SCRAPY Manual

Co-funded by the European Union

WARNING

When using this product, please note the following warning points:

1. This product contains many small parts. Swallowing or improper operation can cause serious infections and death. Seek immediate medical attention when the accident happens.
2. Strictly prohibit use of this product and its parts near AC electrical outlet or other circuits in case of the potential risk of electric shock.
3. Strictly prohibit the use of this product near any liquid or fire.
4. Keep conductive materials away from this product.
5. Do not allow children under 3 years old to use this product without adult supervision. Please place this product in a position where children under 3 years old cannot reach.
6. Do not store or use this product in any extreme environments such as extreme hot or cold, high humidity, under direct sunlight, etc.
7. Remember to break the circuit when it is not needed.
8. Some parts of this product may become warm to touch when used in certain circuit designs, which is normal.
9. Improper use may cause overheating.
10. Using components not in accordance with the specification may cause damage to the product.

Introduction

The SCRAPY KIT is built based on the use of the Raspberry Pi Pico Microcontroller. The “SCRAPY KIT” is created under an Erasmus+ Co-funded project.

The new trend in remote education, brought by the COVID-19 pandemic, poses a gap in teaching and practising activities with physical computing reaching students at risk of low performance in such STEM subjects.

The KIT's goal is to support and promote hands-on educational learning in physical computing and programming, in cases of distance teaching and within the classroom setting. The KIT provides a combination of physical computing principles and programming with hardware and software, leading to an innovative learning experience.

The scope of this Manual is to:

- Inform you of the Kit components and the use of the main electronic elements.
- Guide you step by step to effectively assemble the KIT, considering related precautions.
- Provide tutorials for the components' use and connection with the Pico microcontroller.
- Provide tutorials with the functionalities and scope of each electronic component.

Enjoy reading and have fun through hands-on practice and experimentation with the SCRAPY KIT

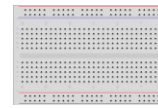
Table of Contents

Included in the SCRAPY KIT

GPIO Breakout Board Raspberry Pi Pico (2pcs)

MB-102 power supply module (1pc)

White breadboard 830 pcs(1pc) + 400 pcs (1pc)



Push Button (1pcs) & Button Cap (1pcs)

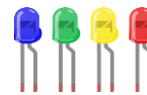
Buzzer (1pc)

Resistors 220 Ohm (5 pcs) + 1k Ohm (5 pcs)

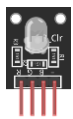
LDR photoresistor (1pc)

100uF capacitor (1pc)

LED 3mm Blue (1pc), Green (1pc), Red(1pc) Yellow (1pc)

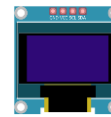


RGB LED 5mm (1pc)

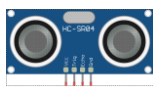


SG90 servo motor (1pc)

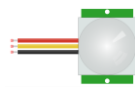
OLED I2C ICC (1pc)



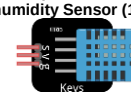
HC-SR04 ultrasonic sensor (1pc)

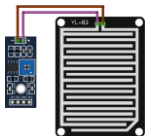
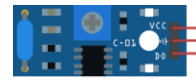
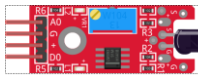


PIR Motion Detector Sensor HC-SR501 (1pc)



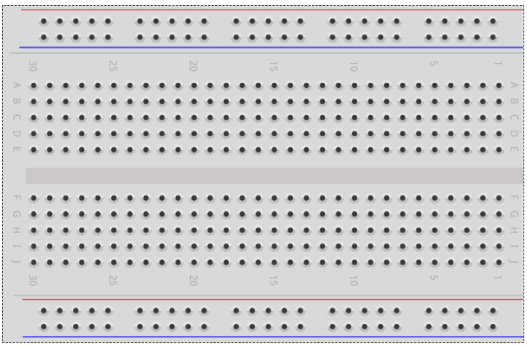
DHT11 digital temperature and humidity Sensor (1pc)



Raindrop sensor (1pc)	**Rotary Potentiometer Linear B1k Ohm (1pc)**	**SW-420 vibration sensor (1pc)**
		
Flame sensor (1pc)	**High Sensitivity Sound Detection Sensor (1pc)**	**Soil hygrometer / moisture detection Sensor (1 pc)**
		
Infrared IR sensor module KY-032 (1pc)	**Joystick module (1pc)**	**USB to micro-USB cable 1m (1pc)**
		
6 x AA 1.5V batteries (1pc) + holder (1pc)	**Jumper cables (6 pcs)**	**Plastic screws (10 pcs) , plastic nuts (6 pcs), plastic pillars (10 pcs)**
		

Components Explanation

1. What is a breadboard?



A breadboard is a plastic board with tiny holes in it which permit the easy insert of electronic components (transistors, resistors, chips, etc.) to prototype (build and test) an electronic circuit. The inside is made up of rows of tiny metal clips to hold the leads to be connected.

Most breadboards have rows of numbers, letters, and plus and minus signs written on them. The purpose of the labels is to help you locate certain holes on the breadboard so you can follow directions when building a circuit.

The long strips at the 2 sides of the breadboard are usually marked with red and blue or red and black and with plus (+) and minus (-) signs respectively. These rows are called the buses or rails and are typically used to supply electrical power to the circuit when connected to a power supply (battery pack or external supply).

The Positive “bus” is marked in red, has the plus sign (+) and provides the power.

The Negative “bus” is marked in blue or black, has the minus sign (-) and provides the ground.

Advantages of using a Breadboard:

- Makes easier to quickly check simple and complex circuits and to easily verify circuits at their initial stage.
- Easy to adjust.
- Flexible.
- No drilling holes.
- No soldering required.
- Easy debugging of circuits and programs.

2. What is a resistor?

A resistor is a little package of resistance. Using it into a circuit reduces the current by a precise amount. To figure out the resistance of a resistor there is a pattern of coloured bands.

Color	1st Band	2nd Band	3rd Band (5-Band Only)	Multiplier (3rd or 4th Band)	Tolerance (Last Band)
Black	0	0	0	1	
Brown	1	1	1	10	± 1%
Red	2	2	2	100	± 2%
Orange	3	3	3	1000	
Yellow	4	4	4	10000	
Green	5	5	5	100000	± 0.5%
Blue	6	6	6	1000000	± 0.25%
Violet	7	7	7	10000000	± 0.1%
Grey	8	8	8		± 0.05%
White	9	9	9		
Gold				0.1	± 5%
Silver				0.01	± 10%
None					± 1%

(Image credit: Future Owns) available at https://www.tomshardware.com/how-to/resistor-color-codes

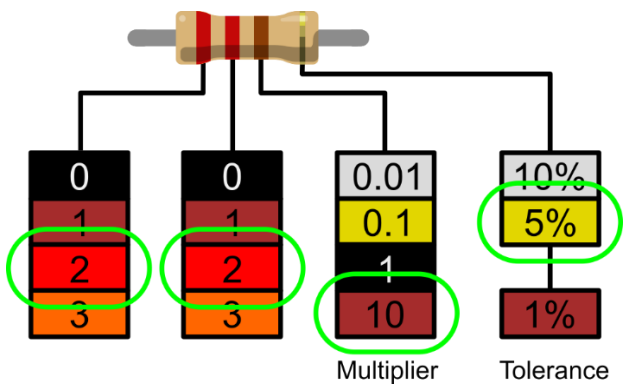
Common Resistor Colour Codes and their uses:

Resistor type	4-Band Colour Code	5-Band Colour Code	Common Uses
220Ohm	Red-Red-Brown-Gold	Red-Red-Black-Black-Gold	LED Light Protection
1K Ohm (1Kiloohm)	Brown-Black-Red-Gold	Brown-Black-Black-Brown-Gold	LED Protection, Voltage Divider

Resistors have no polarity, so they can be used in any orientation in a circuit. But to identify the correct resistor colour code values we need to understand the coloured bands on the resistor.

On a typical four-band hobby level resistor, there are three colours in a group. These are the first, second significant figures and the multiplier. The final band is the resistor's tolerance, a margin of error if you will. For the majority of hobbyists, a tolerance of 5% (Gold) is perfect and common.

220 Ohm Resistor (4-Band)

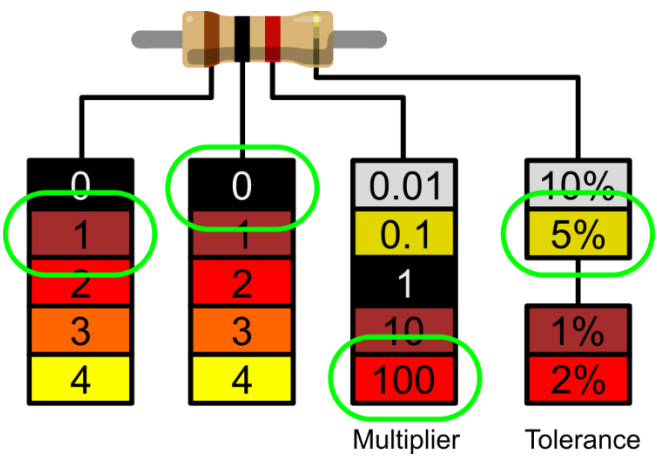


(Image credit: Future) available at https://www.tomshardware.com/how-to/resistor-color-codes

1. The first significant figure is red and using the decoder we can see that red has a value of 2.
2. The second significant figure is also red, so that gives us 22.
3. The multiplier is brown, and this decodes to 10. If we multiply 22 by 10, we get 220.
4. The final band, tolerance, is gold. Gold is 5%, which means we can accept a resistance with a 5% margin of error.

For makers requiring greater precision there are also five band resistors which have a third significant figure. The extra figure provides clarity which can be essential in circuits sensitive to resistance for example scientific and engineering instruments.

1K Ohm Resistor (4-Band)



(Image credit: Tom's Hardware) available at https://www.tomshardware.com/how-to/resistor-color-codes

1. The 1st line is brown, and using the decoder, we can see that the value is 1.
2. The 2nd line is black, so that gives us 10.

3. The multiplier is red, and this decodes to 100. If we multiply 10 by 100 we get 1000.
4. The final band, tolerance, is gold. Gold is 5%, which means we can accept a resistance with a 5% margin of error.

3. What is a capacitor?

A capacitor is a device that stores electrical energy in an electric field. It is a passive electronic component with two terminals. It consists of two electrical conductors that are separated by a distance. The space between the conductors may be filled by vacuum or with an insulating material known as a dielectric. (Wikipedia)

The 100uF capacitor is an Electrolytic decoupling capacitor. These capacitors are great transient/surge suppressors and using one between the power and ground of the circuit ensures ensure smooth power delivery.

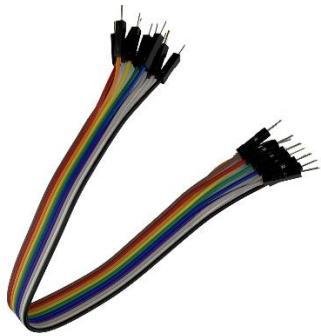
4. What is a diode?

A diode is a two-terminal electronic component that conducts current primarily in one direction (asymmetric conductance); it has low (ideally zero) resistance in one direction, and high (ideally infinite) resistance in the other.

The most common function of a diode is to allow an electric current to pass in one direction (called the diode's forward direction), while blocking it in the opposite direction (the reverse

direction). As such, the diode can be viewed as an electronic version of a check valve. This unidirectional behaviour is called rectification and is used to convert alternating current (ac) to direct current (dc). As rectifiers, diodes can be used for such tasks as extracting modulation from radio signals in radio receivers. (Wikipedia https://en.wikipedia.org/wiki/Diode)

5. What is a jumper-cable?



Jumper cables/ wires are simply wires that have connector pins at each end, allowing them to be used to connect two points to each other without soldering. Jumper wires are typically used with breadboards and other prototyping tools in order to make it easy to change a circuit as needed. The colour variation of the wires can be used as advantage in order to differentiate between types of connections, such as ground or power.

Jumper wires typically come in three versions: male-to-male, male-to-female and female-to-female. The difference between each is in the end point of the wire. Male ends have a pin projecting and can plug into things, while female ends do not and are used to plug things into. Male-to-male jumper wires are the most common. When connecting two ports on a breadboard, a male-to-male wire is mostly required. (https://blog.sparkfuneducation.com/what-is-jumper-wire)

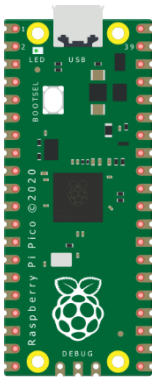
Project Preparation

1. Readme Before Using

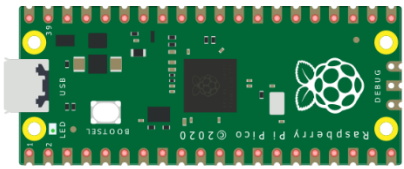
NOTE: Since the experiments involved are all circuit experiments, wrong connection or short circuit may damage your Pico development board. Please, always check the circuit again before connecting the power supply.

2. Raspberry Pi Pico

This is the Raspberry Pi Pico:

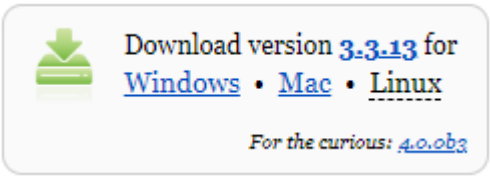


Plug the micro-USB cable into the port on the left-hand side of the board.



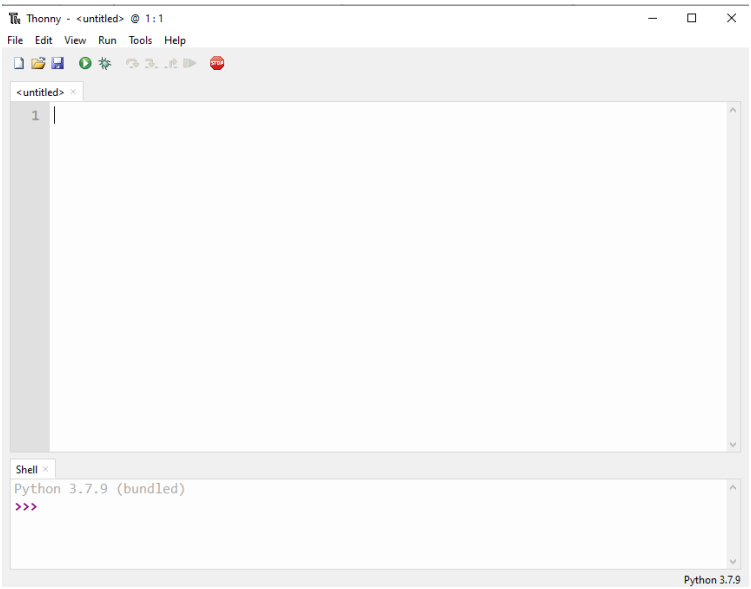
3. Install Thonny IDE

Visit https://thonny.org and choose the appropriate operating system. Follow the instructions to complete the installation.



In this manual, all tutorials are programmed in Windows 10, using a Raspberry Pi Pico and the appropriate firmware.

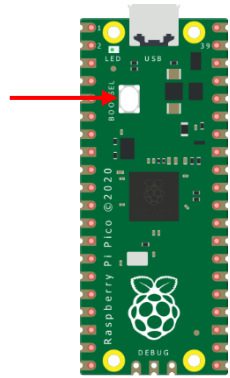
After installation is completed, open Thonny from your computer.



4. Firmware installation

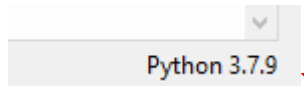
The Raspberry Pi Pico can be programmed using a Python variant, called MicroPython. To use MicroPython on the Pico, first you need to install its firmware.

Step 1: Find the BOOTSEL button on your Raspberry Pi Pico.

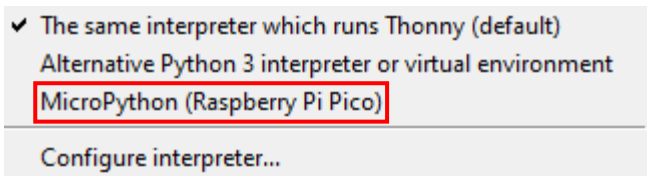


Step 2: Press the BOOTSEL button and hold it while you connect the other end of the micro-USB cable to your computer.

Step3: In the bottom right-hand corner of Thonny you will see the version of Python you currently use.

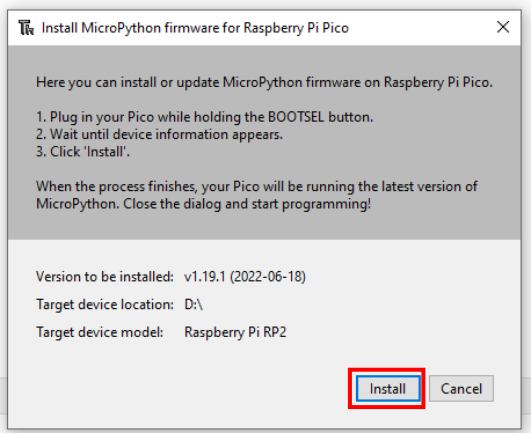


Click on the Python version and choose the MicroPython (Raspberry Pi Pico)

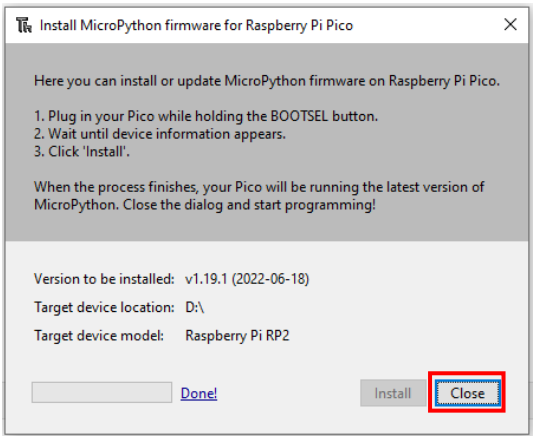


If you don't see this option, make sure the cable is connected properly on the Pico and/or your computer.

Step 4: A dialog box will appear, asking you to install the latest firmware version to your Pico. Click the **Install** button to copy the firmware to your Pico.



Step 5: Wait for the installation to complete and click **Close**.



You don't need to repeat the process every time you connect the Raspberry Pi Pico to your computer, so next time, just plug it in and you are good to go.

5. Introduction to MicroPython Programming

You will now use Thonny IDE to run some simple Python code to get acquainted with Thonny's Shell and MicroPython.

First make sure that your Raspberry Pi Pico is connected to your computer, and you have selected the MicroPython interpreter as explained in the previous section.

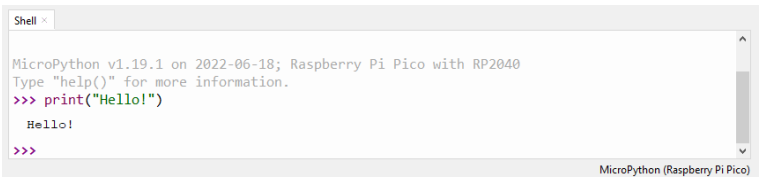
The Shell panel at the bottom of Thonny editor should look like this:

Thonny is ready to communicate with your Pico using REPL (read-eval-print loop) framework, which allows you to write code directly at the Shell and get output.

Type the following command:

```
print("Hello!")
```

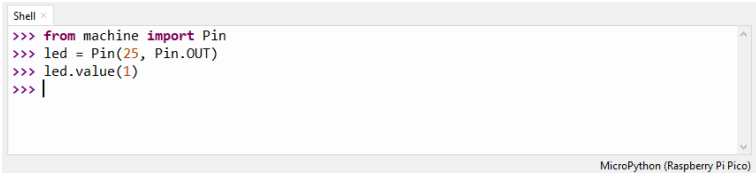
Then hit the Enter key and see the following output:



MicroPython allows you to add hardware-specific modules, such as `machine`, that you can use to program your Pico. In the following example you will use the `machine` module to turn on Pico's onboard LED.

Write the following code in Thonny's Shell:

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



Press the Enter key and immediately Pico's onboard LED will turn on.

To turn off the LED write the following code:

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

For the rest of this section, you will write your first "real" program that will make the onboard LED to blink every time you run your program.

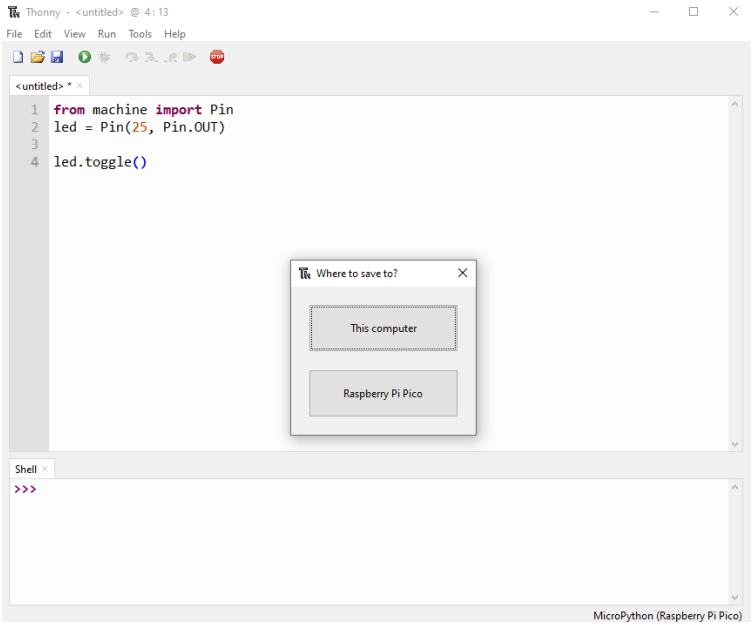
Thonny Shell is useful to try out quick commands and make sure everything is working correctly. However, longer programs

should be saved in a .py file. Using Thonny, you can save programs directly to the Raspberry Pi Pico and then run them.

Open Thonny Python and on the main editor pane write the following code:

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

Now, save your program by clicking the Save icon on the top left-hand side, or by pressing Ctrl+S on your keyboard.



Thonny will ask you where you want your program to be saved. Choose the **Raspberry Pi Pico**. Save the file as *blink.py* and click **OK**. You always need to add the .py extension so that Thonny recognises the file as a Python file.

Now, every time you click the Play icon, you should see the onboard LED switching ON and OFF.

Taking your code one step further, you can make the onboard LED blinking at a certain pace.

Write the following code and save your program using the same name as above.

```
from machine import Pin
from time import sleep
led = Pin(25, Pin.OUT)
while True:
      led.toggle()
      sleep(1)
```

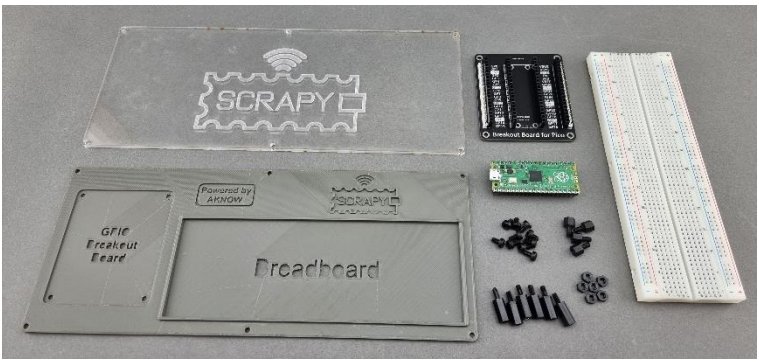
Now when you run the program, the onboard LED will blink every 1 second until we stop the program. To stop the program you can click on the STOP icon or press Ctrl+C on your keyboard.

In future tutorials, we will learn how to add and control other electronics and sensors and create programs that can communicate with each other.

Kit Assembly

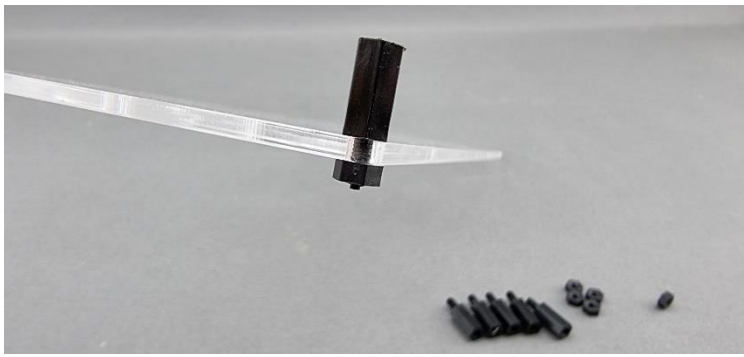
The SCRAPY Kit comprises the following items:

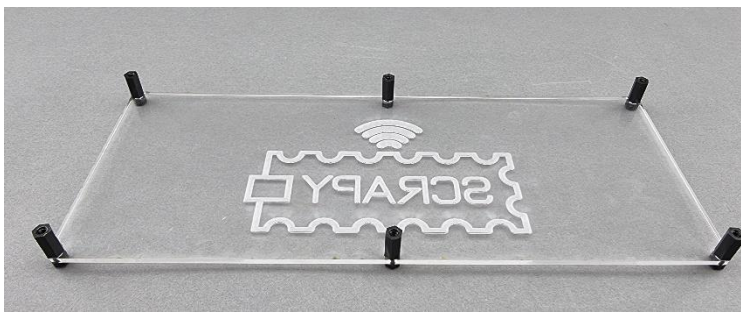
- 1x 3D printed piece
- 1 x Plexiglass piece
- 1 x Raspberry Pi Pico
- 1 x GPIO breakout board
- 1 x Breadboard (830pcs)
- 10 x Plastic screws
- 6 x 12mm plastic pillars
- 4 x 6mm plastic pillars
- 6 x Plastic nuts



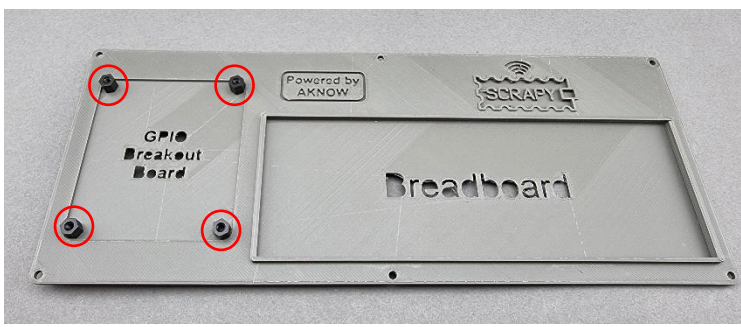
The assembly procedure is simple and can be completed in 6 steps:

Step 1: Mount the 6 x 12mm plastic pillars with the 6 x plastic nuts on the plexiglass piece.

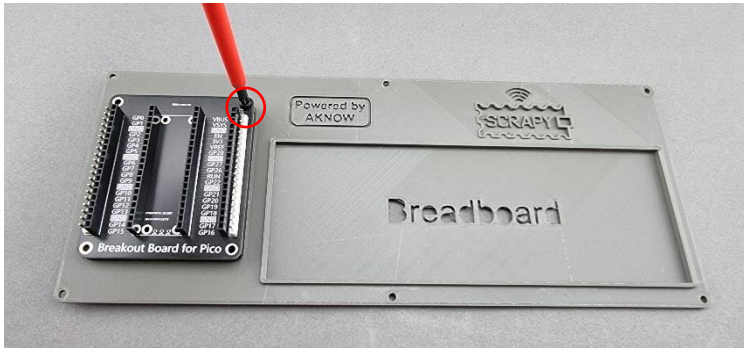


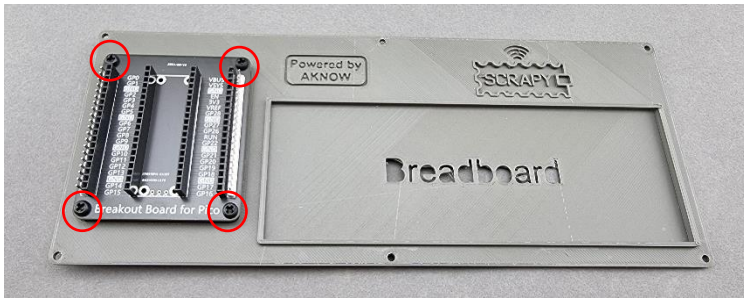


Step 2: On the 3D printed piece, place 4 x 6mm plastic pillars at the “GPIO Breakout Board” section.

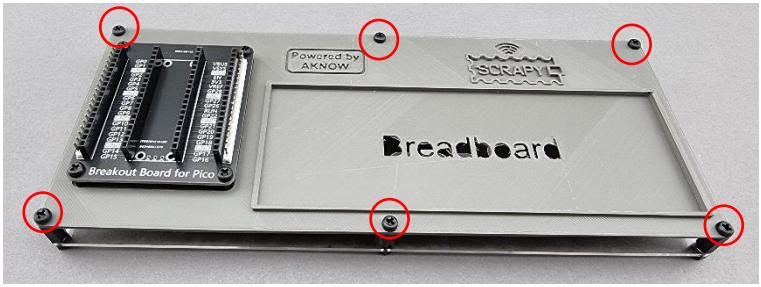


Step 3: Mount the GPIO breakout board on the 4 pillars using 4 plastic screws.

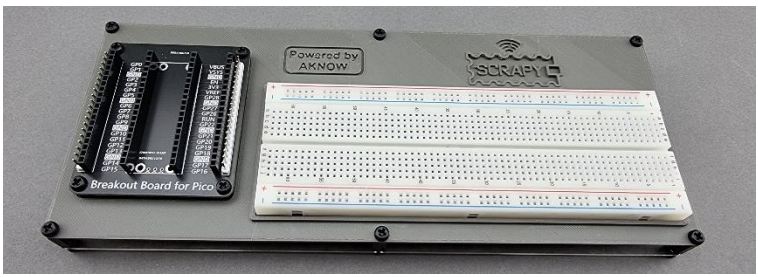




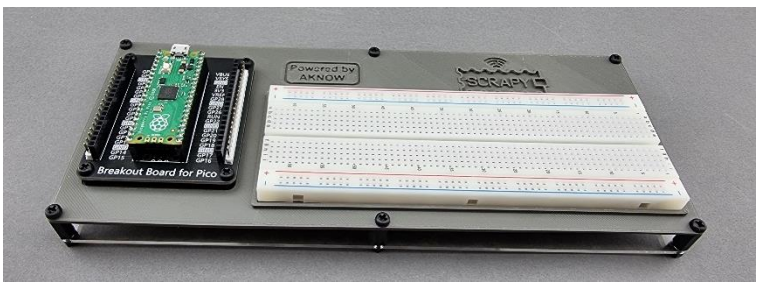
Step 4: Mount the 3D printed object with the plexiglass object using 6 plastic screws on the 6 plastic pillars you mounted on Step 1.



Step 5: Pill off the protective tape underneath the 830 pcs breadboard and place it on the “Breadboard” section of the Kit. Please make sure, the positive (+) side of the breadboard is on top as the following image indicates.



Step 6: Place the Raspberry Pi Pico on top the GPIO Breakout Board and press against it until all GPIO pins have been inserted correctly. Make sure the micro-USB slot in on the top side as the image indicates.



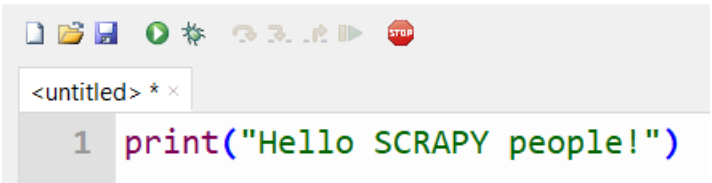
Basic Tutorials

0. “Hello SCRAPY people!”

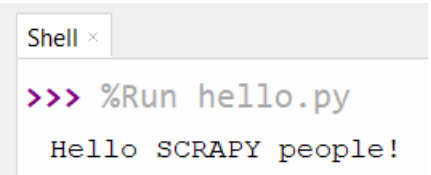
In this basic tutorial, we are going to learn how to print a simple message in Thonny using Python programming.

The print() function

1. Go to the Thonny editor, write the following code and press the play icon. Thonny will ask you to save your program first. Save it under the name hello.py.

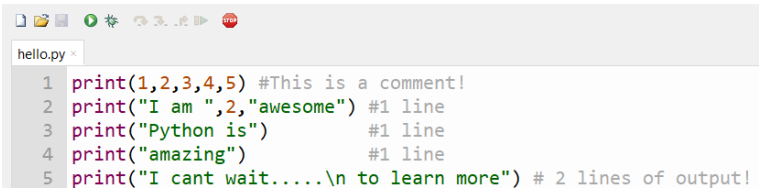


2. Check the shell window.

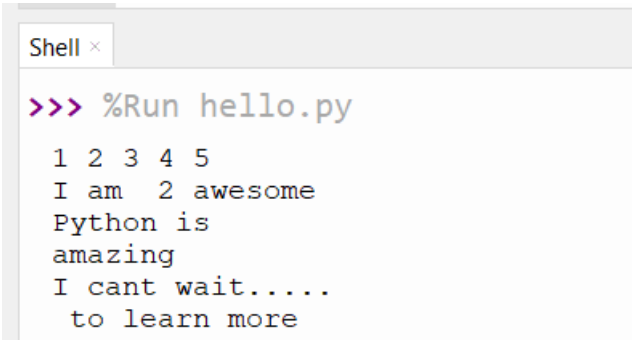


3. Good job! You’ve just created your first Python program.

The print function is a built in Python function that lets us print text in the shell. It can also take parameters. Create a new program and copy the following code, then press play and see how the text appears in the shell.



As you can see from the shell window, each print function prints text on a separate line. However, if you use the “\n” (newline character) you can change line in the same print statement.



Exercise

Use the print function to print in 3 separate lines “Goodbye SCRAPY people!” using only one print statement.

4. Control an LED

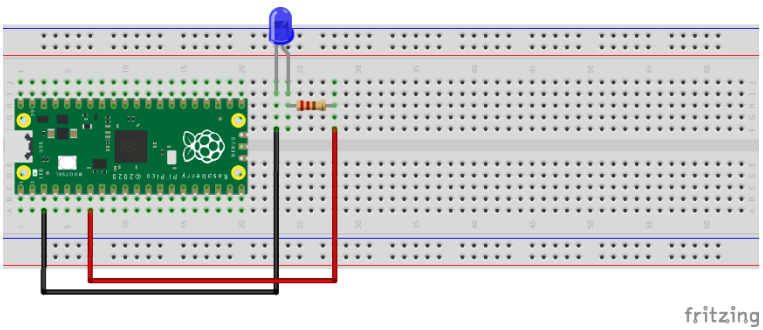
Description

In this tutorial you will learn how to connect and control an LED light. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `led.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x LED (any colour)
- 1 x Pico breadboard kit
- 2 x Male-to-male jumper wires
- 1 x 220 Ohm resistor

Wiring diagram



– connect the longer end (+) of the LED to a 220Ohm resistor

– connect the resistor to GPIO5 (red cable)

– connect the shorter end (-) of the LED to a GND pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /led.py @ 9 : 1

File Edit View Run Tools Help

[led.py]

```
1  from machine import Pin #import  Pin from library machine
2  from time import sleep  #import sleep from library time
3
4  led = Pin(5, Pin.OUT)
5
6  while True:
7      led.toggle()
8      sleep(1)
9
```

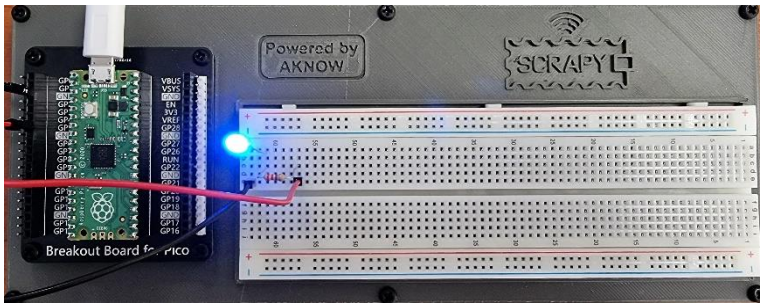
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the circuit looks using the provided hardware:



5. Push Button

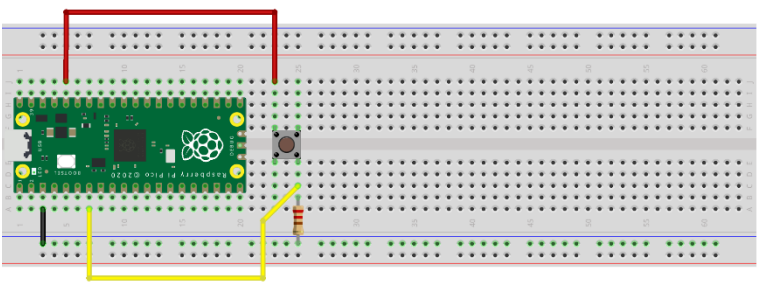
Description

In this tutorial you will learn how to connect and control a push button. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `button.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Push button (any colour)
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x 220 Ohm resistor
- 1 x Button cap

Wiring diagram



– connect the top left button side to the 3v3 pin (red cable)

– connect the bottom right button side to GPIO5 (yellow cable)

– connect a GND pin to the (-) rail (black cable)

– connect the 220 Ohm resistor to the (-) rail and the bottom right button side

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /button.py @ 10 : 1

File Edit View Run Tools Help

[button.py]

```
1  from machine import Pin #import Pin and ADC from library machine
2  from time import sleep  #import sleep from library time
3
4  button = Pin(5, Pin.IN, Pin.PULL_DOWN)
5
6  while True:
7      if button.value():
8          print("Button is pressed!")
9          sleep(1)
10
```

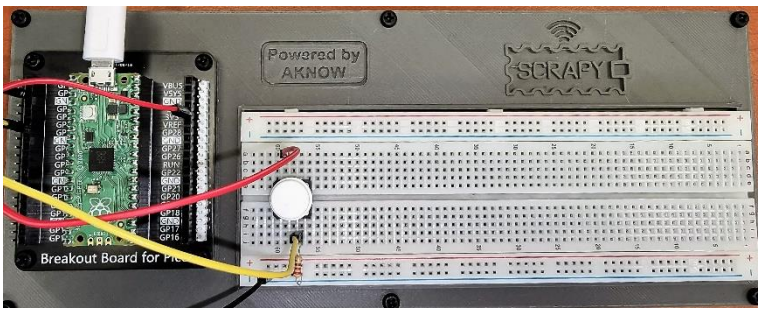
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the circuit looks using the provided hardware:



6. Buzzer

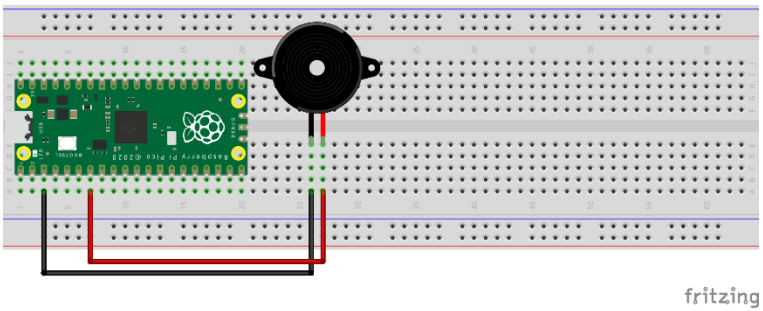
Description

In this tutorial you will learn how to connect and control a buzzer. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `buzzer.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 2 x Male-to-male jumper wires
- 1 x Buzzer

Wiring diagram



– connect the longer end (+) of the buzzer to GPIO5 pin

– connect the shorter end (-) of the buzzer to a GND pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /buzzer.py @ 16 : 1

File Edit View Run Tools Help

[buzzer.py]

```
1  from machine import Pin #import Pin from library machine
2  from time import sleep  #import sleep from library time
3
4  buzzer = Pin(5, Pin.OUT)
5
6  buzzer.value(0)
7  sleep(0.5)
8  buzzer.value(1)
9  sleep(0.5)
10 buzzer.value(0)
11 sleep(0.5)
12 buzzer.value(1)
13 sleep(0.5)
14 buzzer.value(0)
15 sleep(0.5)
16
```

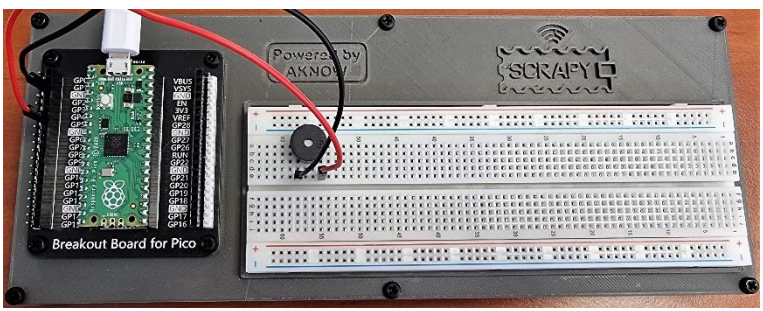
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the circuit looks using the provided hardware:



7. Potentiometer

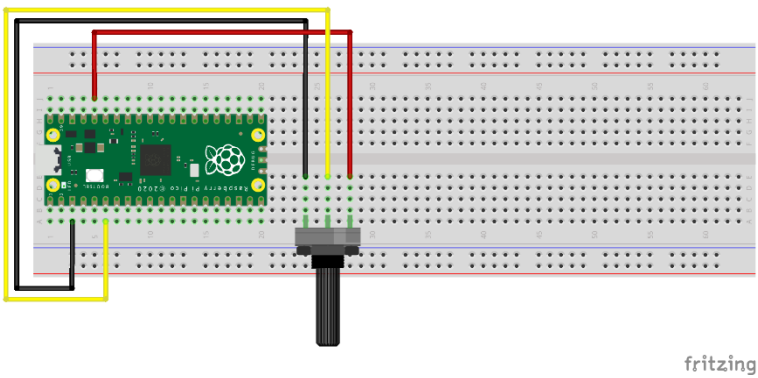
Description

In this tutorial you will learn how to connect and control potentiometer. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `pot.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x Potentiometer

Wiring diagram



– black cable should be connected to GND (Pin 3) pin

– yellow cable should be connected to GPIO5 pin

– red cable should be connected to 3V3 power pin

– turn the potentiometer to the left so it's off

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /pot.py @ 9 : 1

File Edit View Run Tools Help

[pot.py]

```
1  from machine import Pin, ADC #import Pin from library machine
2  from time import sleep  #import sleep from library time
3
4  pot = ADC(Pin(26))
5
6  while True:
7      print(pot.read_u16())
8      sleep(0.5)
9
```

Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

MicroPython (Raspberry Pi Pico)

Sample picture

Image of how the circuit looks using the provided hardware:

8. RGB LED

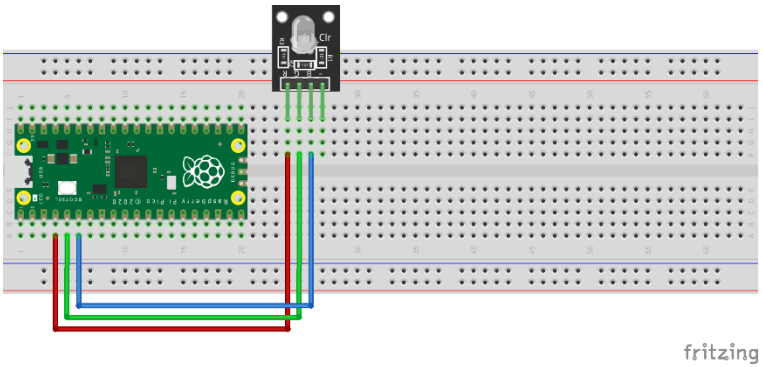
Description

In this tutorial you will learn how to connect and control an RGB LED module light. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `rgb.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 4 x Male-to-male jumper wires
- 1 x RGB LED module

Wiring diagram



– red cable should be connected to GPIO2

– green cable should be connected to GPIO3

– blue cable should be connected to GPIO4

– no GND connection is required

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /rgb.py @ 14 : 1

File Edit View Run Tools Help

[rgb.py]

```
1  from machine import Pin, PWM #import Pin from library machine
2  from time import sleep  #import sleep from library time
3  import random
4  Led_R = PWM(Pin(2))
5  Led_G = PWM(Pin(3))
6  Led_B = PWM(Pin(4))
7  # Define the frequency
8  Led_R.freq(2000)
9  Led_G.freq(2000)
10 Led_B.freq(2000)
11 while True:
12     # range of random numbers
13     R=random.randint(0,65535)
14     G=random.randint(0,65535)
15     B=random.randint(0,65535)
16     print(R,G,B)
17     Led_R.duty_u16(R)
18     Led_G.duty_u16(G)
19     Led_B.duty_u16(B)
20     sleep(1)
```

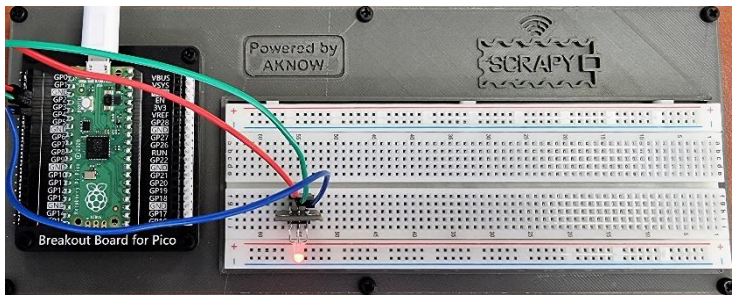
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the circuit looks using the provided hardware:



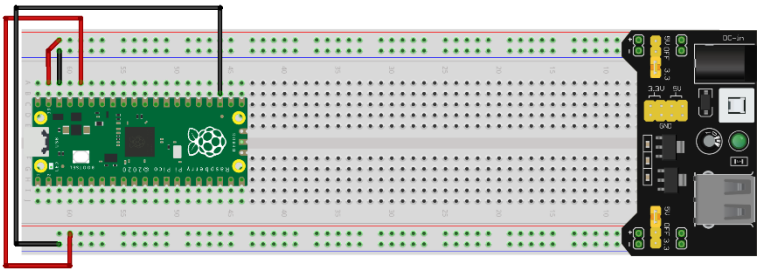
Advanced Tutorials

Before proceeding to this section and the next, we need to make a few modifications to our SCRAPY Kit. This involves the addition of a few extra components and their connectivity.

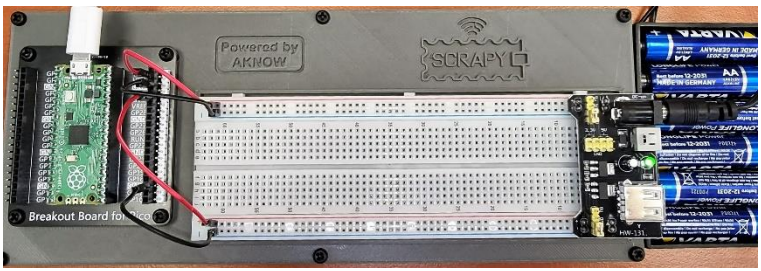
Components

- 1 x 6-AA battery pack
- 1 x MB-102 power module
- 4 x Male-to-male jumper wires

Please follow the schematic to connect the new components:



Circuit



– this setup will be used for the remaining tutorials

– top side connections: VSYS 5V ((+) red) and GND ((-) black)

– bottom side connections: 3V3 ((+) red) and GND ((-) black)

9. LDR Photoresistor

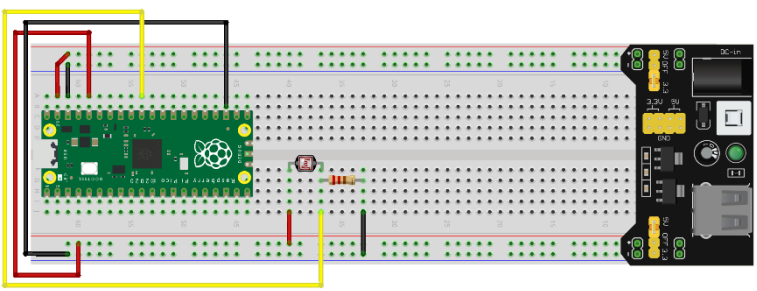
Description

In this tutorial you will learn how to connect and control an LDR photoresistor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `ldr.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x 220Ohm resistor
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x LDR photoresistor

Wiring diagram



– left side of LDR is connected to 3V rail ((+) red)

– right side of LDR is connected to a 220 Ohm resistor and to GPIO26 ADC (yellow cable) pin

– right side of resistor is connected to GND rail ((-) black)

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /ldr.py @ 10 : 1

File Edit View Run Tools Help

[ldr.py]

```
1  from machine import Pin, ADC #import Pin and ADC from library machine
2  from time import sleep  #import sleep from library time
3
4  ldr = ADC(Pin(26))
5
6
7  while True:
8      print(ldr.read_u16())
9      sleep(1)
10
```

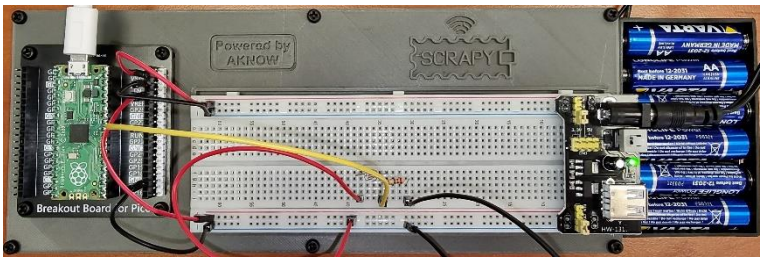
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  8642
  7329
  4032
  3504
  3840
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



10. Servo motor

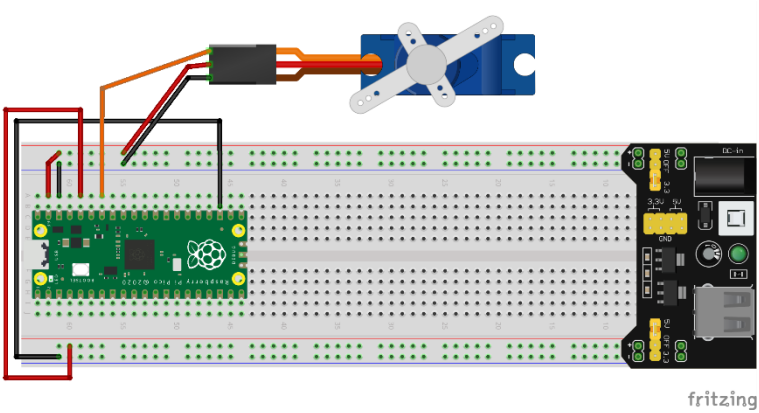
Description

In this tutorial you will learn how to connect and control a servo motor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `servo.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x SG-90 servo motor

Wiring diagram



– red cable is connected to 5V rail (+)

– black/brown cable is connected to GND rail (-)

– orange cable is connected to GPIO28 ADC pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /servo.py @ 13 : 20

File Edit View Run Tools Help

servo.py

```
from machine import Pin, PWM #import Pin and PWM from library machine
from time import sleep  #import sleep from library time

pwm = PWM(Pin(28))
pwm.freq(50)

while True:
    for position in range(1000,9000,50):
        pwm.duty_u16(position)
        sleep(0.01)
    for position in range(9000,1000,-50):
        pwm.duty_u16(position)
        sleep(0.01)
```

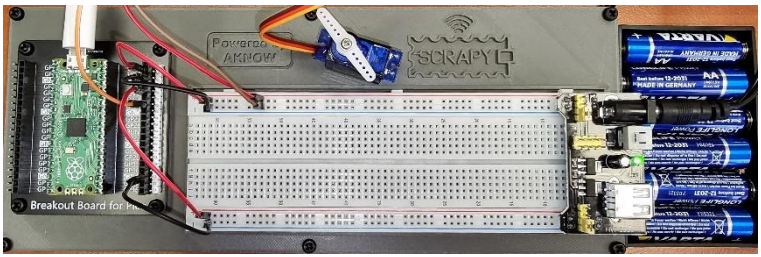
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



11. OLED I2C SSD1306 display

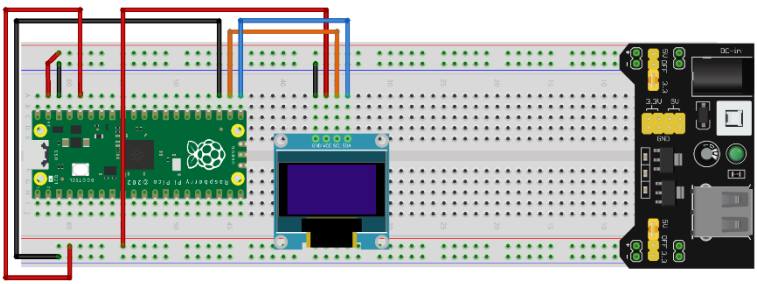
Description

In this tutorial you will learn how to connect and control the I2C ICC OLED display. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `oled.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 4 x Male-to-male jumper wires
- 1 x OLED I2C SSD1306 display

Wiring diagram



– red cable is connected to 3v3 rail (+)

– black cable is connected to GND rail (-)

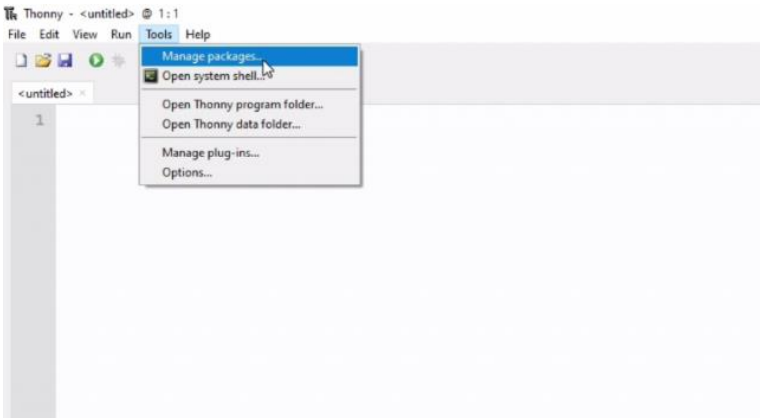
– orange cable is connected to GPIO17 I2C0 SCL pin

– blue cable is connected to GPIO26 I2C0 SDA pin

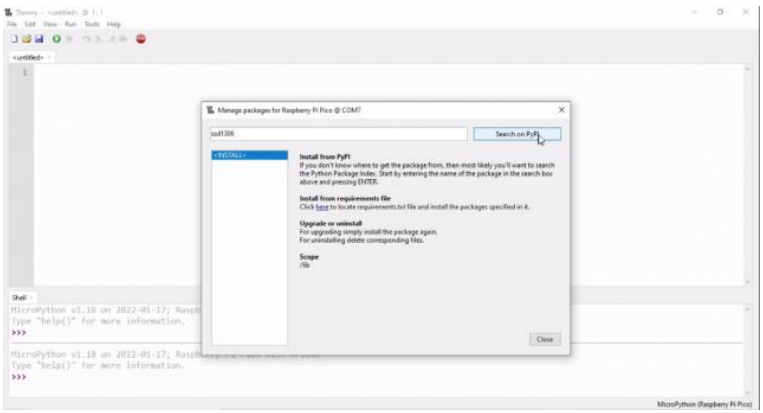
Code

Before starting programming, the OLED display, we first need to add the SSD1306 package to our RPi Pico. To do that, please follow the next steps:

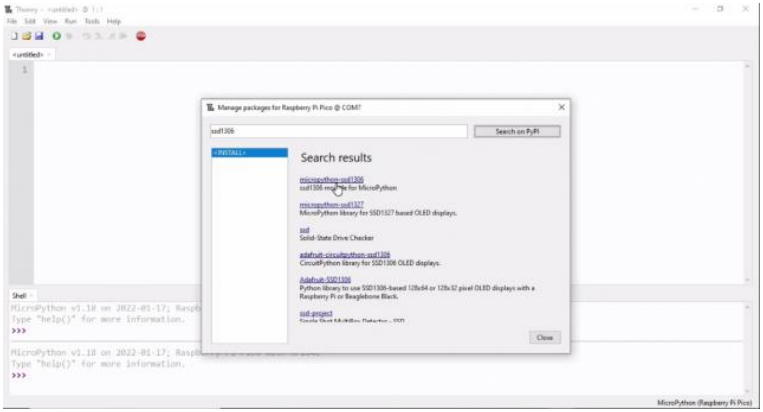
1. Open Thonny and go to **Tools → Manage packages…**



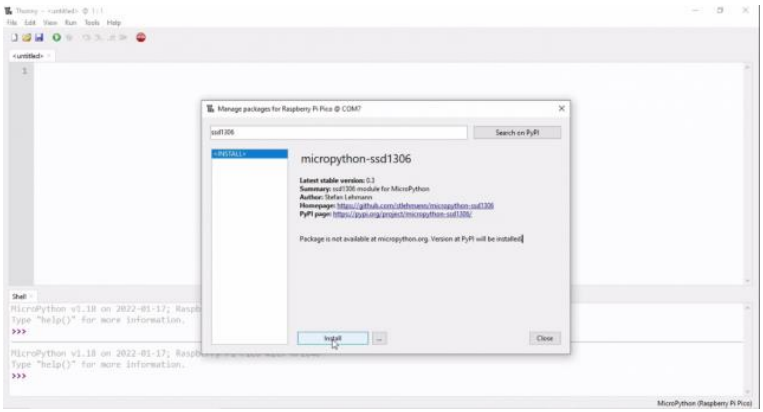
2. In the manage packages window, type **SSD1306** and click *Search*.



3. Once search is over, click on the *micropython-ssd1306.*



4. On the next window, click *Install.*



5. Wait for package installation, then click close.

Now, we are ready to proceed with programming the OLED display.

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /oled.py @ 20 : 1

File Edit View Run Tools Help

[oled.py]

```
from machine import Pin, I2C
import ssd1306

WIDTH =128
HEIGHT = 64

PIN_SCL = 17
PIN_SDA = 16

i2c = I2C(0,scl=Pin(PIN_SCL),sda=Pin(PIN_SDA))
oled = ssd1306.SSD1306_I2C(WIDTH,HEIGHT,i2c)
oled.fill(0)

oled.text("OLED", 0, 0)
oled.text("Display", 0, 10)
oled.text("Hello SCRAPY", 0, 30)
oled.text("  People!", 0, 40)

oled.show()
```

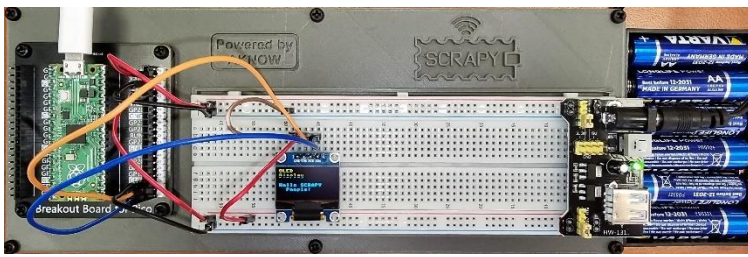
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
>>>
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



12. Joystick module

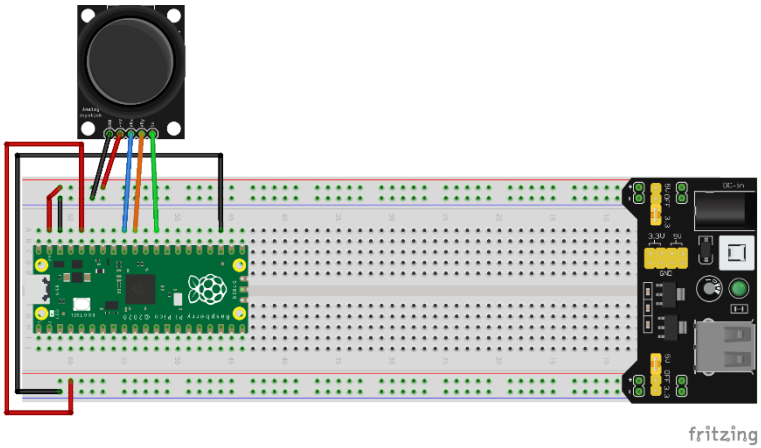
Description

In this tutorial you will learn how to connect and control a joystick module. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `joystick.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 5 x Male-to-male jumper wires
- 1 x Joystick module

Wiring diagram



– +5V (red cable) is connected to 5V rail (+)

– GND (black cable) is connected to GND rail (-)

– VRx (blue cable) is connected to GPIO27 ADC1 pin

– VRy (orange cable) is connected to GPIO26 ADC0 pin

– SW (green) cable is connected to GPIO22 pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /joystick.py @ 18 : 5

File Edit View Run Tools Help

[joystick.py]

```
from machine import Pin, ADC
from time import sleep

VRX = ADC(Pin(27))
VRY = ADC(Pin(26))
SW = Pin(22,Pin.IN, Pin.PULL_UP)

while True:
    xAxis = VRX.read_u16()
    yAxis = VRY.read_u16()
    switch = SW.value()

    print("X-axis: " + str(xAxis) + ", Y-axis: " + str(yAxis) + ", Switch " + str(switch))
    if switch == 0:
        print("Push button pressed!")
    print(" ")
    sleep(1)
```

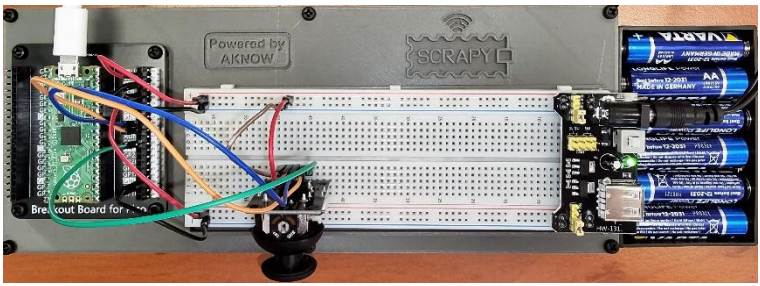
Shell

```
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
  X-axis: 32792, Y-axis: 34616, Switch 1

  X-axis: 32471, Y-axis: 34520, Switch 1

  X-axis: 32455, Y-axis: 34472, Switch 1
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



Tutorials with Sensors

13. Raindrop sensor

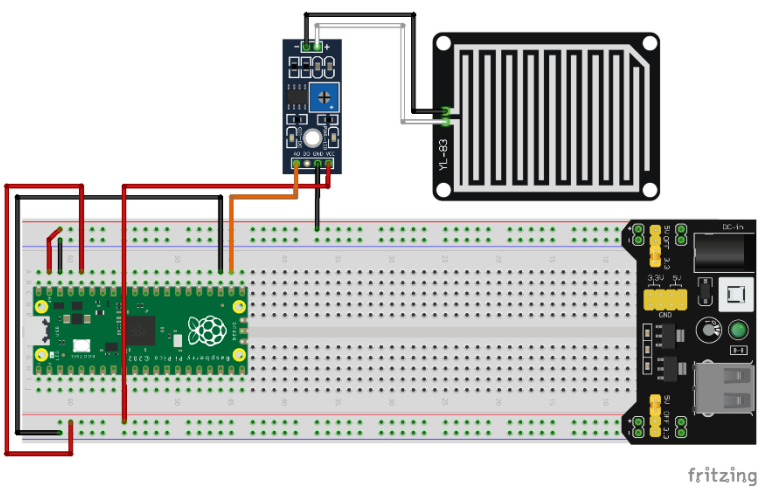
Description

In this tutorial you will learn how to connect and control the raindrop sensor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `raindrop.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x Raindrop sensor

Wiring diagram



– 3v3V (red cable) is connected to 3v3V rail (+)

– GND (black cable) is connected to GND rail (-)

– DO (orange cable) is connected to GPIO17 pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /raindrop.py @ 15 : 1

File Edit View Run Tools Help

[raindrop.py]

```
1  from machine import Pin
2  from time import sleep
3
4  PIN_RAIN = 17
5
6  rainSensor = Pin(PIN_RAIN, Pin.In)
7
8  while True:
9      if rainSensor.value() == 1:
10         print("Not raining")
11         sleep(0.5)
12     if rainSensor.value() == 0:
13         print("It's raining!")
14         sleep(0.5)
15
```

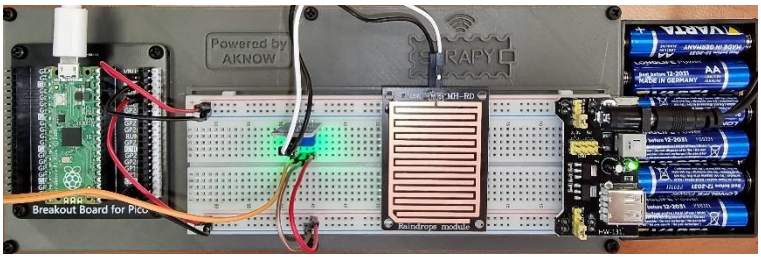
Shell

```
MicroPython v1.20.0 on 2023-04-26; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



14. HC-SR04 Ultrasonic Sensor

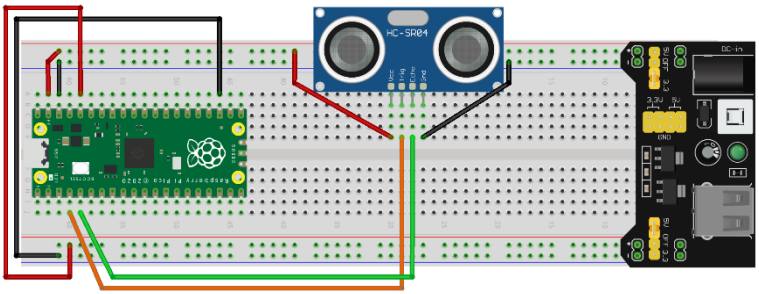
Description

In this tutorial you will learn how to connect and control the HC-SR04 ultrasonic sensor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `ultrasonic.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x HC-SR04 Ultrasonic sensor

Wiring diagram



– VCC (red cable) is connected to 5V rail (+)

– GND (black cable) is connected to GND rail (-)

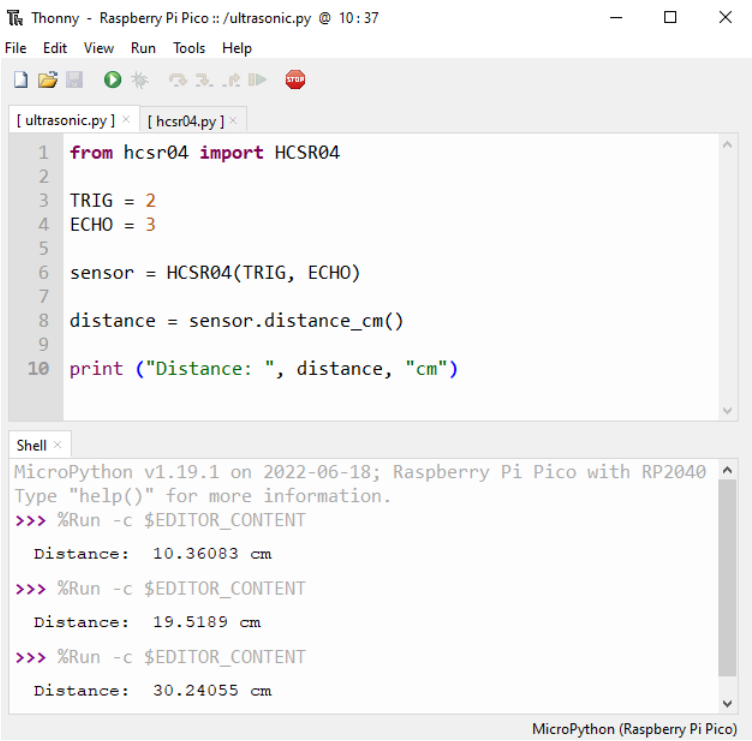
– TRIG (orange cable) is connected to GPIO2 pin

– ECHO (green cable) is connected to GPIO3 pin

Code

To use the HC-SR04 Ultrasonic sensor we can develop our own program, or use one of the available libraries existing online, e.g., on Github. If you choose to download the *hcsr04.py* library, you should save it in your Pico under that same name.

MicroPython code using an existing library:



MicroPython code without existing library:

Thonny - Raspberry Pi Pico :: /ultrasonic.py @ 21 : 18

File Edit View Run Tools Help

[ultrasonic.py]

```
from machine import Pin
import utime

TRIG = Pin(2, Pin.OUT)
ECHO = Pin(3, Pin.IN)
def ultra():
    TRIG.low()
    utime.sleep_us(2)
    TRIG.high()
    utime.sleep_us(5)
    TRIG.low()
    while ECHO.value() == 0:
        signaloff = utime.ticks_us()
    while ECHO.value() == 1:
        signalon = utime.ticks_us()
    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    print("The distance from object is ",distance,"cm")
while True:
    ultra()
    utime.sleep(1)
```

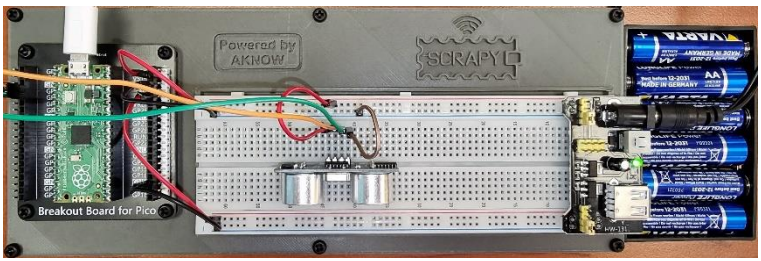
Shell

```
The distance from object is  153.9212 cm
The distance from object is  12.91395 cm
The distance from object is  22.8095 cm
The distance from object is  155.0874 cm
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



15. PIR Motion Sensor

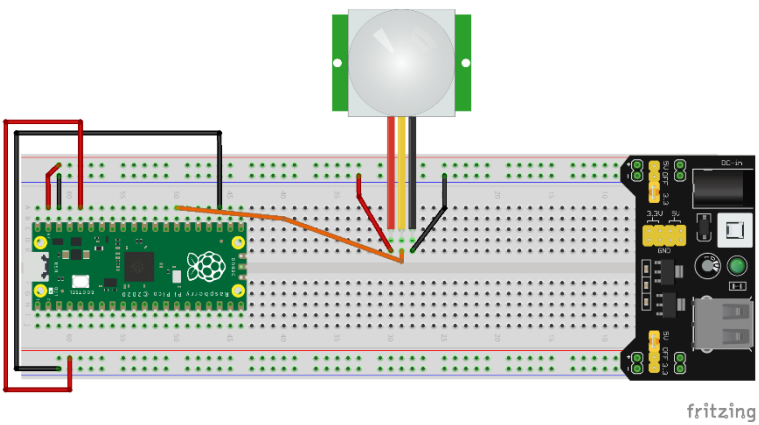
Description

In this tutorial you will learn how to connect and control the PIR motion sensor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `motion.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-female jumper wires
- 1 x PIR motion sensor

Wiring diagram



– VCC (red cable) is connected to 5V rail (+)

– GND (black cable) is connected to GND rail (-)

– OUT (orange cable) is connected to GPIO21 pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /motion.py @ 6 : 12

File Edit View Run Tools Help

[motion.py]

```
1  from machine import Pin
2  from time import sleep
3
4  PIR = Pin(21, Pin.IN, Pin.PULL_UP)
5
6  while True:
7      if PIR.value() == 1:
8          print("Motion detected!")
9          sleep(1)
10     if PIR.value() == 0:
11         print("No motion detected!")
12         sleep(1)
```

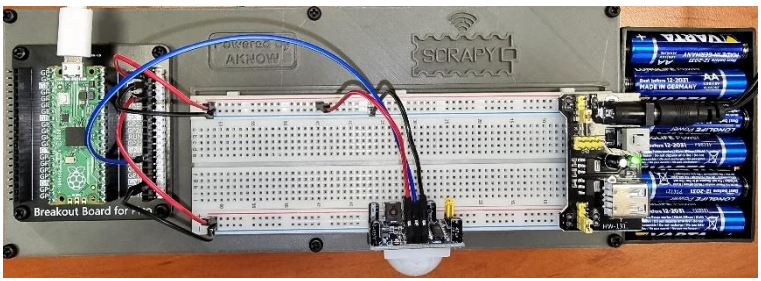
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  No motion detected!
  No motion detected!
  Motion detected!
  Motion detected!
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



16. DHT11 Sensor

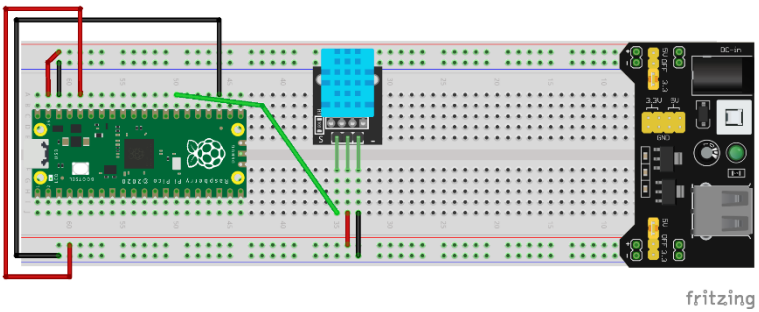
Description

In this tutorial you will learn how to connect and control the DHT11 temperature and humidity sensor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `dht11.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x DHT11 temperature sensor

Wiring diagram



– VCC (red cable) is connected to 3v3 rail (+)

– GND (black cable) is connected to GND rail (-)

– S (green cable) is connected to GPIO21 pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /dht11.py @ 12 : 38

File Edit View Run Tools Help

[dht11.py]

```
1  from machine import Pin
2  from time import sleep
3  import dht
4
5  sensor = dht.DHT11(Pin(21))
6
7  while True:
8      sensor.measure()
9      temp = sensor.temperature()
10     hum = sensor.humidity()
11     print("Room temperature: ",temp, "Celcius")
12     print("Room humidity: ",hum, "%")
13     sleep(5)
```

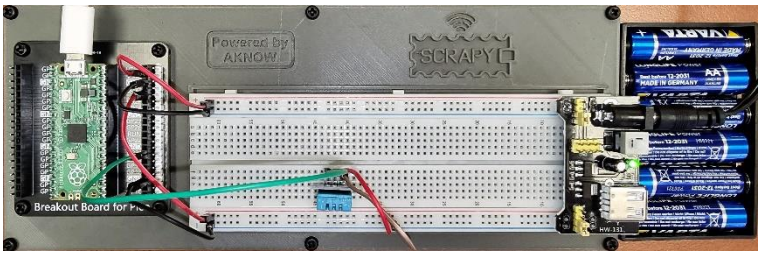
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  Room temperature:  27 Celcius
  Room humidity:  34 %
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



17. SW-420 Vibration Sensor

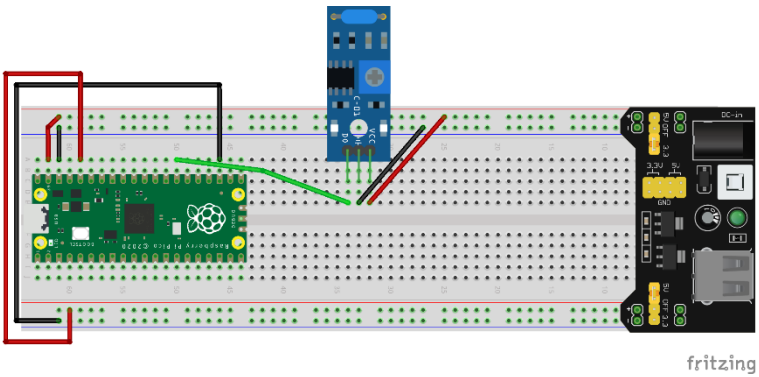
Description

In this tutorial you will learn how to connect and control the SW-420 vibration sensor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `vibration.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x SW-420 vibration sensor

Wiring diagram



– VCC (red cable) is connected to 5V rail (+)

– GND (black cable) is connected to GND rail (-)

– DO (green cable) is connected to GPIO21 pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /vibration.py @ 8 : 37

File Edit View Run Tools Help

[vibration.py]

```
1  from machine import Pin
2  from time import sleep
3
4  vibration = Pin(21, Pin.IN, Pin.PULL_UP)
5
6  while True:
7      if vibration.value() == 0:
8          print("Vibration detected!")
```

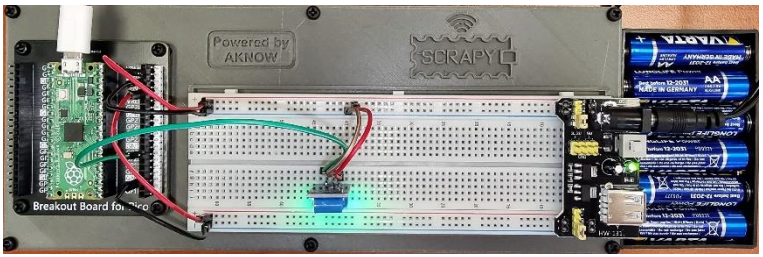
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  Vibration detected!
  Vibration detected!
  Vibration detected!
  Vibration detected!
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



18. Flame Sensor

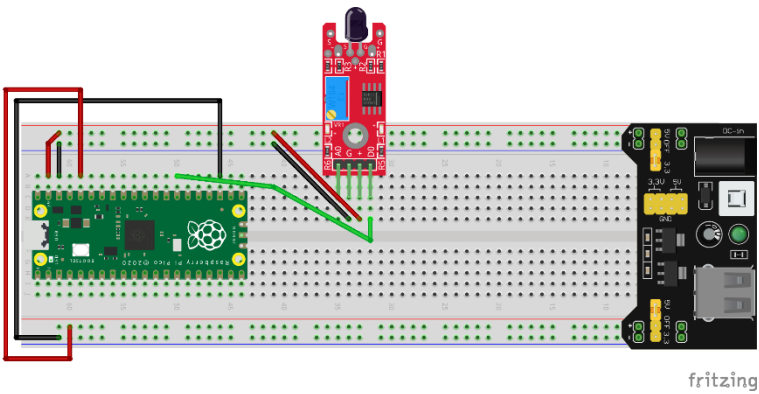
Description

In this tutorial you will learn how to connect and control the KY-026 flame sensor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `flame.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x KY-026 flame sensor

Wiring diagram



– VCC (red cable) is connected to 5V rail (+)

– GND (black cable) is connected to GND rail (-)

– DO (green cable) is connected to GPIO21 pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /flame.py @ 10 : 16

File Edit View Run Tools Help

[flame.py]

```
1  from machine import Pin
2  from time import sleep
3
4  flame = Pin(21, Pin.IN)
5  sleep(2)
6
7  while True:
8     if flame.value() == 0:
9          print("Flame Detected")
10         sleep(3)
```

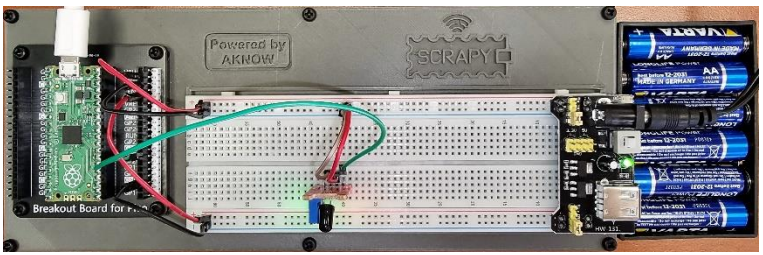
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



19. Sound Detection Sensor

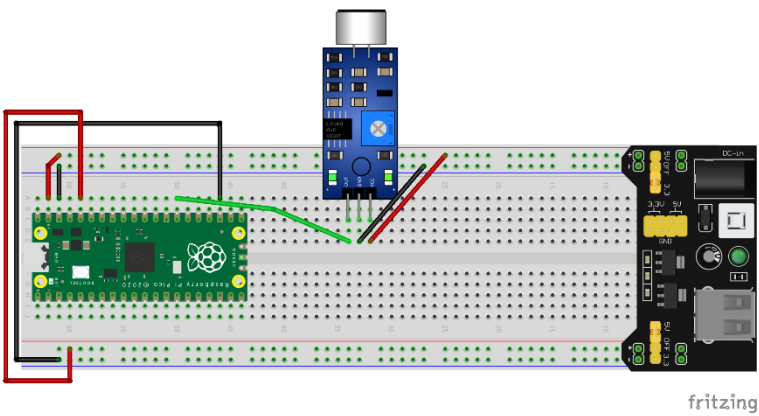
Description

In this tutorial you will learn how to connect and control the KY-037 sound detection sensor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `sound.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x KY-037 sound sensor

Wiring diagram



– VCC (red cable) is connected to 5V rail (+)

– GND (black cable) is connected to GND rail (-)

– DO/OUT (green cable) is connected to GPIO21 pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /sound.py @ 10 : 16

File Edit View Run Tools Help

[sound.py]

```
1  from machine import Pin
2  from time import sleep
3
4  sound = Pin(21, Pin.IN)
5  sleep(2)
6
7  while True:
8     if sound.value() == 0:
9          print("Sound Detected")
10         sleep(3)
```

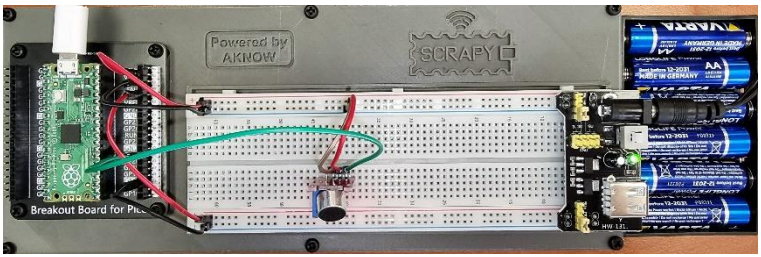
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  Sound Detected
  Sound Detected
  Sound Detected
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



20. Soil Moisture Sensor

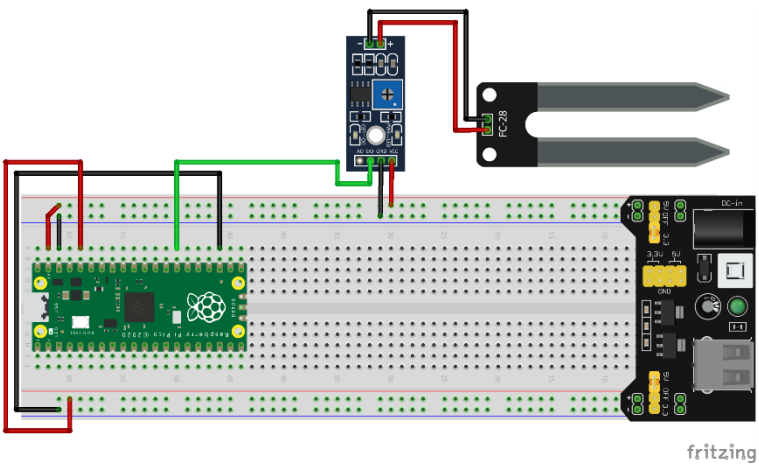
Description

In this tutorial you will learn how to connect and control the soil moisture sensor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `soil.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x KY-037 sound sensor

Wiring diagram



– VCC (red cable) is connected to 5V rail (+)

– GND (black cable) is connected to GND rail (-)

– DO/OUT (green cable) is connected to GPIO21 pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /soil.py @ 10 : 16

File Edit View Run Tools Help

[soil.py]

```
1  from machine import Pin
2  from time import sleep
3
4  soil = Pin(21, Pin.IN)
5  sleep(2)
6
7  while True:
8     if soil.value() == 0:
9         print("Moisture Detected")
10        sleep(3)
```

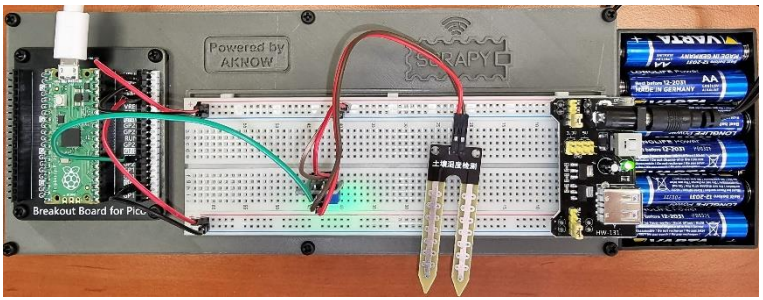
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  Moisture Detected
  Moisture Detected
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



21. Infrared IR Sensor

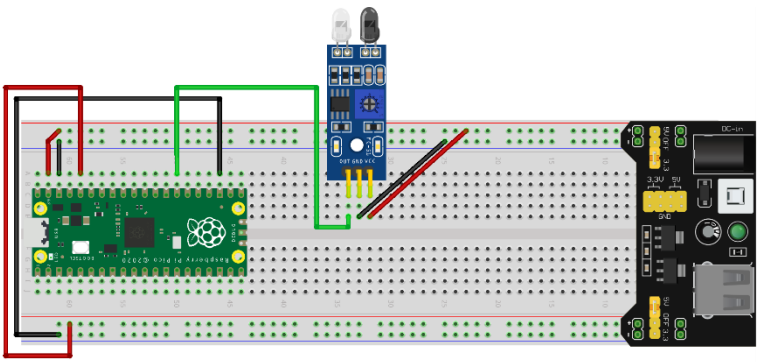
Description

In this tutorial you will learn how to connect and control the infrared IR sensor. Open Thonny Python, then go to File → Save as…, choose Raspberry Pi Pico, and save your file under the name `ir.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required material

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Pico breadboard kit
- 3 x Male-to-male jumper wires
- 1 x Infrared IR sensor

Wiring diagram



– VCC (red cable) is connected to 5V rail (+)

– GND (black cable) is connected to GND rail (-)

– OUT (green cable) is connected to GPIO21 pin

Code

MicroPython code for the tutorial:

Thonny - Raspberry Pi Pico :: /ir.py @ 13 : 17

File Edit View Run Tools Help

[ir.py]

```
1  from machine import Pin
2  from time import sleep
3
4  ir = Pin(21, Pin.IN)
5  sleep(2)
6
7  while True:
8      if ir.value() == 1:
9          print("No obstacles")
10         sleep(3)
11     else:
12         print("Obstacle detected!")
13         sleep(3)
14
```

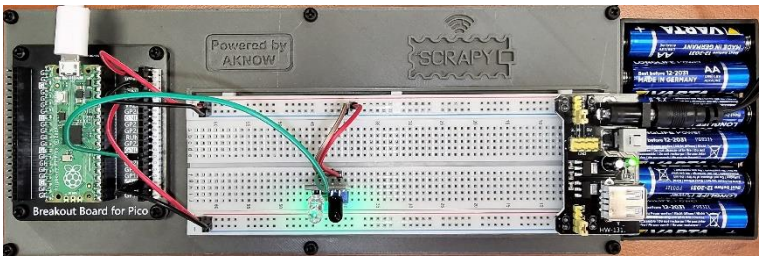
Shell

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  Obstacle detected!
  Obstacle detected!
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



APPENDIX: MicroPython Sum-up Table

Digital Output		
Calling the Pin class	`from machine import Pin`	
Initialisation of the digital output object	`led = Pin(pin_value, Pin.OUT)`	`pin_value` from 0 to 40
Open digital output (3.3V output)	`led.value(1)`	ON
Close digital output (0V output)	`led.value(0)`	OFF

Digital Input		
Calling the Pin class	`from machine import Pin`	
Initialisation of the digital output object	`button = Pin(pin_value, Pin.IN)`	`pin_value` from 0 to 40
	`button = Pin(pin_value), Pin.IN, Pin.PULL_UP)`	Activation of PULL UP resistance
	`button = Pin(pin_value), Pin.IN, Pin.PULL_DOWN)`	Activation of PULL DOWN resistance
Input reading	`value = button.value(1)`	Return value could be 0 if pin is at 0V, or 1 if pin is at 3.3V

Analog Output (Pulse Width Modulation - PWM)		
Calling the PWM class	`from machine import PWM`	
Initialisation of the analog output	`led = PWM(Pin(pin_value), frequency)`	`pin_value` from 0 to 40 `frequency` in HZ, from 0 to 78125
Input reading	`led.duty(duty_cycle)`	`duty_cycle` from 0 to 1023 (0V output to 3.3V output respectively)

Analog Input		
Calling the ADC class	`from machine import ADC`	
Initialisation of the analog input	`pot = ADC(Pin(pin_value))`	`pin_value` can be GPIO26, GPIO27 and GPIO28
Declaration at which voltage the input will give its maximum value (in ESP32 usually 3.3V)	`pot.atten(ADC.ATTN_11DB)`	**ADC.ATTN_0DB:** full range voltage: 1.2 V **ADC.ATTN_2_5DB:** full range voltage: 1.5 V **ADC.ATTN_6DB**: full range voltage: 2.0 V **ADC.ATTN_11DB**: full range voltage: 3.3 V
Declaration of the input value range (default 12bit)	`pot.width(ADC.WIDTH_10BIT)`	**ADC.WIDTH_9BIT**: range 0 to 511 **ADC.WIDTH_10BIT**: range 0 to 1023 **ADC.WIDTH_11BIT**: range 0 to 2047 **ADC.WIDTH_12BIT**: range 0 to 4095
Input reading	`value = pot.readl()`	`value` is an integer from 0 to the maximum of the range specified by the ADC.WIDTH_#BIT statement (see previous)

The time library		
Calling the sleep class	`from time import sleep`	
Use of sleep function	`sleep(sec)`	`sec` is the number of seconds for which the program will be delayed
Calling the time class	`import time`	
Use of time function	`current_time = time()`	The `current_time` variable will take a numeric value, equal to the number of seconds since the last reset on the board.

If statement structure	

`if <expr1>:`
`  <statement1>`
`elif <expr2>:`
	<expr#>: the control condition that must return True or False

<statement2>
elif <expr3>:
<statement3>
(...)
else:
<statementn>
	<statement#>: set of commands to be executed when the adjacent condition is satisfied
	
	<expr#> (e.g. the set <statement2> is executed when <expr2> is satisfied)
	
	<statementn>: set of instructions executed when none of the <expr#> conditions are satisfied

While loop structure	
while <expr>: <statement(s)>	<expr>: the control condition that must return True or False <statement#>: set of commands to be executed as long as <expr> condition is satisfied

For loop structure	
for <var> in <iterable>: <statement(s)>	**<iterable>**: a collection of objects, for example a list containing numbers, alphanumerics, etc. **<var>**: a variable to which the value of the next item in the collection <iterable> is assigned. **<stametement(s)>**: a set of instructions executed at each iteration
for <var> in range(<start>, <end>, <step>): <statement(s)>	**range(<start>, <end>, <step>)**: function that returns a sequence of numbers from <start> to <end>-1, with a <step> difference between two consecutive numbers (<start> and <step> parameters are optional). **<var>**: variable to which the value of the next element of the sequence produced by range is assigned. **<statement(s)>**: a set of instructions executed in each iteration

Miscellaneous		
DHT11	`import dht`	Importing the DHT library
	`sensor = dht.DHT11(Pin(pin_number))`	Initialisation of the sensor variable with its associated pin_number.
	`sensor.measure()`	Updating sensor values
	`temp = sensor.temperature()`	Saving current temperature value
	`hum = sensor.humidity()`	Saving current humidity value
OLED DISPLAY	`from machine import I2C`	Importing the I2C library
	`import ssd1306`	Importing the ssd1306 library
	`i2c = I2C(-1, scl=Pin(1), sda=Pin(0))`	Initialization of the i2c variable on the SCL & SDA pins of Pico

	`oled_width = 128`
	`oled_height = 64`
	`oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)`
		Initializing the screen

	`oled.text('Hello, World 1!', 0, 0)`
	`oled.text('Hello, World 2!', 0, 10)`
	`oled.text('Hello, World 3!', 0, 20)`
		Storing messages in the screen buffer

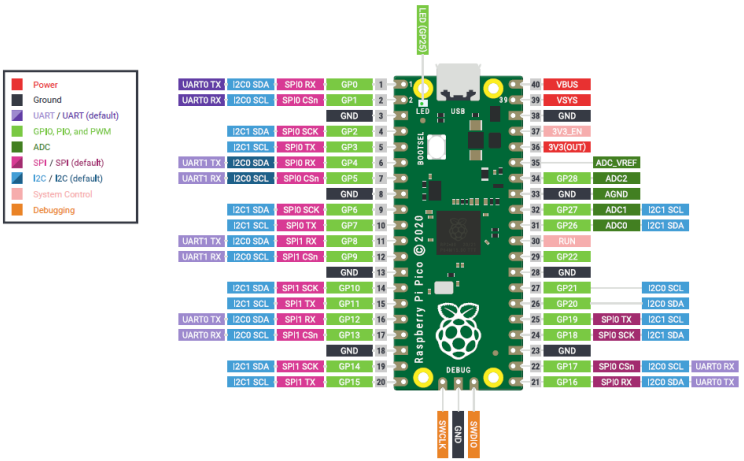
	`oled.show()`	Showing messages (necessary to display messages stored in the screen buffer)
	`display.pixel(3, 4, 1)`	Set the pixel located at position (x,y) on the screen, with x=3 & y=4, to state 1 (i.e. display)

Co-funded by
the European Union

SCRAPY

2021-1-FR01-KA220-SCH-000031617

Raspberry Pi Pico Pinout



Raspberry Pi

w: scrapykit.eu
e: info@scrapykit.eu