



# MicroPython Demoboard

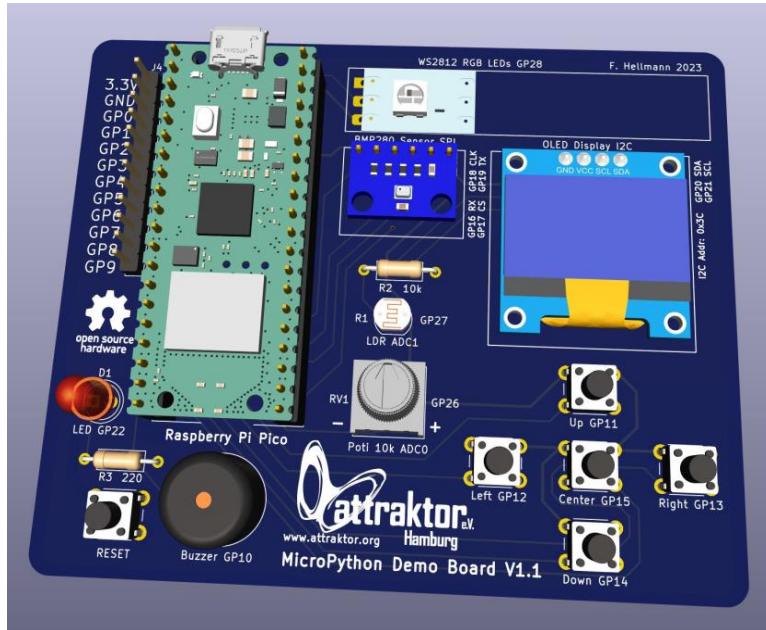
Introduction to the MicroPython Demoboard with programming examples

OSHW under MIT License

Frank Hellmann - 2024

# Introduction to the Attraktor MicroPython Demoboard

The MicroPython Demoboard is designed to give an easy entry into hardware-oriented programming with MicroPython and provide a good foundation for various sensors, input and output devices across a variety of bus systems for demonstration purposes. No breadboards and cables are necessary.



## Hardware of the MicroPython Demoboard:

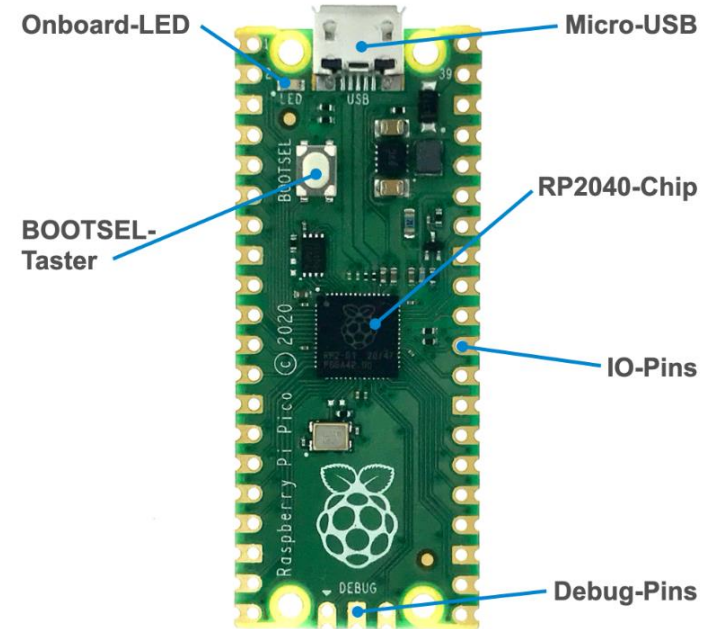
- Pi Pico W Micro-Controller with 2MB Flash, 264KB RAM, 28 In- and Output-Pins, Wifi
- OLED Display: 128x64, blue, SSD1306
- WS2812 LED Strip with 5 RGB LEDs
- 1 blue LED
- Temperature/Pressure Sensor: BMP280
- Lightsensor: LDR
- Potentiometer
- 5 User Buttons
- 1 Reset Button
- 10 free I/O Pins max. 3.3V
- Free and Open Source Hardware Design

# Raspberry Pi Pico: What does it do?

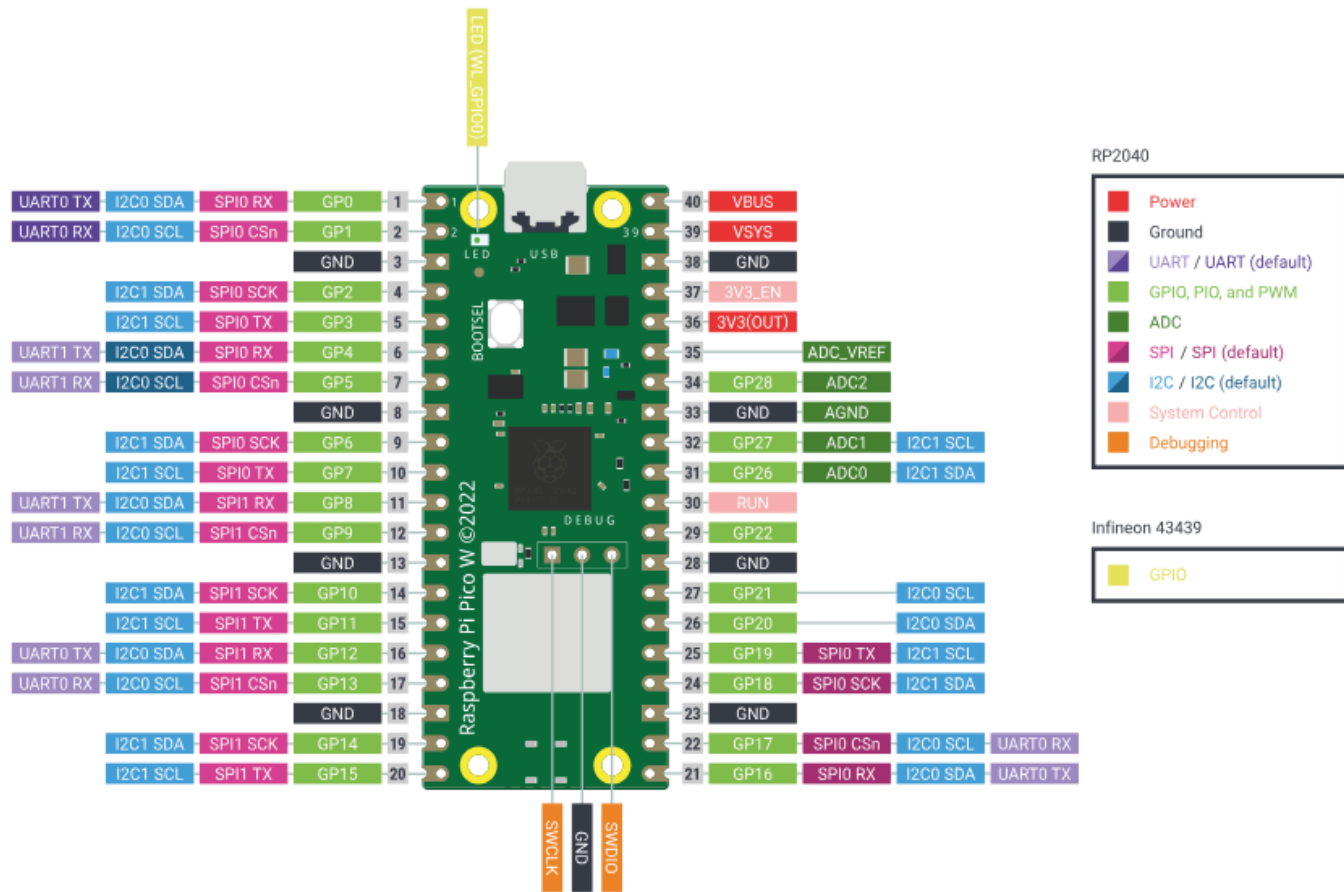
The Raspberry Pi Pico is a microcontroller board from the Raspberry Pi Foundation. It is often affectionately just called 'Pico.' It has some useful features for controls, as known from other microcontrollers. It can be programmed with MicroPython, C/C++, and Thonny or Visual Studio Code. The board can be powered and programmed via a micro-USB socket.

The microcontroller is an RP2040. For storing program code, there is 2 MB of flash memory and 264KB RAM available. A special feature, compared to other microcontrollers, are the Programmable IOs (PIO), which are independently programmable from the main controller.

A microcontroller like the Pico is intended to take over individual tasks. Typically from the field of control, regulation, and automation. For this purpose, it has numerous analog and digital inputs and outputs with different functions.



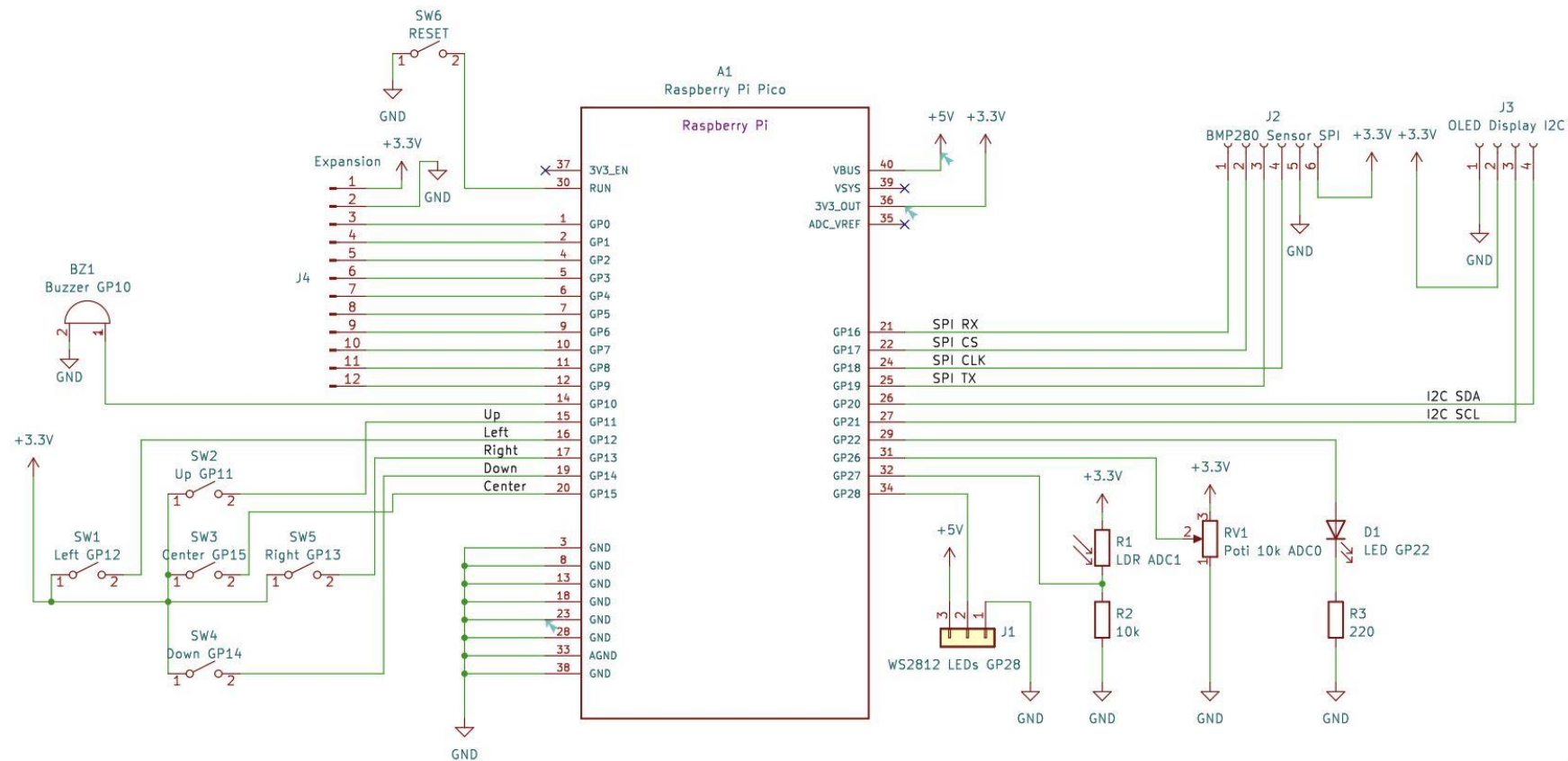
# Raspberry Pi Pico W mit Wifi: Pinout



# MicroPython Demoboard

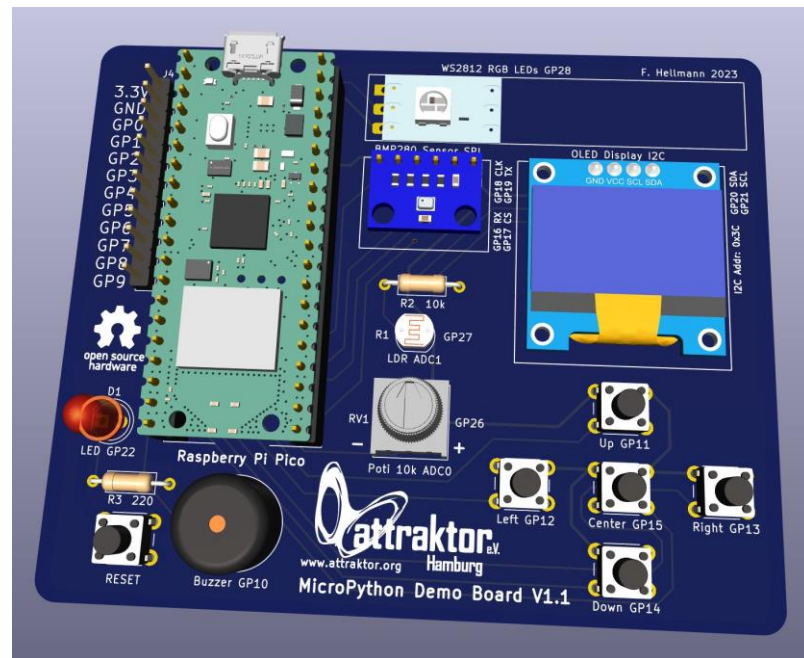
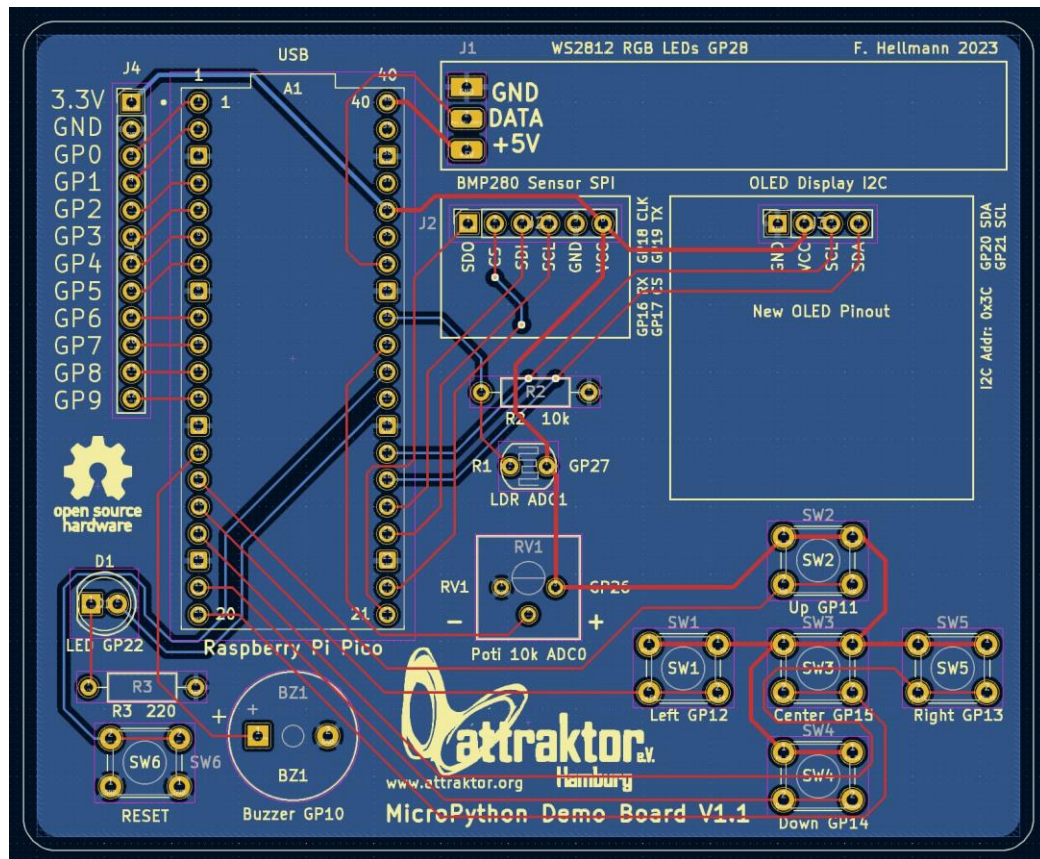
- Schematic, PCB and BOM
- Building the board
- Installing the firmware

# MicroPython Demoboard Schematic






# MicroPython Demoboard PCB



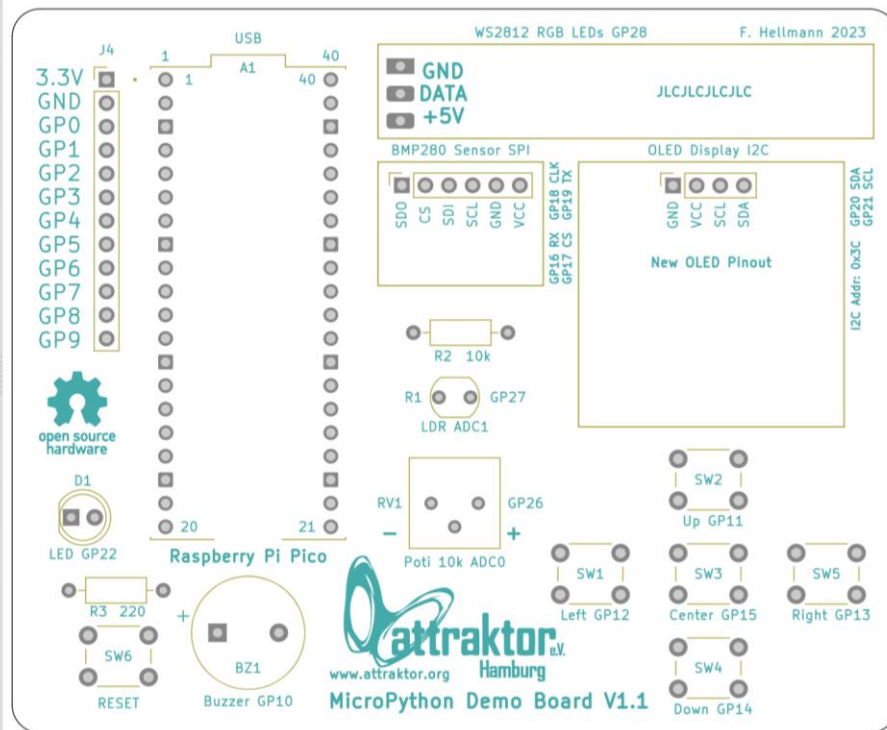
[https://github.com/sandman72/Micropython\\_Demoboard](https://github.com/sandman72/Micropython_Demoboard)

# MicroPython Demoboard – Bill of Material

	Sour ced	Plac ed	References	Value	Footprint	Quantity
1	<input type="checkbox"/>	<input type="checkbox"/>	R1	LDR ADC1	R_LDR_5.1x4.3mm_P3.4mm_Ve rtical	1
2	<input type="checkbox"/>	<input type="checkbox"/>	R2	10k	R_Axial_DIN0207_L6.3mm_D2 .5mm_P10.16mm_Horizontal	1
3	<input type="checkbox"/>	<input type="checkbox"/>	R3	220	R_Axial_DIN0207_L6.3mm_D2 .5mm_P10.16mm_Horizontal	1
4	<input type="checkbox"/>	<input type="checkbox"/>	D1	LED GP22	LED_D5.0mm	1
5	<input type="checkbox"/>	<input type="checkbox"/>	SW1	Left GP12	SW_PUSH_6mm_H7.3mm	1
6	<input type="checkbox"/>	<input type="checkbox"/>	SW2	Up GP11	SW_PUSH_6mm_H7.3mm	1
7	<input type="checkbox"/>	<input type="checkbox"/>	SW3	Center GP15	SW_PUSH_6mm_H7.3mm	1
8	<input type="checkbox"/>	<input type="checkbox"/>	SW4	Down GP14	SW_PUSH_6mm_H7.3mm	1
9	<input type="checkbox"/>	<input type="checkbox"/>	SW5	Right GP13	SW_PUSH_6mm_H7.3mm	1
10	<input type="checkbox"/>	<input type="checkbox"/>	SW6	RESET	SW_PUSH_6mm_H7.3mm	1
11	<input type="checkbox"/>	<input type="checkbox"/>	A1	Raspberry Pi Pico	Raspberry_Pi_Pico_WH Dual 20pin Sockets	1
12	<input type="checkbox"/>	<input type="checkbox"/>	BZ1	Buzzer GP10	Buzzer_12x9.5RM7.6	1
13	<input type="checkbox"/>	<input type="checkbox"/>	RV1	Poti 10k ADC0	Potentiometer_Bourns_3386 P_Vertical	1
14	<input type="checkbox"/>	<input type="checkbox"/>	J1	WS2812 RGB LEDs GP28	LED Strip 100Leds/m 5pcs	1
15	<input type="checkbox"/>	<input type="checkbox"/>	J2	BMP280 Sensor SPI	BMP280 I2C/SPI 6pin Module	1
16	<input type="checkbox"/>	<input type="checkbox"/>	J3	OLED Display I2C	OLED 1306 0,96 4pin Module	1
17	<input type="checkbox"/>	<input type="checkbox"/>	J4	Erweiterung	PinHeader_1x12_P2.54mm_Ve rtical	1

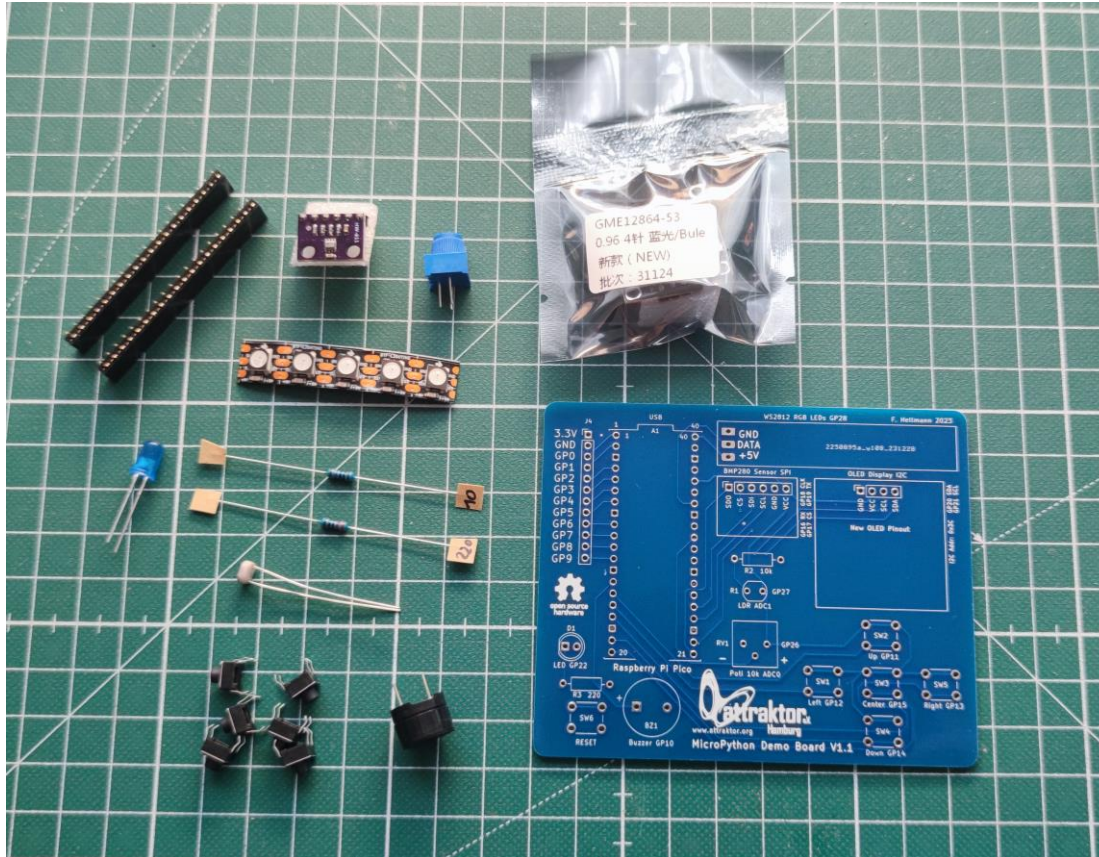
**Interactive BOM is found here:**

[https://github.com/sandman72/Micropython\\_Demoboard](https://github.com/sandman72/Micropython_Demoboard)





# Building the MicroPython Demoboard



Here are the parts of the Demoboard that we are building.

You see the blue board also known as a PCB that will hold all of our components.

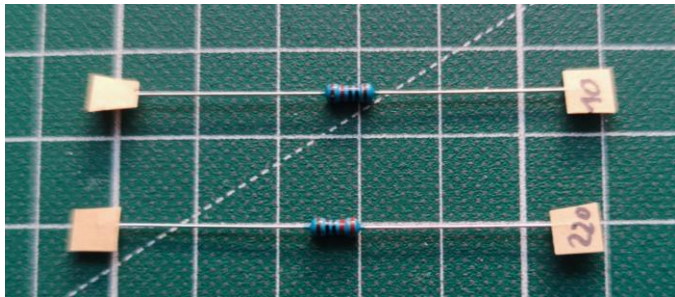
We will go over the components on the following pages.

Please make sure, if you are soldering a kit, that you know how to do this. If you are unsure about anything, makerspaces are a great place to get advice.

# Building the MicroPython Demoboard



R1



R2

R3

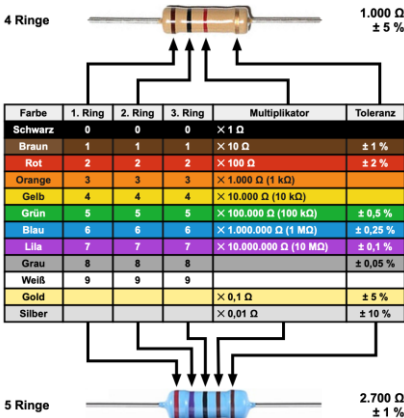
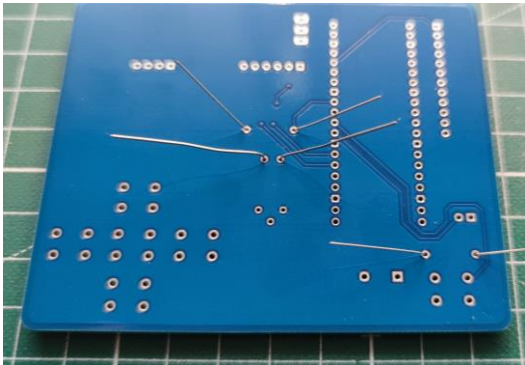
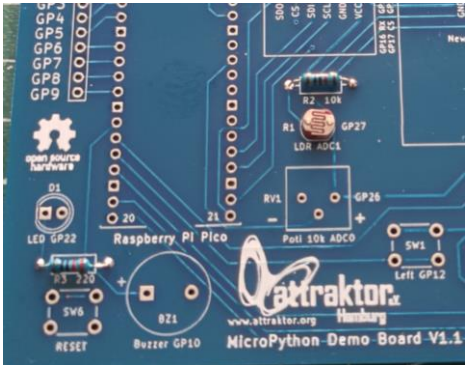
First we place the Resistors R1, R2 und R3

R1 is the light dependent resistor (LDR)

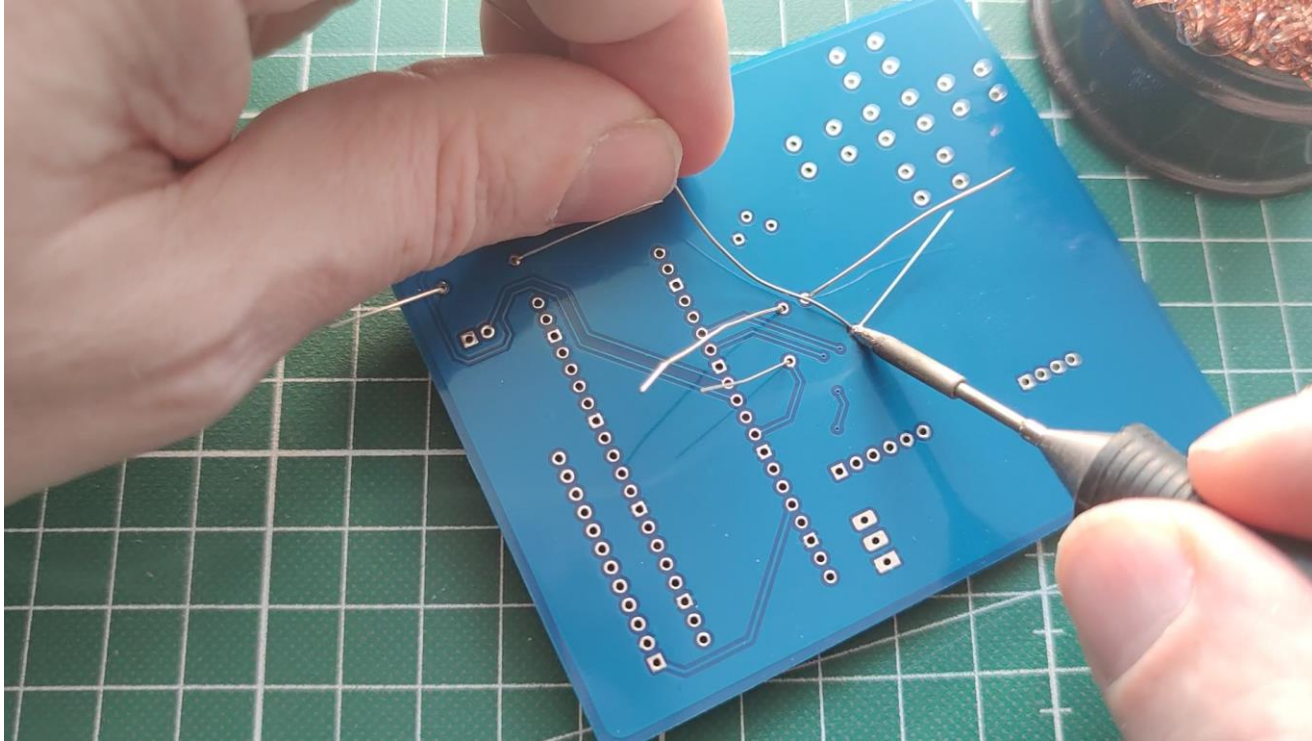
R2 is a 10K Ohm Resistor (10 Kilo Ohm)  
(Color code: brown – black – black – red)

R3 is a 220 Ohm Resistor  
(Color code:: red – red – black – black)

First we'll bend the legs of R2 and R3, place them into the matching footprints and bend the legs slightly on the other side to prevent them from falling out.



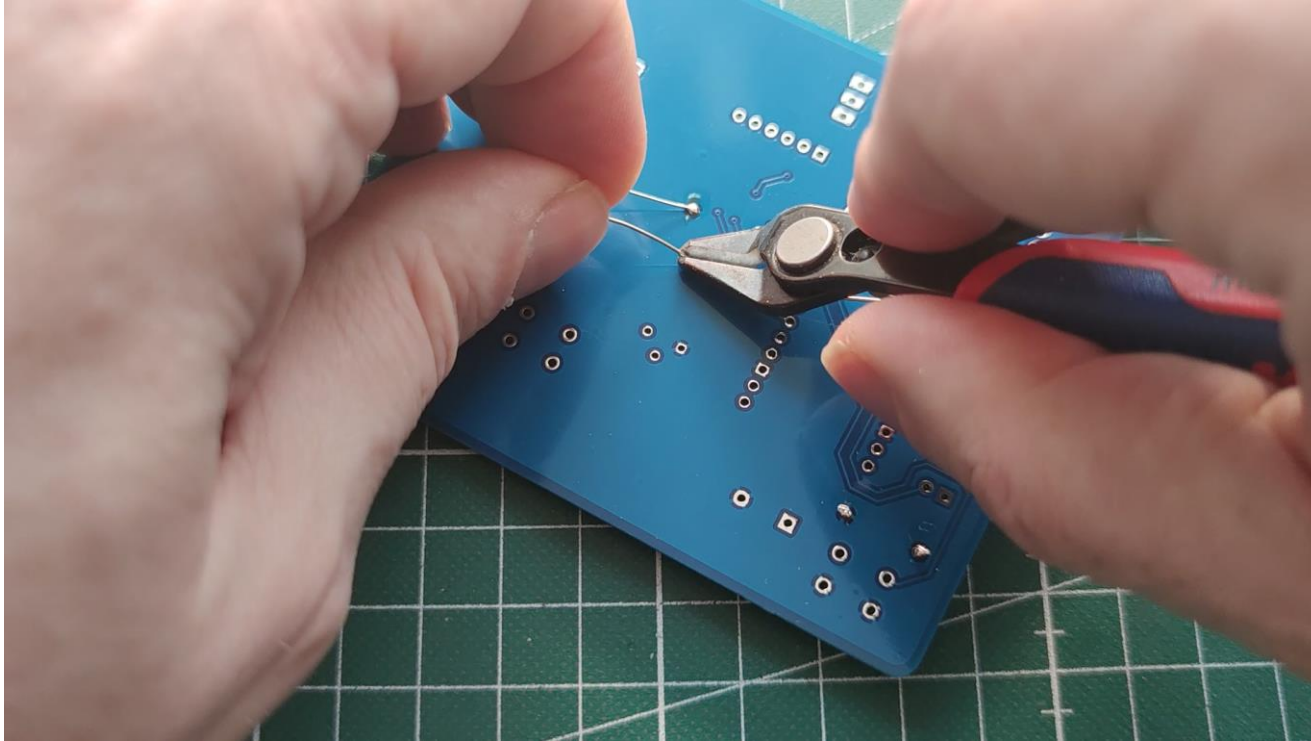
# Building the MicroPython Demoboard



Now we solder them into place.



# Building the MicroPython Demoboard



And we trim off the excess legs.

Flush above the solder joint,  
making sure not to cut into the  
solder.

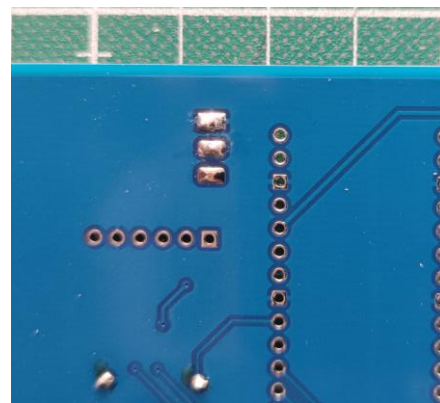
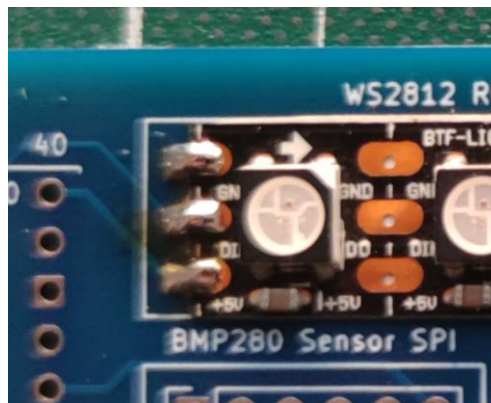
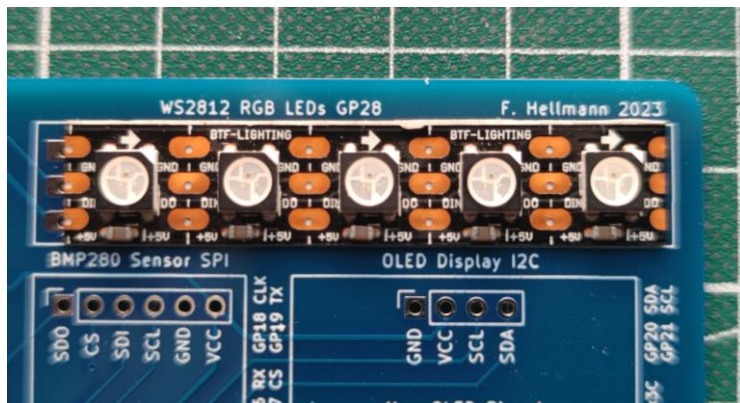
# Building the MicroPython Demoboard



Next we will solder the WS2812 LED Strip

It is important that the little **arrow points right** or the Pin **DO** is **right**!

First we peel off the protective film on the back of the LED strip and stick it onto the board so that the solder pads are nicely aligned. Then we solder the three pads down, making sure they make good contact. On the backside of the board check that the soldering pads look good as well.

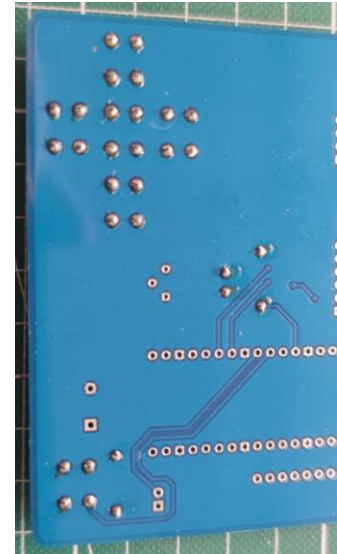
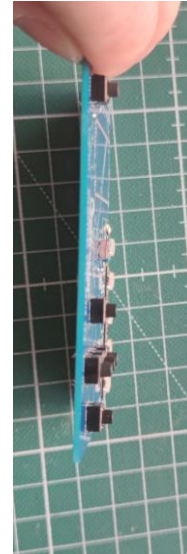
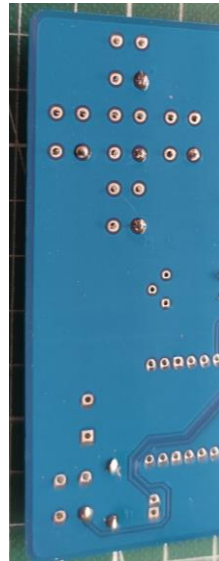


# Building the MicroPython Demoboard



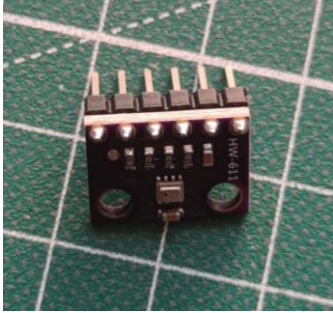
## Now we solder on the buttons

First check that the legs of the buttons are not mangled and look reasonably springy. Then place the buttons and solder on one pin of each button. Check that the buttons are flush on the board and correct if necessary. If all buttons are looking good solder the remaining pads.





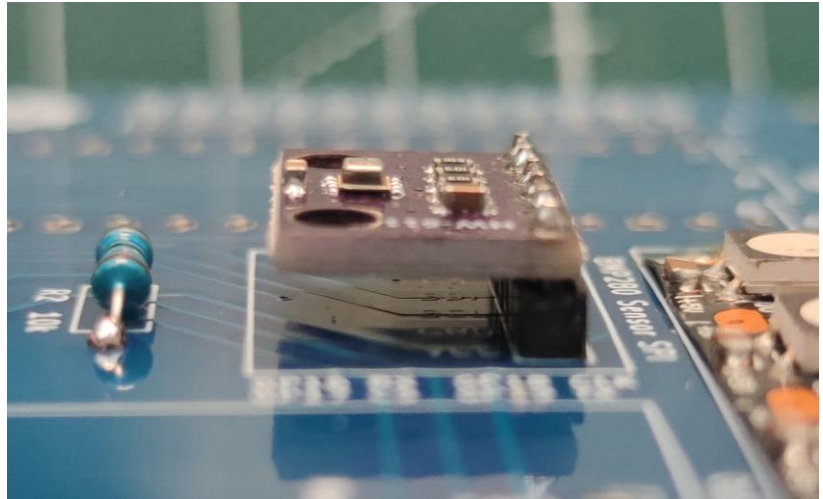
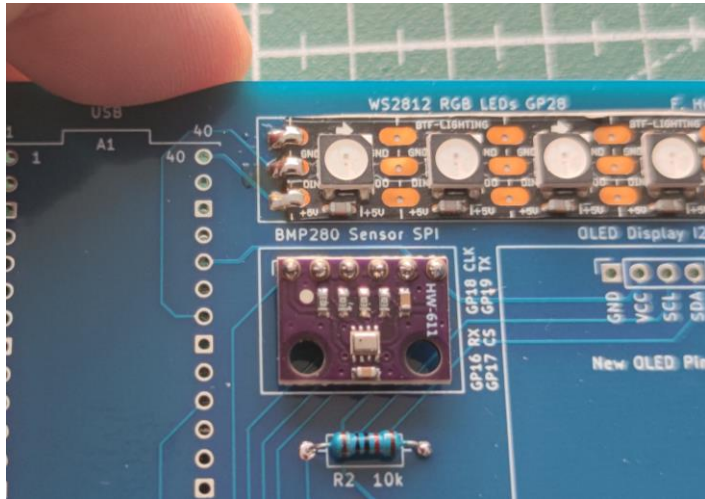
# Building the MicroPython Demoboard



**Next we solder the climate sensor BMP280 on to the board**

This sensor is also placed into the right spot on the board and we solder one pin down to allow for easier positioning and corrections. If the sensor is sitting straight and parallel to the board, solder down the rest of the pins.

Now trim off the excess legs and be careful as these pins are thicker and tend to fly off with more force.



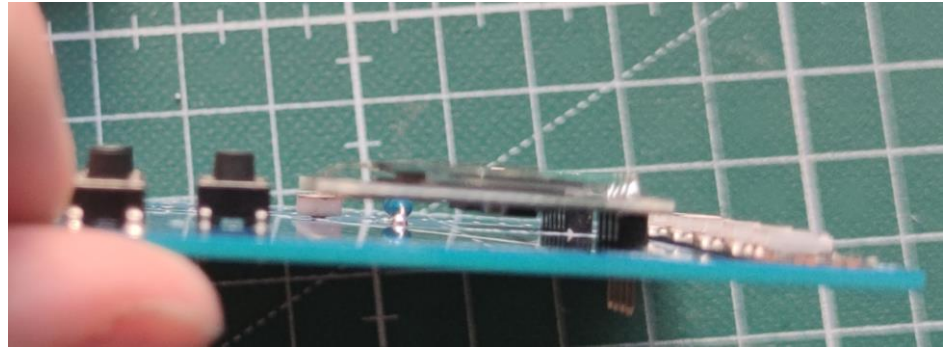
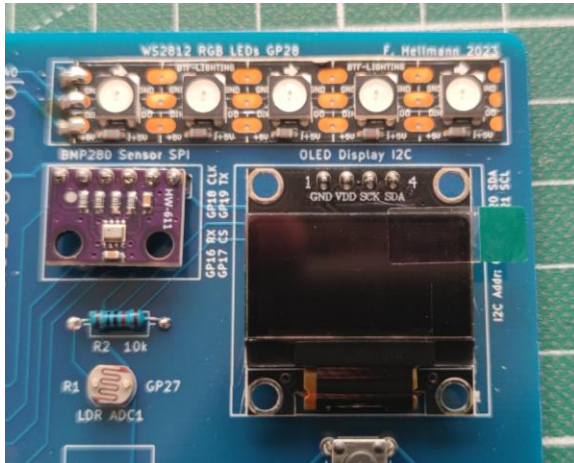
# Building the MicroPython Demoboard



## Now the OLED display is soldered on

This OLED is also placed into the right spot on the board and we solder one pin down to allow for easier positioning and corrections. If the display is sitting straight and parallel to the board, solder down the rest of the pins.

Now trim off the excess legs and be careful as these pins are thicker and tend to fly off with more force.



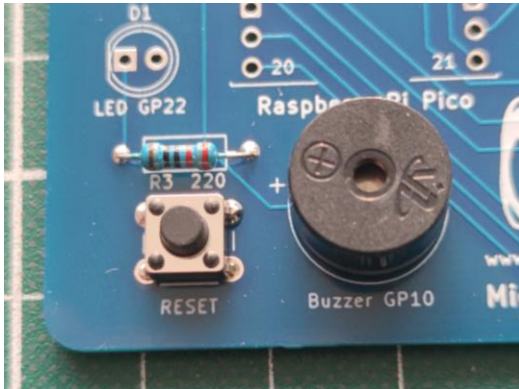
# Building the MicroPython Demoboard



## We solder the buzzer on

The buzzer is marked with a Plus + indentation on one side. Put this leg through the appropriate hole and solder it down. Check that everything looks flush and correct and solder the other pin down.

Now trim off the excess legs and be careful as these pins are thicker and tend to fly off with more force.



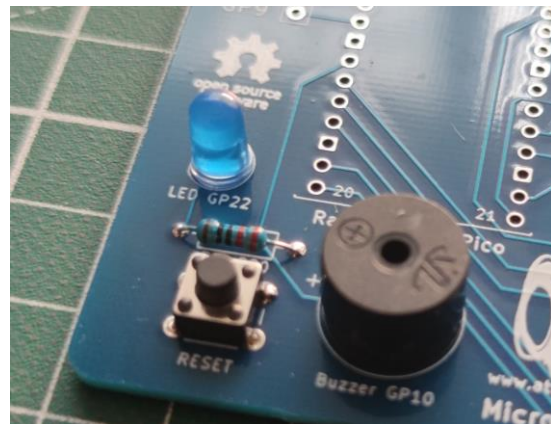
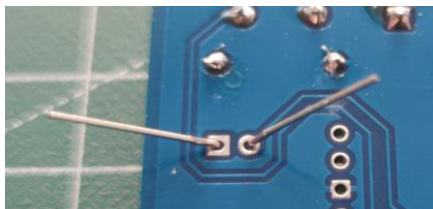
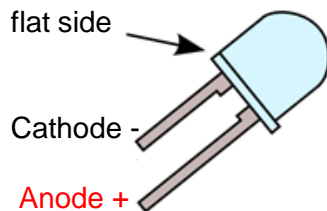
# Building the MicroPython Demoboard



## Now we solder on the light emitting diode LED

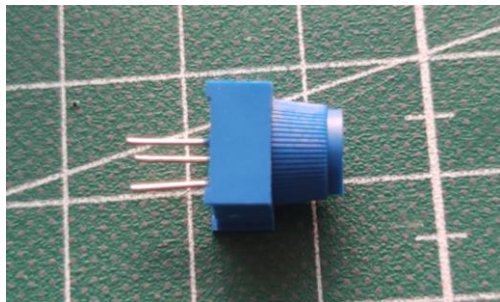
The LED has a specific mounting direction that is important. The **flat side** with the **shorter leg** is the **Cathode -** of the LED and needs to be placed into the **square pad**. The so called **Anode +** into the other hole next to it.

Then bend the legs on the underside slightly, check that the LED sits flush, solder it down and trim off the excess legs.





# Building the MicroPython Demoboard

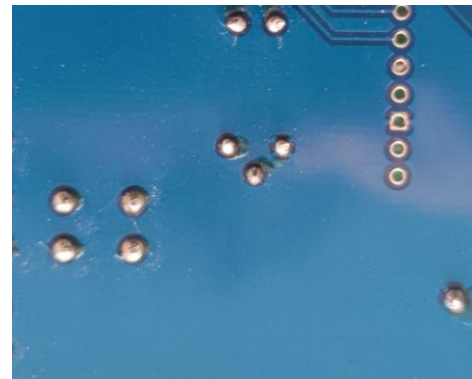
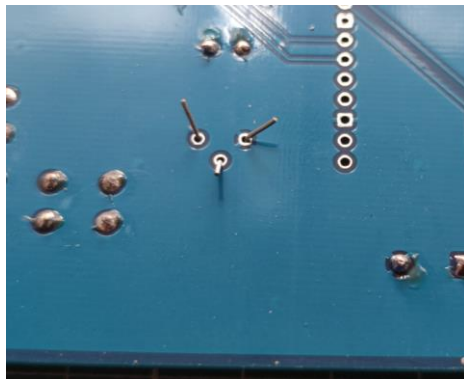


## The Potentiometer follows next

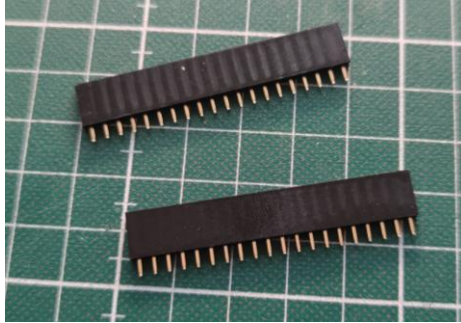
A potentiometer is a variable Resistor that goes into position RV1. Put it into the matching pads and bend the legs slightly. Check that it sits flush and solder the legs down. Trim the excess off.

This potentiometer allows us to send an analog voltage to the Pi Pico that can be read on one of its analog inputs as a digital value:

0V = 0    1.8V = 512    3.3V = 1023



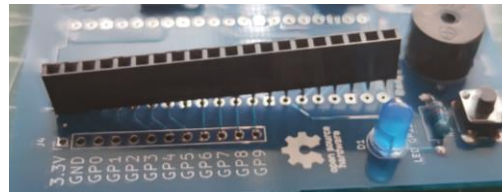
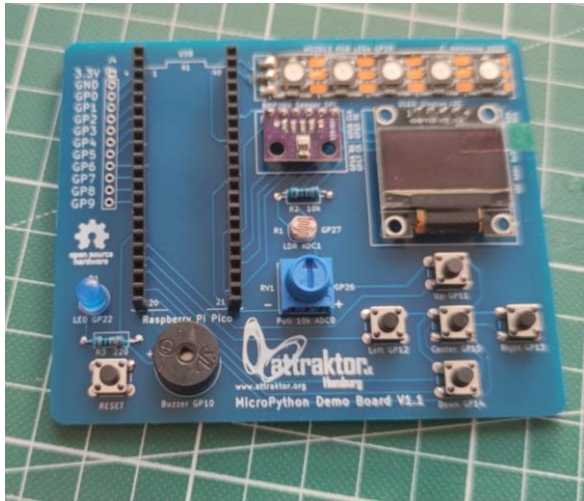
# Building the MicroPython Demoboard



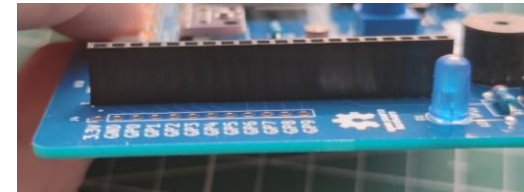
## The pin sockets are the last parts to solder

Be careful to double-check the position of these sockets as they are very hard to fix later and the Pi Pico might not fit into them! You can place the Pi Pico into the sockets to help with alignment.

First put one socket into the board and solder down both end pins. Check that the socket is straight in all axis and flush to the board. Now do this with the second socket as well. You can test fit the Pi Pico now and if it fits well into the socket solder down all the remaining pins. There is no need to remove the Pi Pico during soldering.



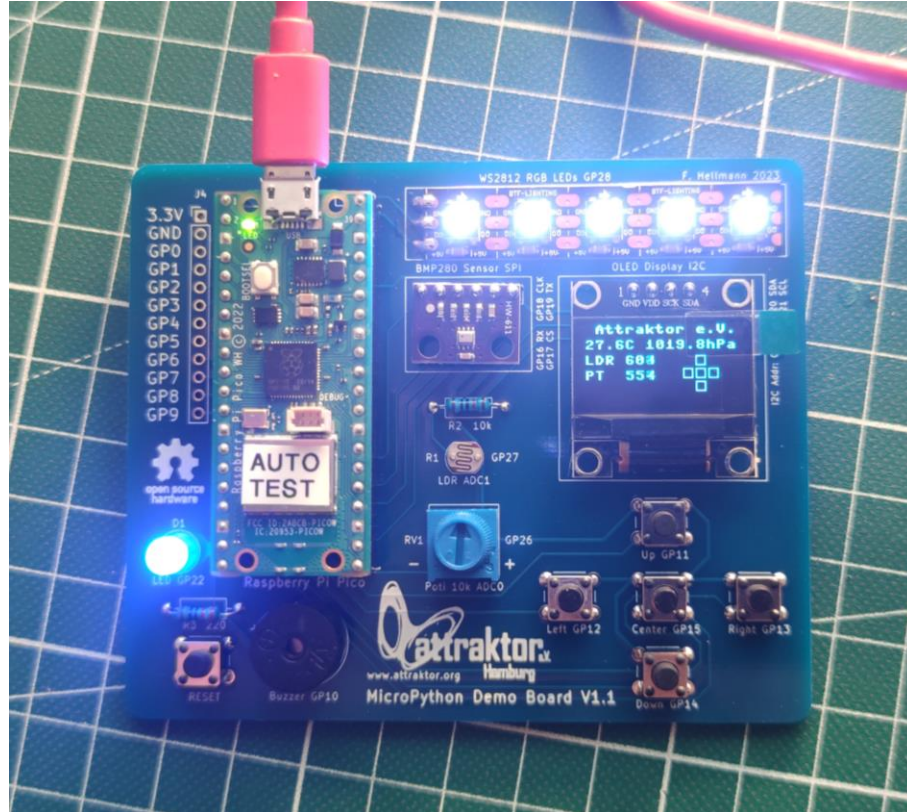
Needs rework!



Looking good!



# Building the MicroPython Demoboard



**We are done soldering!**

Before you plug in USB power, go over the board once again and double-check all pads for cold solder joints, solder blobs, bridges and shorts. Make sure it all looks good to you.

(If you didn't get a preprogrammed Pi Pico with your kit follow the firmware installation procedure on the next page)

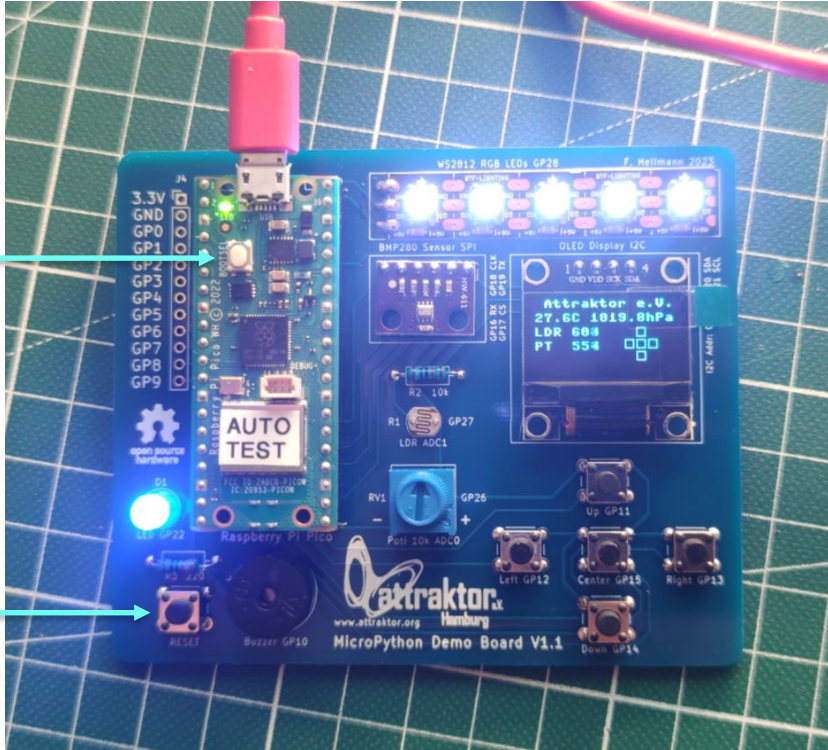
Now you can install the Pi Pico and connect it to USB power or your computer. The demo menu should come up and you can run the *Board Test* to check all the functions of the board.

If everything works, we are:

**Done!**

# Installing the Demoboard Firmware

1. Press Bootsel and keep it pressed



2. Push Reset briefly

## Installing the Demoboard Firmware:

In case the Pi Pico does not have the firmware for the MicroPython Demoboard preinstalled, or you need to reinstall the firmware:

Connect the Pi Pico with the USB cable to a computer. Then press the little white Bootsel button and at the same time briefly press the Reset button. Now your computer should show a new drive (RPI-RP2) and you can simply Drag&Drop the firmware file on to this drive. You can now release the buttons. This will erase the Pi Pico and install the clean firmware.

The firmwares are available here in the UF2 format:

[https://github.com/sandman72/Micropython\\_Demoboard](https://github.com/sandman72/Micropython_Demoboard)

# MicroPython Introduction

- What is MicroPython?
- First examples with Thonny

# What is MicroPython?

MicroPython is a reduced version of the Python programming language that was specifically developed for microcontrollers. It covers the main functionalities of Python but is designed for the smaller memory of microcontrollers. With MicroPython, the hardware of the Pi Pico is completely abstracted and can be loaded with a simple *import machine*.

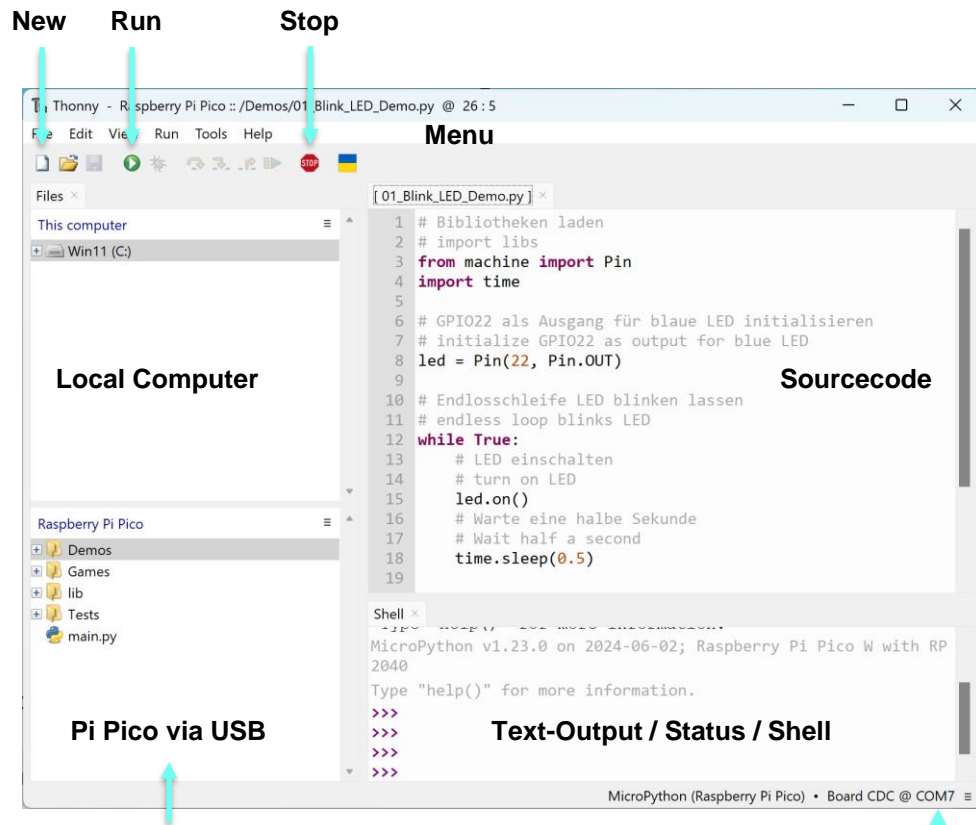
The most striking difference from Python to other programming languages is the indentation of groupings with tabs instead of brackets or similar. This can be seen in the examples.

By default, the Raspberry Pi Pico is set up for programming with C and C++. If you want to program the Pico with MicroPython, then MicroPython must first be installed on the board. The process is one-time and consists of downloading a firmware file from the Internet and copying it onto the Pico. This is described on the page: *Installing the Demoboard Firmware*.

Also editors like Thonny can do this, although not all the necessary libraries for this MicroPython Demoboard are included there. Further information can be found in the official documentation:

<https://docs.micropython.org/en/latest/rp2/quickref.html>

# Thonny Editor



**Tipp:** Under Menu **View** turn on the **Files** option

**selected USB Port**

The **Thonny** Python Editor has all the needed functions and helpers to program the Pi Pico in MicroPython. It is easy to work with and a good choice for beginners.

## Thonny interface is split up into

1. Menu Bar to control things
2. Source code text editor
3. Text output and Status area
4. USB Port and Board selection

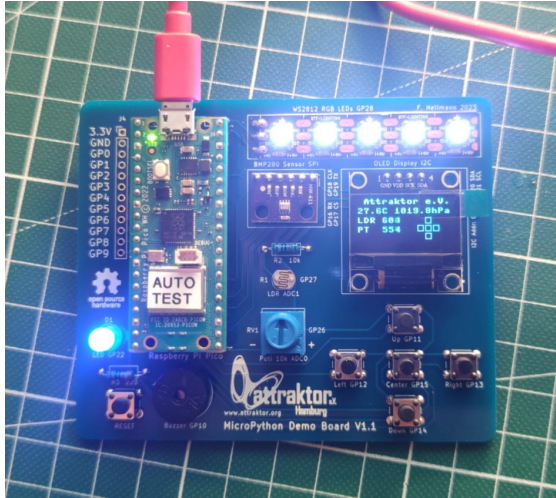
## Important functions in the menu bar

- New file
- Load file
- Save file
- Run active editor on Pi Pico
- Stop program running on Pi Pico

Thonny is available here: <https://thonny.org/>



# First examples with Thonny



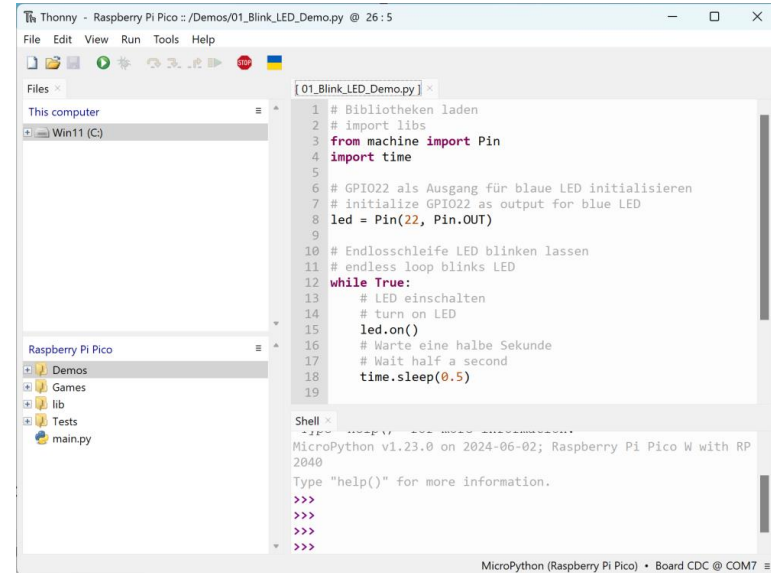
***hook up board via USB***

from machine import Pin  
import time



led = Pin(22, Pin.OUT)

while True:  
 led.on()  
 time.sleep(0.5)  
 led.off()  
 time.sleep(0.5)

***enter source code***

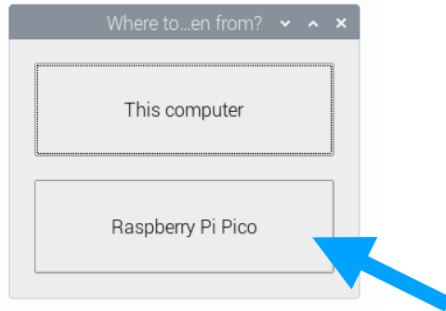


***run program from Thonny***

1. Connect the Pi Pico via USB cable to your Computer and select the USB port within the Thonny editor.
2. Press the  Stop Button in Thonny to connect to the Pi Pico.
3. Enter the source code.
4. Press the  Start Button in Thonny to run the program on the Pi Pico.
5. In the best case the program just works and blinks an LED. If not, check the output of the status area.



# First examples with Thonny



If you want to permanently save a program on the Pi Pico, you can do so using the 'Save' function and selecting the Raspberry Pi Pico as the destination. It's best practice to give the program a meaningful name with the **.py** extension so that the Pi Pico recognizes it as a MicroPython program.

**There are certain files that should only be changed if you know what you're doing:**  
***main.py*** and the libraries provided in the ***lib*** directory fall into this category.

When a program no longer runs, it can happen that the light-emitting diodes still remain lit from the last program step. These "states" are preserved for now. If you then start a different program, some LEDs continue to light up even though you've pressed the Stop button or the reset button.

It's advisable to unplug and replug the Raspberry Pi Pico before starting a new program. This action clears all remnants of the old program from memory.

Note: After reconnecting, you'll need to reset the connection to the Raspberry Pi Pico by clicking the





Stop button.

# First examples with Thonny

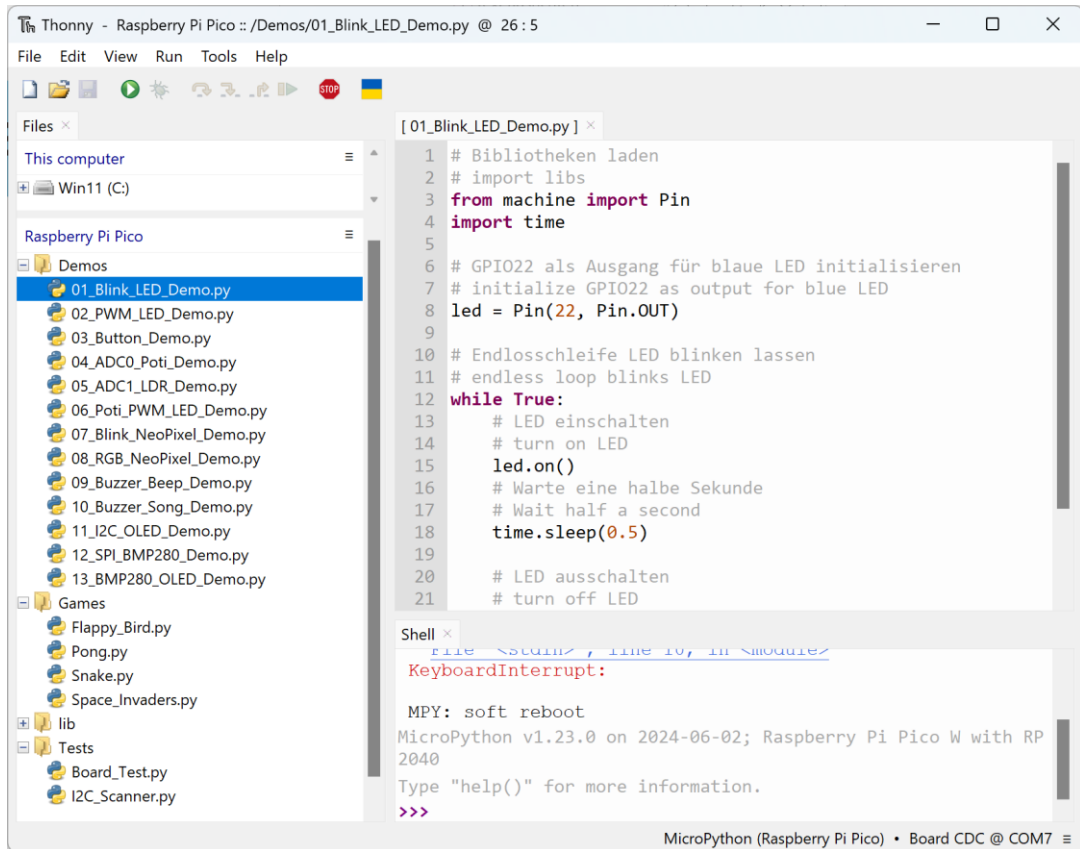
## Supplied Examples

To avoid having to type in all the examples by hand, there is a Demos folder on the Pi Pico Filesystem that contains the MicroPython examples.

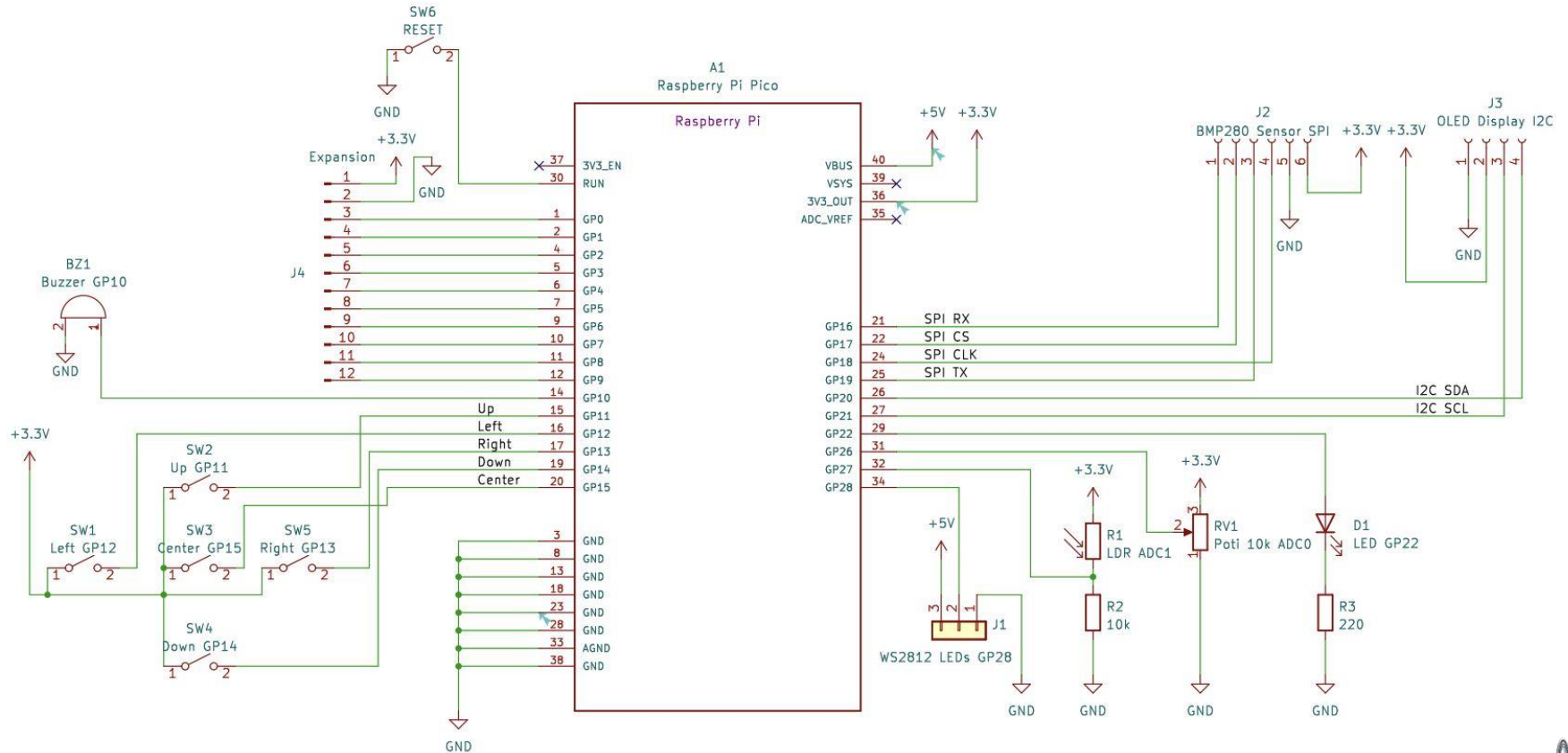
Let's start with the first example by simply loading it into the editor with a double click. If we now press the  Start button, the blue LED on our board should flash at half-second intervals.

If we then press the  Stop button, the last state of the LED is retained, and the program stops.

**Tip:** If no files are displayed on the left, turn on *Files* under the *View* menu.



# MicroPython Demoboard Schematics



# Example 1 – Blink LED

```
# import libs
from machine import Pin
import time

# initialize GPIO22 as output for blue LED
led = Pin(22, Pin.OUT)

# endless loop blinks LED
while True:
    # turn on LED
    led.on()
    # Wait half a second
    time.sleep(0.5)

    # turn off LED
    led.off()
    # Wait half a second
    time.sleep(0.5)
```

In this example we will let the blue LED blink.

On the board the LED is marked with GP22 and on the circuit diagram, we see at the bottom right that GPIO Pin 22 is connected to the LED. GPIO stands for General Purpose I/O, which allows us to use it as an input or output pin and control the LED.

First we import the libraries that enable MicroPython to communicate with the hardware and also allows for delays.

Next we initialize GPIO Pin 22 with *Pin.OUT* as an output.

Inside the while loop the LED is turned on, followed by a half-second delay, then the LED is turned off, and another half-second delay occurs.

The *while* loop repeats indefinitely because it never breaks with the *True* condition.

# Example 2 – Fade LED with PWM

```
# import libs
from machine import Pin, PWM
from time import sleep

# initialize GPIO22 as PWM output for blue LED
pwm = PWM(Pin(22))

# set PWM frequency (Hz)
pwm.freq(4000)

# endless loop fading LED brightness up and down
while True:
    # fade up brightness
    for duty_cycle in range(0, 65536, 129):
        pwm.duty_u16(duty_cycle)
        sleep(0.005)

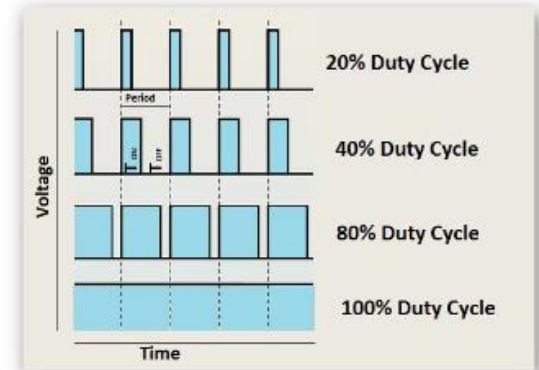
    # fade down brightness
    for duty_cycle in range(65536, 0, -129):
        pwm.duty_u16(duty_cycle)
        sleep(0.005)
```

In this example we let the blue LED slowly fade up and down.

We have seen in the first example that we can control the LED with GPIO pin 22. However, here we do not simply turn it on or off; instead, we use pulse-width modulation (PWM).

The Pi Pico assists us with its hardware PWM to adjust the brightness of an LED.

This means that we only need to tell the Pi Pico how bright we want the LED to be. We can achieve this using the `pwm.duty_u16` function, which accepts a value between 0 and 65535.



0 corresponds to 0%, 32768 to 50%, and 65535 to 100%.

The frequency determines how often this happens per second. If the frequency is too low, it may result in flickering.

# Example 3 – Buttons

```
# import libraries
from machine import Pin
import time

# initialize GPIO pins as input with Pull_Downs enabled
button_up   = Pin(11, Pin.IN, Pin.PULL_DOWN)
button_left = Pin(12, Pin.IN, Pin.PULL_DOWN)
button_right = Pin(13, Pin.IN, Pin.PULL_DOWN)
button_down = Pin(14, Pin.IN, Pin.PULL_DOWN)
button_center = Pin(15, Pin.IN, Pin.PULL_DOWN)

# endless loop with button actions
while True:
    if button_up.value() == 1:
        print("Up Button is Pressed")
    if button_left.value() == 1:
        print("Left Button is Pressed")
    if button_right.value() == 1:
        print("Right Button is Pressed")
    if button_down.value() == 1:
        print("Down Button is Pressed")
    if button_center.value() == 1:
        print("Center Button is Pressed")
    time.sleep(0.1)
```

In this example we use the buttons on the board for input.

On the board the buttons are marked with GP11 to GP15, and on the circuit diagram, we see how they are connected to the Pi Pico at the bottom left. When a button is pressed the corresponding input of the Pi Pico is connected to 3.3V.

First we import the libraries again.

Then we initialize all buttons with **Pin.IN** as input and with the option **Pin.PULL\_DOWN**. This pull-down slightly pulls the inputs to ground as long as no button is pressed. This guarantees that we have only two valid states: either 3.3V = button pressed or 0V = nothing happening.

In the *while* endless loop, the buttons are queried using the **value()** function. If a button is pressed (i.e., value = 1), this is written to the status area. The print function allows communication from the Pi Pico to the outside world, in this case, to our Thonny status area.



# Example 4 – Reading a Potentiometer

```
# import libs
import machine
import utime

# initialize analog/digital converter on ADC0 GPIO 26
# where the RV1 potentiometer is connected to
potentiometer = machine.ADC(26)

# endless loop reading and displaying the potentiometer value
while True:
    print("Poti: " + str(potentiometer.read_u16()))
    utime.sleep(1)
```

In this example we use an analog input.

Computers struggle with analog values and prefer digital zeros and ones. To transition from our analog world to the digital one, we use what's called an ADC (Analog/Digital Converter). On our demo board ADC0 is connected to a rotary potentiometer. The potentiometer is used as an adjustable voltage divider and allows us to set analog voltages from 0V to 3.3V. The Pi Pico reads these with the ADC0 and prints them as digital values.

First, we load the libraries again.

Next, we initialize GPIO 26 as ADC0 and continuously display the queried `potentiometer.read_u16()` values in Thonny's status area. If we turn the blue potentiometer, the values change once per second.

In the print function, we concatenate a text/string and an integer value. Therefore, we need to convert the integer value using the `str()` function to get a string.

# Example 5 – Reading the light dependent resistor

```
# import libs
import machine
import utime

# initialize analog/digital converter on ADC1 GPIO 27
# where the light dependent resistor LDR1 is connected to
ldr = machine.ADC(27)

# endless loop reading and displaying the LDR value
while True:
    print("LDR: " + str(ldr.read_u16()))
    utime.sleep(1)
```

In this example we use the second analog input to read the light dependent resistor.

A light dependent resistor, or also called LDR, changes its resistance when light reaches it.

First, we load the libraries again.

Next, we initialize GPIO 27 as ADC1 and continuously display the queried `ldr.read_u16()` values in Thonny's status area. If we now block light reaching the LDR with a finger, the values will get lower and if we shine light onto the LDR the values will go higher.

# Example 6 – Controlling the LED brightness with the Poti

```
# import libs
from machine import Pin, PWM
from time import sleep

# initialize GPIO22 as PWM output for blue LED
pwm = PWM(Pin(22))

# set PWM frequency (Hz)
pwm.freq(4000)

# initialize analog/digital converter on ADC0 GPIO 26
# where the RV1 potentiometer is connected to
potentiometer = machine.ADC(26)

# endless loop reading the potentiometer value and
# adjusting the LED brightness via PWM
while True:
    pwm.duty_u16(potentiometer.read_u16())
    sleep(0.005)
```

In this example we combine the previous examples and control the brightness of the blue LED with the potentiometer RV1. When you turn the potentiometer the brightness of the LED will change.

The previous examples also showed us the different ways to let the Pi Pico wait a bit. Here we do this with the *sleep* command.

# Example 7 & 8 – Controlling the LED Strip

```
# import libs
from machine import Pin
from neopixel import NeoPixel
from time import sleep_ms

# initialize NeoPixel/WS2812 LED Strip on GPIO 28
neopin = Pin(28, Pin.OUT)

# Number of LEDs
leds = 5

# Max. Brightness: 0 to 255
brightness = 30

# Speed in milli seconds
speed = 50

# initialize WS2812/NeoPixel
pixels = NeoPixel(neopin, leds)

# endless loop turning on LEDs
while True:
    for i in range (leds):
        # turn on next LED (R,G,B)
        pixels[i] = (brightness, brightness, brightness)
        pixels.write()
        # short delay
        sleep_ms(speed)
        # reset LED
        pixels[i] = (0, 0, 0)
```

In this example we will control the LED strip.

An intelligent LED strip is installed on the demo board that we can address with the NeoPixel library. This LED strip is connected to output GPIO 28 which we then pass on to the NeoPixel library so that it can take control of this pin. This means we don't have to worry about the details of the NeoPixel and can simply pass values for the red, green, and blue color components.

First, we load the libraries and define GPIO 28 as an output. Then we specify the number of LEDs, how bright and fast our blinking should run, and create a NeoPixel object that contains information about our pixels; for example, the very first pixel is controlled with `pixel[0] = (red, green, blue)`. So Python counts these pixel array from zero on upwards.

In the endless loop, we then briefly light up one pixel at a time and then move on to the next one in each iteration of the for loop.

Example 8 creates a rainbow on the LED strip.

# Example 9 & 10 – Making Sound

```
# import libs
from machine import Pin, PWM
from utime import sleep

# initialize Buzzer on GPIO 10 with PWM
buzzer = PWM(Pin(10))

# set PWM frequency (Hz)
frequency = 1000
buzzer.freq(frequency)

# turn Buzzer on for 0.3 seconds
buzzer.duty_u16(1000)
sleep(0.3)

# turn off Buzzer
buzzer.duty_u16(0)
```

In this example we use the buzzer to make sound.

Here we take another look at the PWM generator and use it to play a tones. We use the PWM frequency to determine the pitch of the tone and use the duty cycle only for turning it on and off.

For example, the duty cycle can be used to adjust the volume and the frequency to adjust the pitch.

Example 10 then plays a little song.



# Example 11 – OLED Display „Hello World“

```
# import libs
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

# initialize I2C Bus
# GPIO 20 - SDA (Data)
# GPIO 21 - SDC (Clock)
i2c = I2C(0, sda=Pin(20), scl=Pin(21))

# OLED display is connected to this bus
# Resolution 128x64 Pixel
oled = SSD1306_I2C(128, 64, i2c)

# clear OLED display and show text
oled.fill(0)
oled.rect(0,0,128,64,1)
oled.text("Attraktor e.V.", 10, 16)
oled.text("Hello World!", 16, 40)
oled.show()
```

In this example we use the OLED display to display message.

The OLED display can display us texts and graphics on our board, whether it's a menu, measurement values, or simply status texts. For this purpose the display is connected to the Pi Pico via the so-called I2C bus. The data to be displayed is transmitted serially over this bus, which is done with one data pin and one clock pin. Our OLED display is connected to the pins GPIO 20 (SDA Data) and GPIO 21 (SDC Clock).

The SSD1306\_I2C library allows us to communicate with the display in a simple way and provides us with the most common functions directly. For example, the function `oled.fill(0)` clears the display, and `oled.text("", x, y)` writes text at a specific x & y position. It is important that you call `oled.show()` at the end to update the display.

You can see all the available functions here:

<https://docs.micropython.org/en/latest/esp8266/tutorial/ssd1306.html>

# Example 12 – Temperature & Air Pressure Sensor BMP280

```
# import libs
from machine import Pin, SPI
from utime import sleep
from bmp280_spi import BMP280SPI

# initialize BMP280 Sensor on SPI Bus
# GPIO 16 - Pico RX <- BMP TX
# GPIO 17 - Chip Select
# GPIO 18 - Clock
# GPIO 19 - Pico TX -> BMP RX
spi = SPI(0, sck=Pin(18), mosi=Pin(19), miso=Pin(16), polarity=1, phase=1)
spi_cs = Pin(17, Pin.OUT, value=1)
bmp280_spi = BMP280SPI(spi, spi_cs)

# read BMP280 Chip ID. Should be 0x58
print(bmp280_spi.chip_id)

# read sensor and print out measurements
while True:
    readout = bmp280_spi.measurements
    print(f"Temperature: {readout['t']} °C, pressure: {readout['p']} hPa.")
    sleep(1)
```

In this example we use the BMP280 sensor for measurements.

The BMP280 sensor can provide us with temperature and air pressure readings and is controlled via a slightly different bus. The SPI bus used here is also a serial bus, like I2C, but it has additional lines and can be clocked faster. For communication, we use the BMP280SPI library here, which makes it easier for us to convert the readings into human-readable formats.

First, we load the libraries again, initialize GPIO pins 16 to 19 and create a bmp280\_spi object with it, which contains all the functionality. We first read out the chip\_id and then read in the measured values every second and display them in the status area in Thonny.

The print output here is somewhat more complex but shows the powerful text formatting capabilities of MicroPython.

# Example 13 – Display Room Climate on OLED Display

```
# import libs
from machine import Pin, I2C, SPI
from utime import sleep
from ssd1306 import SSD1306_I2C
from bmp280_spi import BMP280SPI
```

```
# initialize OLED display on I2C Bus
i2c = I2C(0, sda=Pin(20), scl=Pin(21))
oled = SSD1306_I2C(128, 64, i2c)
oled.fill(0)
oled.show()
```

```
# initialize BPM280 sensor on SPI BUS
spi = SPI(0, sck=Pin(18), mosi=Pin(19), miso=Pin(16), polarity=1, phase=1)
spi_cs = Pin(17, Pin.OUT, value=1)
bmp280_spi = BMP280SPI(spi, spi_cs)
```

```
# endless loop shows measurements on display
while True:
```

```
    oled.fill(0)
    oled.text("Temp & Pressure", 0, 0)
```

```
    readout = bmp280_spi.measurements
    oled.text(f"Tc: {readout['t']:.2f}C", 2,16)
    oled.text(f"Tf: {(readout['t']*1.8+32):.2f}F", 2,26)
    oled.text(f"Tk: {(readout['t']+273.15):.2f}K", 2,36)
    oled.text(f"Pa: {readout['p']:.2f}hPa", 2,46)
```

```
    oled.show()
    sleep(0.25)
```

Here is the grand finale where we use the BMP280 sensor and display it's measurements on the OLED screen. With this, we have gone through all the functions of the demo board and nothing should stand in the way of your own experiments now.

So, how about building a temperature alarm, a simple audio synthesizer or a traffic light?