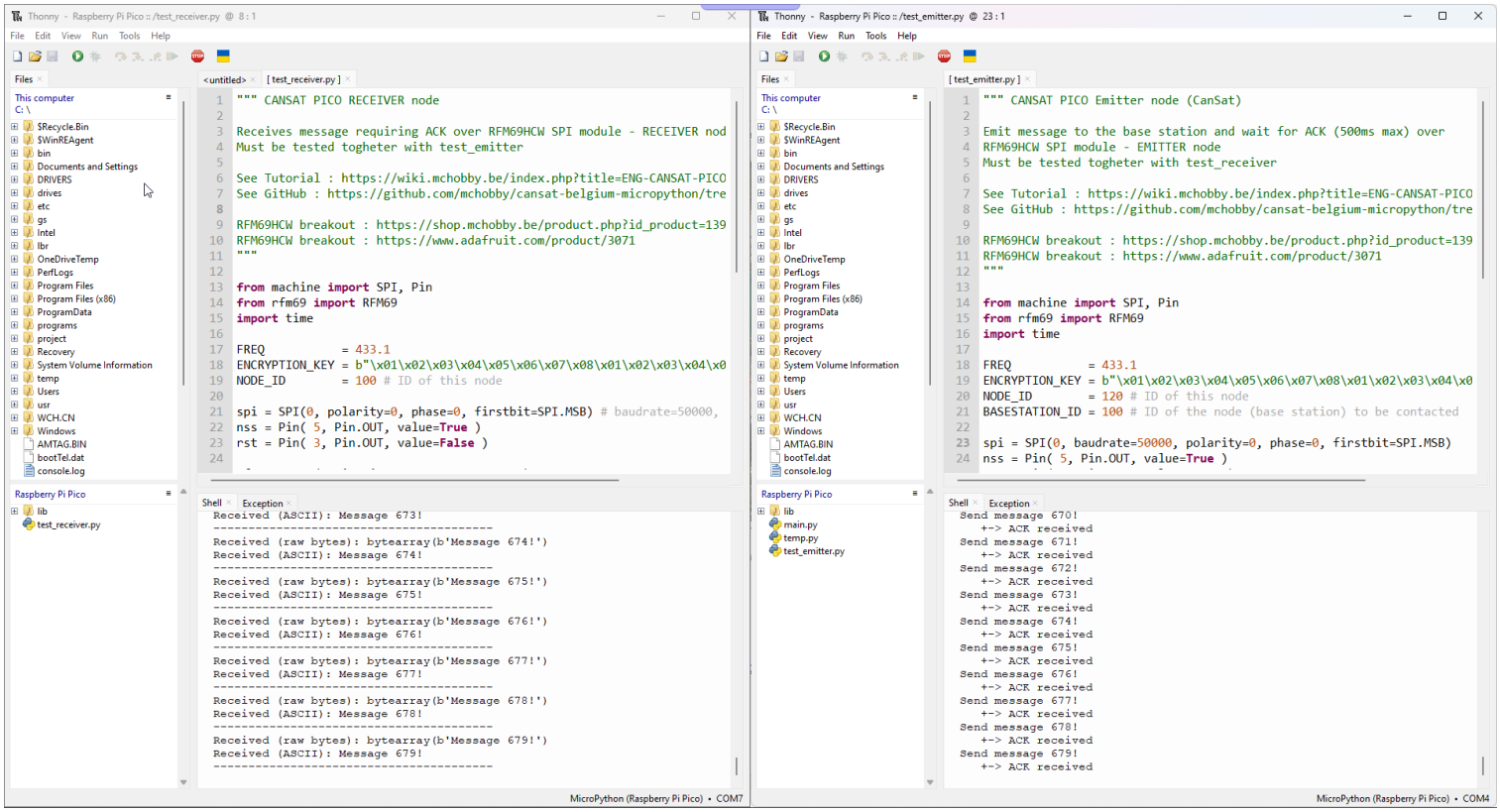


4- Download and open the sender script in Thonny

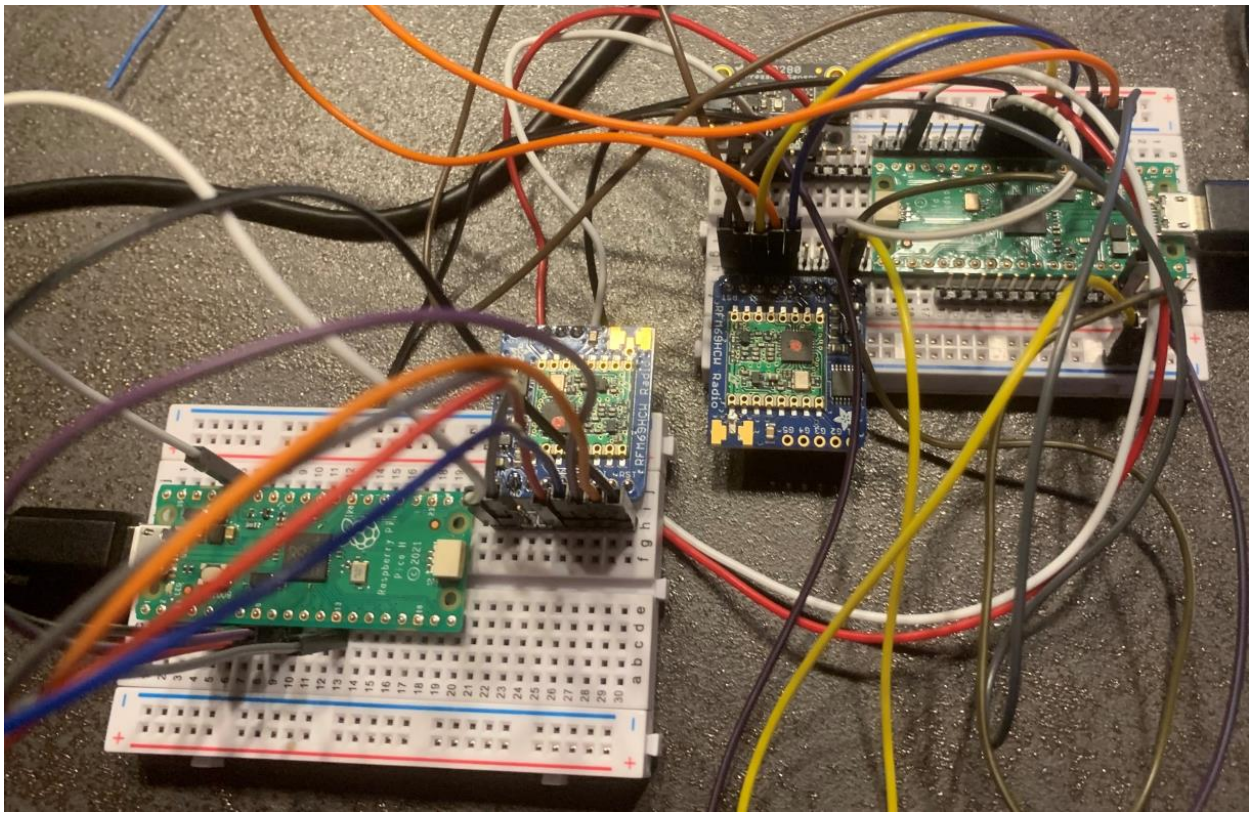
5- Update the frequency and the encryption key on lines 18 and 32 to the same values than the ground station

6- Upload the script to the Pico

7- Run the script



The 2 radio transmitting data to each other.
They must not be more than 5cm away from each other.



More detailed information can be found on [McHobby website]

Assembling your CanSat

When you have tested all your components, on the breadboards

The McHobby CanSat wiki provides all the necessary information to assemble the components on the Cansat base and extension boards provided with the kit.

The assembly steps for the primary mission are the following:

1- Solder a Pico to the CanSat base

   IMPORTANT: in the kit we gave you Pico boards with pre-soldered header which you can also use to solder on the CanSat base

2- Solder the PowerBoost 500 component to the CanSat base

3- Connect the BMP280 to the Qwiic/StemmaQt cable

4- Connect the TMP36 (optional)

5- Connect the radio module

The CanSat extension is useful if you need space to add more components for the secondary mission, like GPS, accelerometer, camera etc

What to do next

- Read the resource to design your parachute
- Read the resource to design your radio antennas

Going further

- Consider adding a GPS to increase your chances of finding your CanSat after the launch.
  - Tutorial
  - GPS module
  - MicroPython GPS library
- Some teams design their own ground station real time dashboard

Annexes

Learning Python

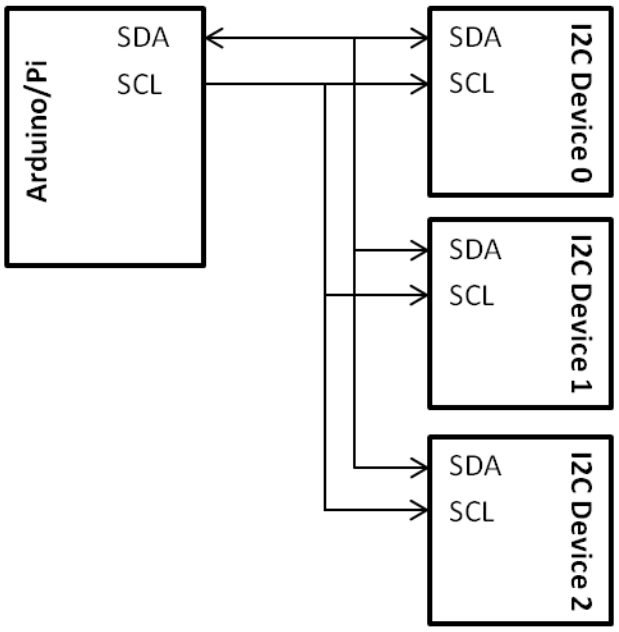
MicroPython is very similar to Python. If you are an absolute beginner in Python, here is a list of useful resources to get you started:

https://wiki.python.org/moin/BeginnersGuide/NonProgrammers

For instance, LearnPython is a great interactive tutorial that is suitable for absolute beginners.

I²C Protocol

I2C allows multiple devices (up to 1008) to be connected to the same I2C interface with just 2 wires. It also allows bi-directional communication over these two wires and so is ideal for communicating with many sensors. An example wiring with three devices would be as follows:



The software required to communicate with I2C devices can be complex, however most devices will have a software library provided that will give you functions that make the device easy to use. For example, we use the provided BMP280 library to hide away the low-level I2C code.

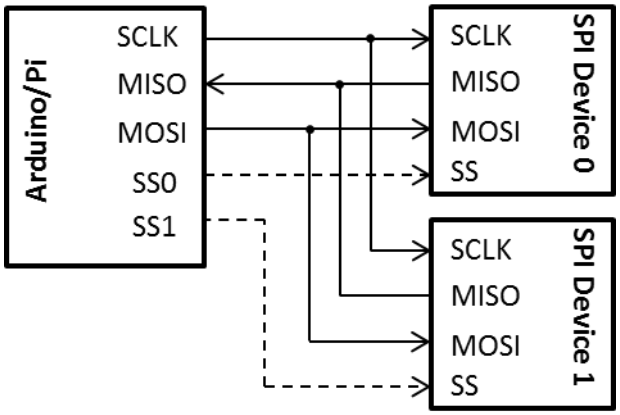
Typical I2C CanSat Uses:
"Smarter" sensors (e.g. the BMP280), Accelerometers, Analog-Digital Converters, Digital-Analog Converters, LCD Screens, Battery Controllers

Serial Peripheral Interface (SPI)

SPI offers an interface with more powerful capabilities than I2C at the cost of more wiring required. As with I2C it also supports bi-directional communication with several devices but offers a much higher data throughput. This makes it suitable for communicating with the most complex devices that you might connect to the CanSat. The interface consists of at least four pins:

SCLK: *Serial Clock*. A stream of 0-1s that the data is aligned to. The SPI clock rate is related to the speed of this stream, you can slow this down if having data integrity issues.
MISO: *Master Input / Slave Output.* The data from the peripheral device to the Pi.
MOSI: *Master Output / Slave Input.* The data from the Pi to the peripheral device.
SS0/CE0: *Slave Select / Chip Enable.* Enables a peripheral device and means that the device can output to the MISO pin. One SS/CE pin is needed for each peripheral device.

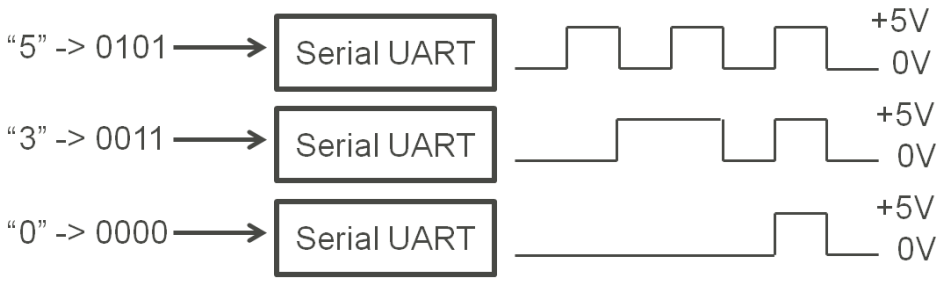
To use SPI you don't need to be too concerned about the function of these pins as the device's software library will take care of most of the low-level SPI code for you. However, it is good to be aware of their function when cascading multiple SPI devices together, for example to connect two devices you will need two SS/CE pins:



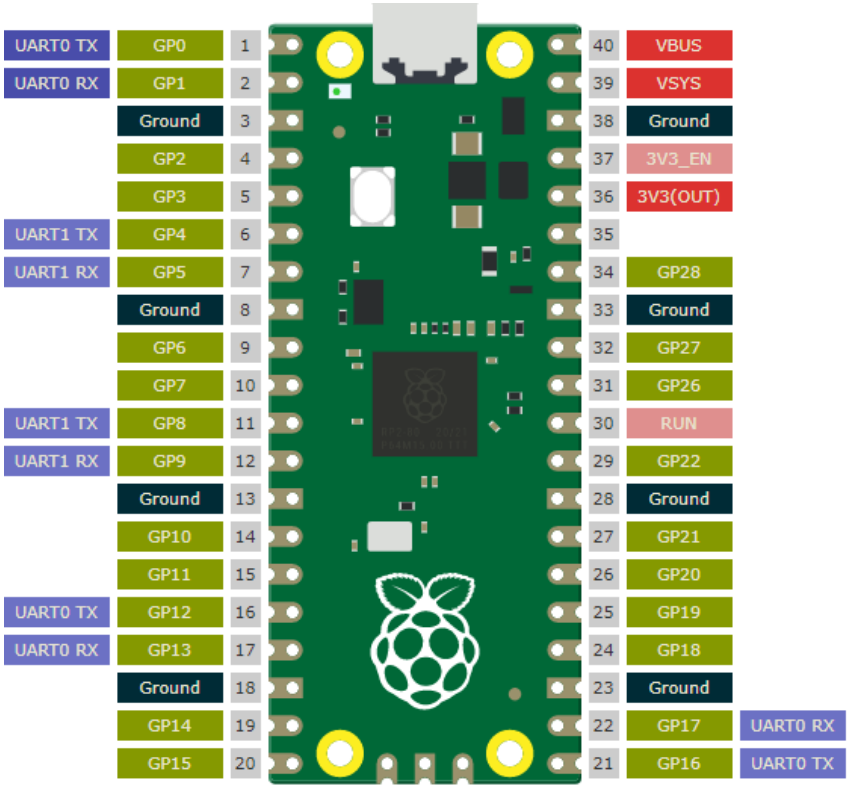
Typical SPI CanSat Uses:
Cameras, Storage cards (e.g. SD cards), GPS modules, WiFi Modems

Universal Asynchronous Receiver-Transmitter (UART)

Despite the name UART is a relatively simple communication interface. It operates in the same fashion as the GPIO with true/false values represented as 0V and 5V but pulses are sent across the wire instead of a steady voltage pulses. This allows a numerical value to be converted to a series of pulses and sent over a single wire:



The Raspberry Pi Pico has two UARTs. These can be connected to many pairs of GP pins as shown in purple in the [pinout diagram] below: TX is the transmit (i.e. data sent out of the Pi) and RX as the receive (i.e. data sent to the Pi).



Typical UART CanSat Uses:
Sending debug and development messages to a PC, communicating with GPS sensors, communicating with external WiFi and GPRS (3G) modems.