

SmartHome4SENIORS Kit Manual



SmartHome
4SENIORS



Co-funded by
the European Union

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

WARNING

When using this product, please note the following:

1. This product contains many small parts. Swallowing or misusing any of those parts can lead to serious medical complications, including death. Seek immediate medical attention if an accident happens.
2. The use of this product and its parts near AC electrical outlets or other circuits is strictly prohibited given the potential risk of electric shock.
3. The use of this product near any liquid or fire is strictly prohibited.
4. Keep conductive materials away from this product.
5. Do not allow small children to use this product without adult supervision. This product must be stored out of the reach of children and pets.
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 the 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 SmartHome4SENIORS Kit is built based on the use of the Raspberry Pi Pico (RPI Pico) microcontroller. The Kit is created under the homonymous Erasmus+ Co-funded project with project number 2021-1-DE02-KA220-ADU-000033587.

Smart-home automation is an existing trend helping thousands of people simplify processes in their households. The SmartHome4SENIORS Kit was conceptualized and developed in light of seniors' lower level of readiness to embrace and understand smart home technological solutions.

The Kit aims to promote safe and healthy living for seniors through hands-on educational learning. The Kit offers a great opportunity for users to dive into the world of DIY smart home automation solutions.

The Kit includes all the necessary hardware (microcontroller, electronics, sensors, peripherals, etc.) incorporating physical computing and programming concepts.

The scope of this Manual is to:

- Inform you about 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 use of the components and their connection with the RPI 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 SmartHome4SENIORS Kit

Table of Contents

Introduction	1
Table of Contents	2
Included in the SmartHome4SENIORS Kit.....	4
Components Explanation.....	6
1. What is a breadboard?.....	6
2. What is a resistor?	7
3. What is a capacitor?	10
4. What is a diode?	10
5. What is a jumper cable?	11
Project Preparation	12
SmartHome Kit Assembly	20
SmartHome Kit Electronics Mounting.....	27
Basic Tutorials	32
0. “Hello SmartHome people!”	32
1. Control an LED	34
2. Push Button.....	36
3. Buzzer	38
4. Potentiometer.....	40
Advanced Tutorials	42
5. LED Traffic Lights Module.....	43
6. LDR Photoresistor.....	45
7. DC Motor (small fan).....	47
8. SG-90 Servo Motor	49
9. OLED I2C SSD1306 Display	51

10. RFID Reader RC522.....	55
Tutorials with Sensors	58
11. Raindrop sensor.....	58
12. HC-SR04 Ultrasonic Sensor	60
13. PIR Motion Sensor.....	63
14. DHT11 Sensor	65
15. Flame Sensor.....	67
16. MQ-135 Gas Detection Sensor.....	69
APPENDIX: MicroPython Sum-up Table.....	71

Included in the SmartHome4SENIORS Kit

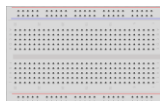
1 x Raspberry Pi Pico
Microcontroller



1 x MB-102 Power
Supply Module



1 x White Breadboard
830 pcs



1 x Push Button &
1 x Button Cap



1 x Buzzer



5 x Resistors 220 Ohm &
2 x 1k Ohm



1 x LDR Photoresistor



1 x 100uF Capacitor



4 x LED 3mm
(blue, green, red, yellow)



1 x RGB LED



1 x Traffic Light LED
Module



1 x Fan (DC Motor)



1 x TIP-120 Control
Module



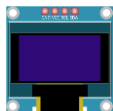
1 x Diode 1N4007



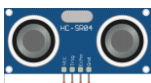
1 x SG90 Servo Motor



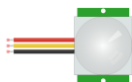
1 x OLED SSD1306 I2C Display



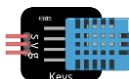
1 x HC-SR04 Ultrasonic Sensor



1 x PIR Motion Detector Sensor HC-SR501



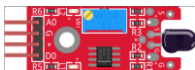
1 x DHT11 Digital Temperature & Humidity Sensor



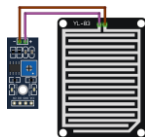
1 x MQ-135 Air Quality Sensor



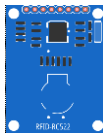
1 x Flame Detection Sensor



1 x Raindrop Sensor



1 x RFID Reader RC522



1 x RFID Tag 3.56MHz



1 x Rotary Potentiometer Linear B1k Ohm



1 x USB to micro-USB Cable



6 x AA 1.5V Batteries & 1 x Battery Holder



9 x Jumper Cables Packs



19 x Metal Bolts 2mm & 19 x Metal Nuts 2mm

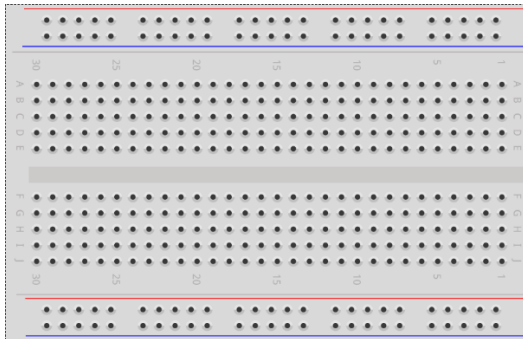


4 x Metal Screws 2mm



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>

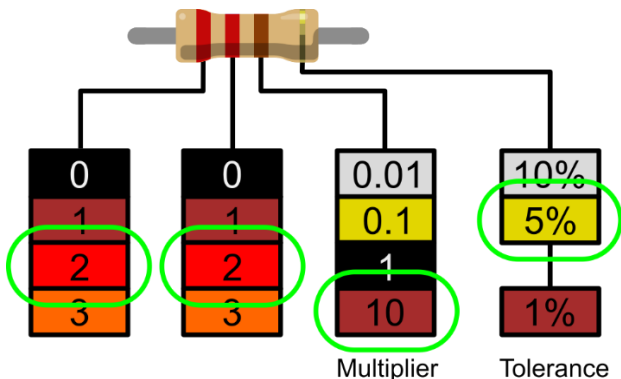
Common Resistor Colour Codes and their uses:

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 most hobbyists, a tolerance of 5% (Gold) is perfect and common.

220 Ohm Resistor (4-Band)

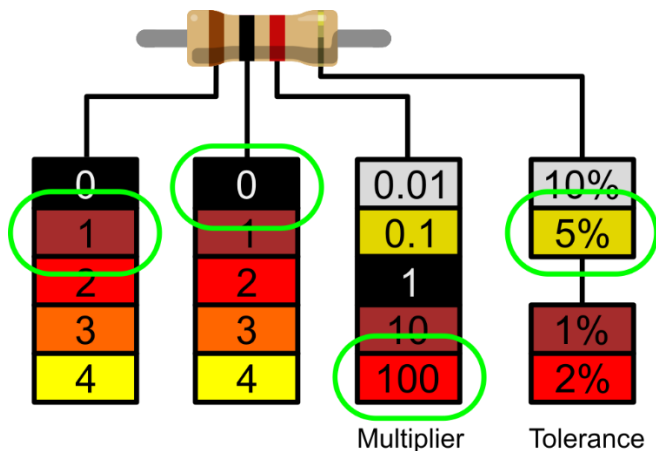


(Image credit: Future) available at <https://www.tomshardware.com/how-to/resistor-color-codes>

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 smooth power delivery.

4. What is a diode?

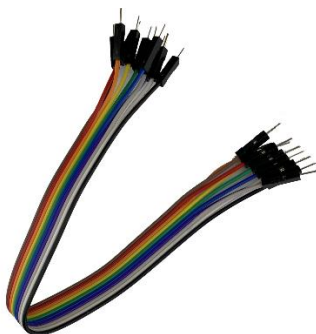


A diode is a two-terminal electronic component that conducts current primarily in one direction (asymmetric conductance); it has low (ideally zero) resistance in one direction, and high (ideally infinite) resistance in the other.

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 to make it easy to change a circuit as needed. The colour variation of the wires can be used as advantage 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>)

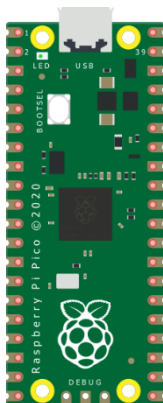
Project Preparation

1. Readme Before Using

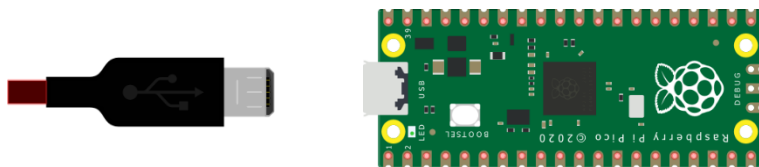
NOTE: Since the experiments involved are all circuit experiments, a wrong connection or short circuit may damage your RPi Pico development board. Please, always check the circuit again before connecting the power supply.

2. Raspberry Pi Pico Microcontroller

This is the Raspberry Pi Pico:

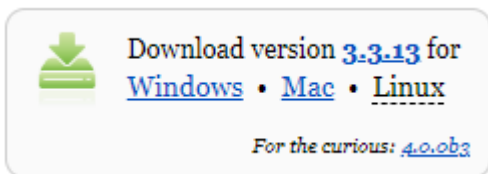


Plug the micro-USB cable into the port on the left-hand side of the board.



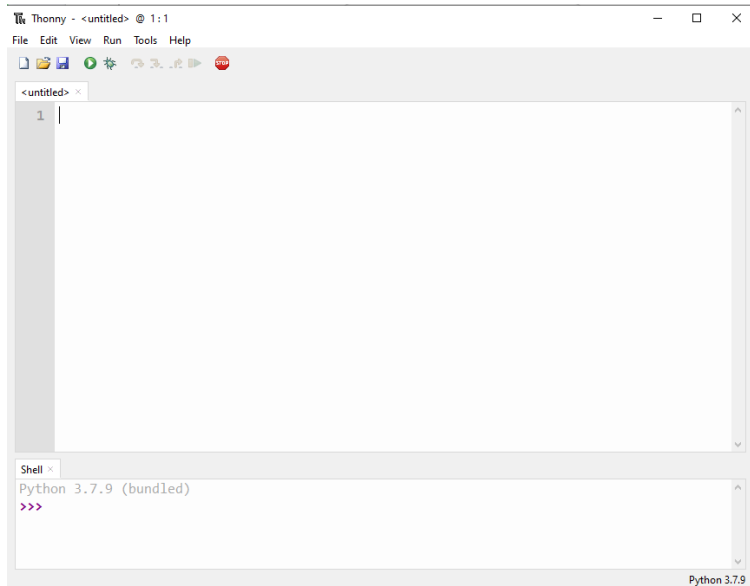
3. Install Thonny IDE

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 RPi Pico microcontroller and the appropriate firmware.

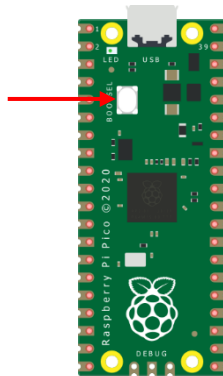
After installation is completed, open Thonny from your computer.



4. Firmware installation

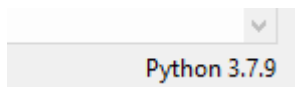
The RPi Pico can be programmed using a Python variant, called MicroPython. To use MicroPython on the RPi Pico, first you need to install its firmware.

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

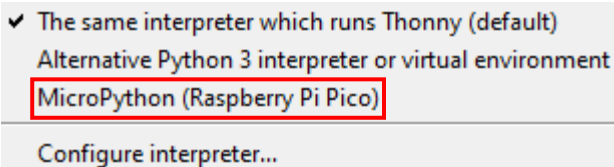


Step 2: Press the BOOTSEL button and hold it while you connect the other end of the micro-USB cable to your computer.

Step 3: In the bottom right-hand corner of Thonny you will see the version of Python you currently use.

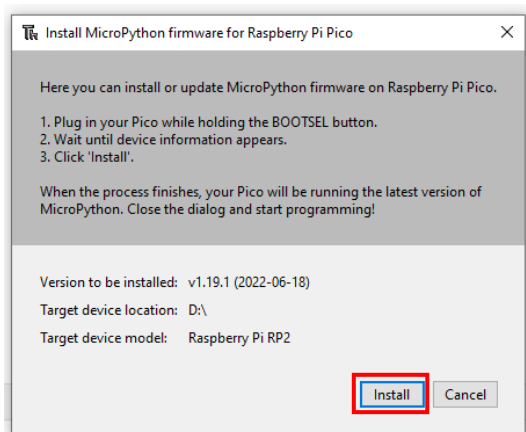


Click on the Python version and choose the MicroPython (Raspberry Pi Pico)

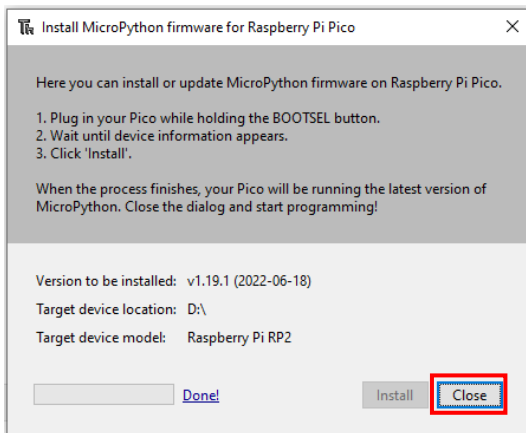


If you don't see this option, make sure the cable is connected properly on the Pico and/or your computer.

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**.



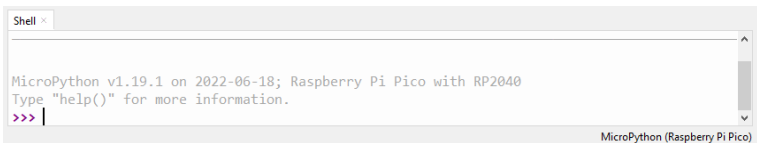
You don't need to repeat the process every time you connect the Raspberry Pi Pico to your computer, so next time, just plug it in and you are good to go.

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:



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

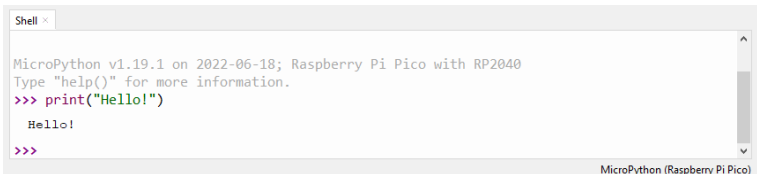
MicroPython (Raspberry Pi Pico)

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:



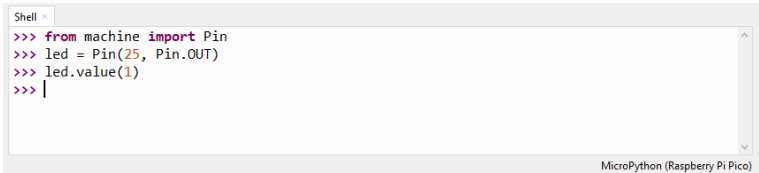
```
Shell <-
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> print("Hello!")
Hello!
>>>
```

MicroPython (Raspberry Pi Pico)

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)
```



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

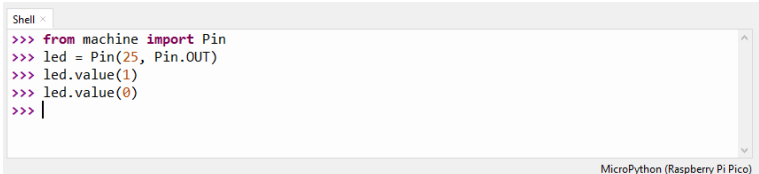
MicroPython (Raspberry Pi Pico)

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)
```



```
Shell
>>> from machine import Pin
>>> led = Pin(25, Pin.OUT)
>>> led.value(1)
>>> led.value(0)
>>> |
```

MicroPython (Raspberry Pi Pico)

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.

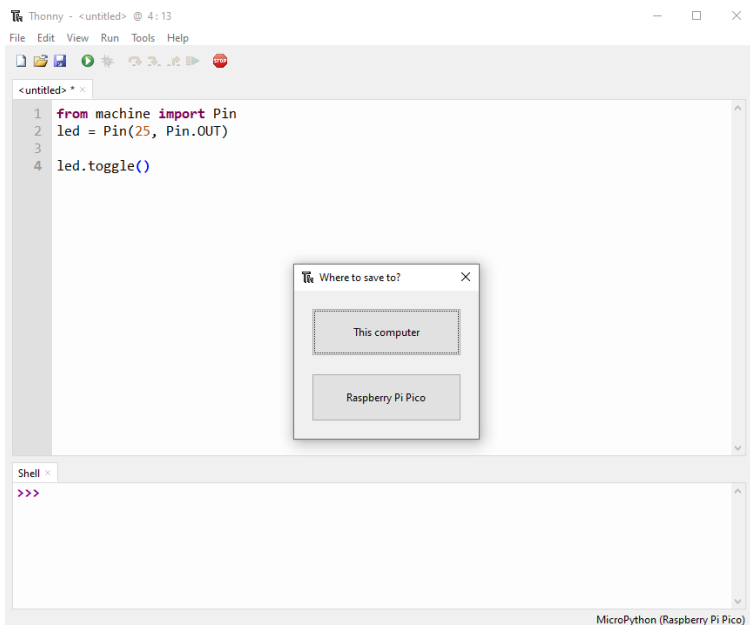
Thonny Shell is useful to try out quick commands and make sure everything is working correctly. However, longer programs

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

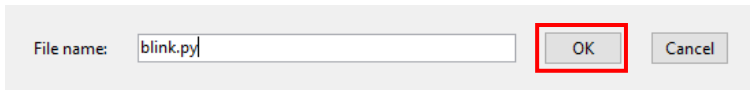
Open Thonny Python and on the main editor pane write the following code:

```
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.

SmartHome Kit Assembly

The assembly process requires the following items:

- 6 x Plywood pieces
- 10 x Metal bolts
- 10 x Metal nuts



The procedure is simple and can be completed in **7 steps**. All plywood pieces are marked on the right-hand side corner with the following notation:

BP: Base piece

B: Back-side piece

F: Front-side piece

L: Left-side piece

R: Right-side piece

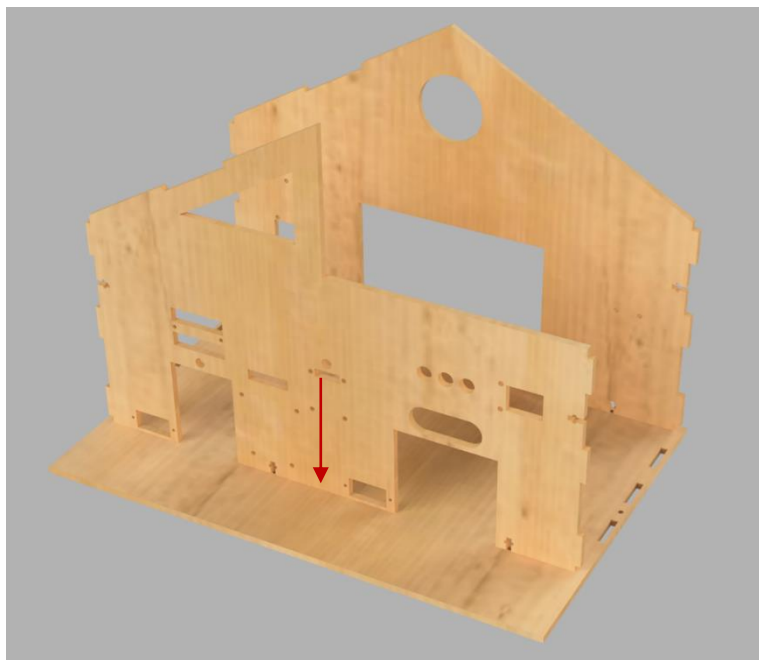
Step 1: Place the plywood base piece on a horizontal surface.



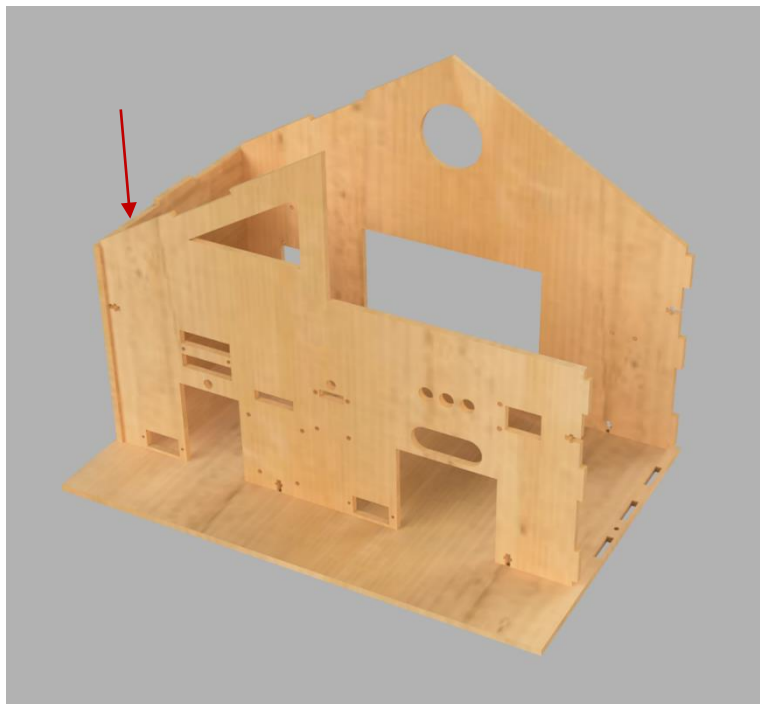
Step 2: Insert the back-side piece as shown in the image.



Step 3: Insert the front-side piece as shown in the image.



Step 4: Insert the plywood left-side piece as shown in the image.

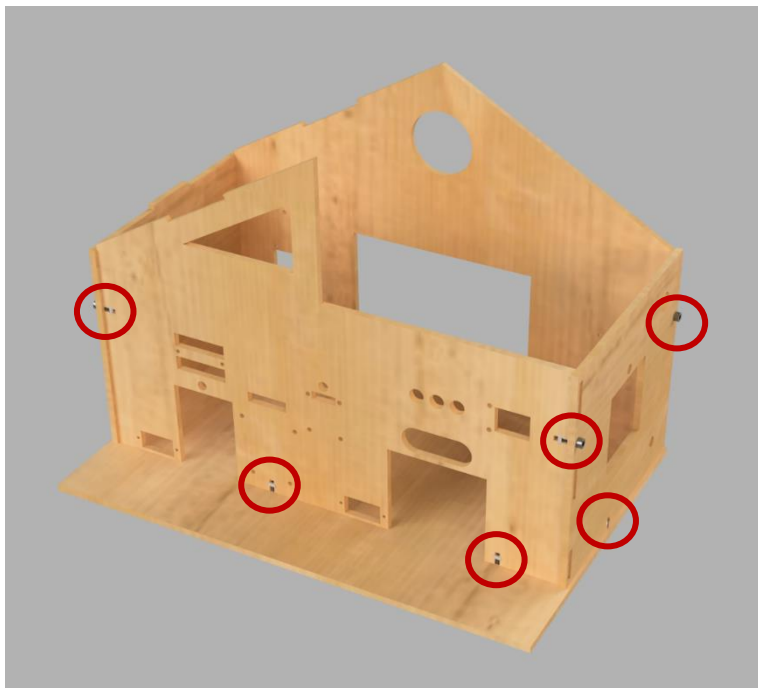


Step 5: Insert the plywood right-side piece as shown in the image.

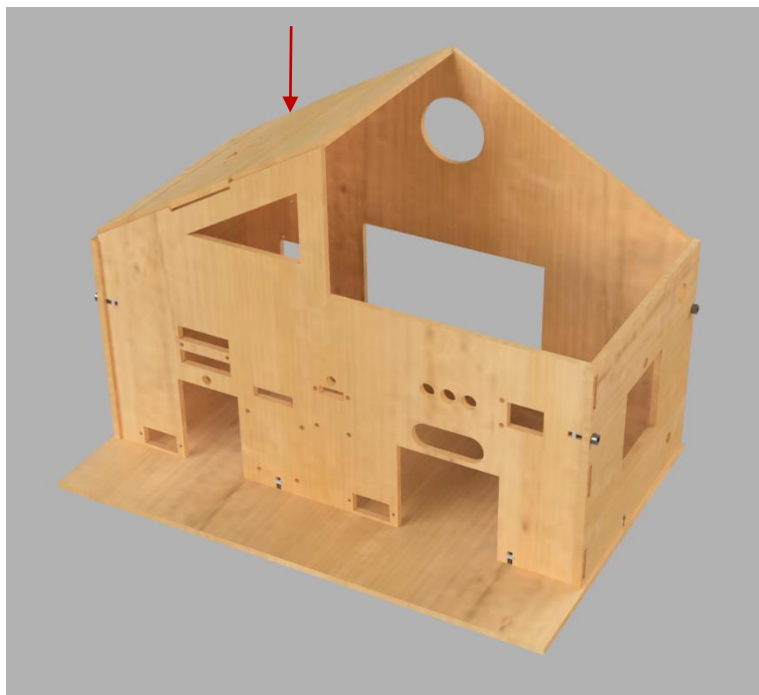


Step 6: Tighten the plywood pieces together using 10 bolts and 10 nuts.

- 6 bolts and nuts are placed under the base piece and are responsible for keeping together the walls.
- 4 bolts and nuts are placed on the left-side and right-side walls and are responsible for keeping the construction stable.

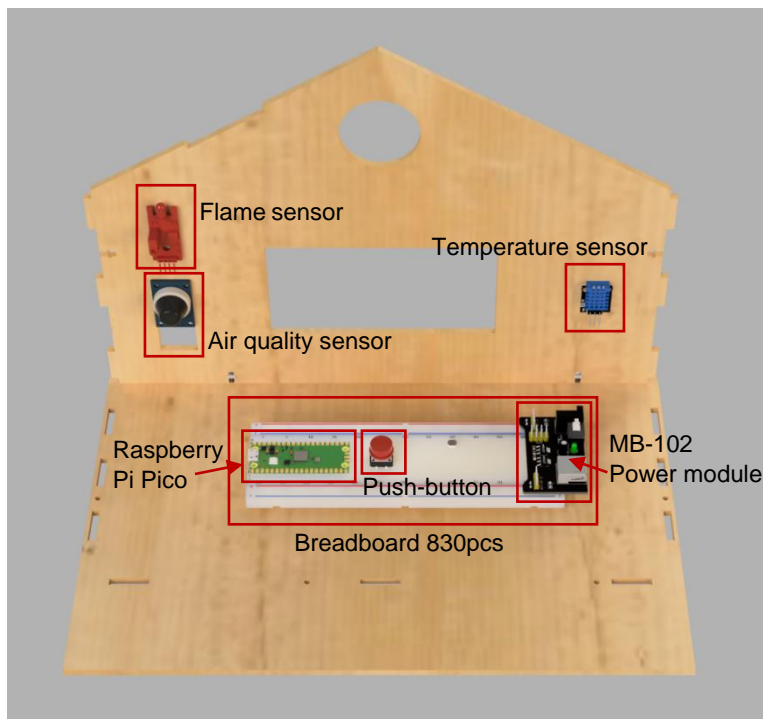


Step 7: Place the plywood roof piece on top of the house model as shown in the image:



SmartHome Kit Electronics Mounting

There are several electronics that needs to be mounted on the SmartHome house model. The following images present in detail what electronics to mount and how to mount them on the different slots. You will also need a Phillips-head screwdriver and a plier.

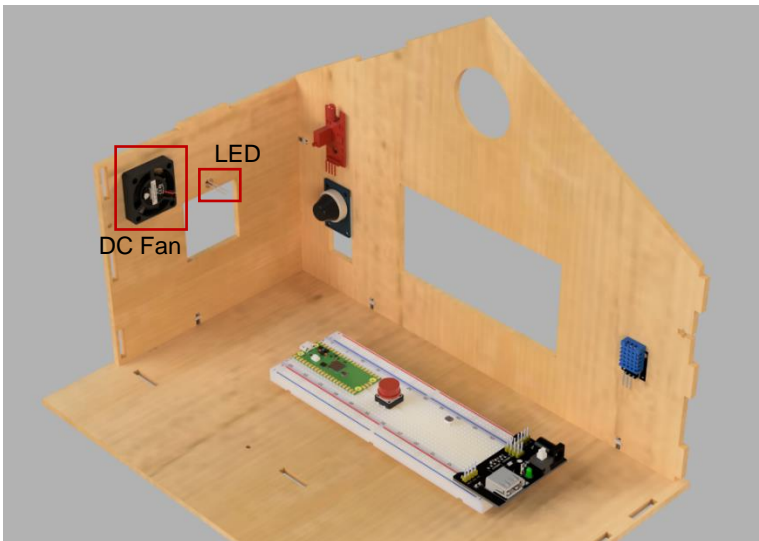


Place the Raspberry Pi Pico on the breadboard, along with the power module (MB-102) and the push button. The breadboard has an adhesive surface underneath, so make sure to peel off the protection layer and then place it on the base piece of the house.

There are some electronics that need to be mounted on the back-side wall. These are the flame sensor, the air-quality sensor and the DHT11 temperature and humidity sensor. You need to use the following items:

- **Flame sensor:** 1 x bolt and 1 x nut. Mount it through the middle section of the sensor and make sure the sensor (black part) looks up.
- **Air-quality sensor:** 2 x bolt and 2 x nut. Mount it through the top left and top right mounting holes.
- **DHT11:** 2 x bolt and 2 x nut. You need to bend the sensor (blue part) forward to access the mounting holes. Then mount the sensor through the top left and top right mounting holes.

On the left-side piece you will need to mount a **DC fan** and an **LED** light. For the **DC fan** you will use the bolts and nuts that come in its package (4 x bolts and 4 x nuts). Insert the **LED** to the mounting hole and friction will keep it in place.



On the right-side piece you will need to mount a 5V buzzer, a potentiometer, and an LED.

Insert the **buzzer** to the upper-right corner slot (make sure its pins are on the inside of the house model) and friction will keep it in place.

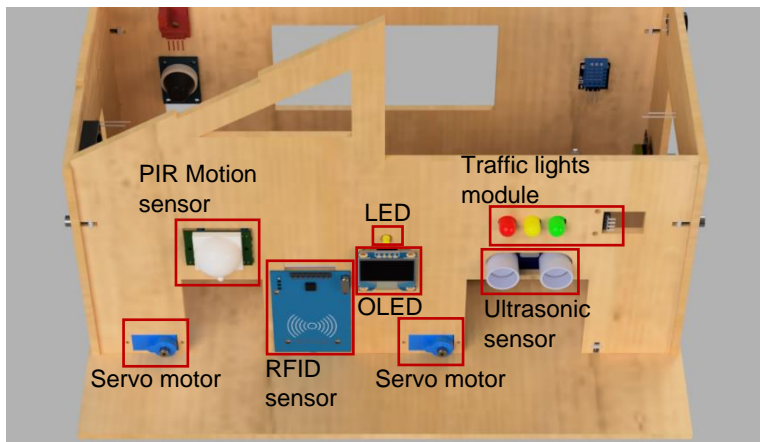
The **potentiometer** has already a nut ring installed that you will need to unscrew. Place the potentiometer from the inside of the house towards the outside so the lever looks like in the below image. Then screw back the nut ring until it tightens enough to keep it in place when you turn the lever.

Finally, an **LED** needs to be installed. Insert the LED in the mounting hole and friction will keep it in place.

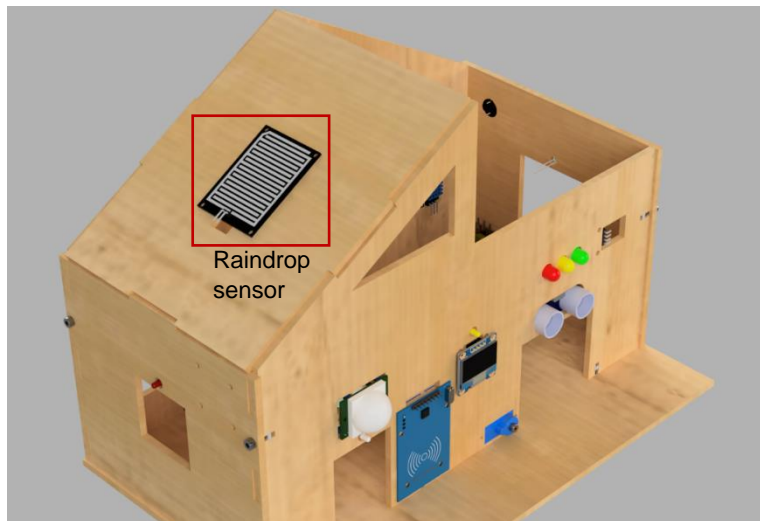


There are several sensors and electronics that need to be mounted in the front-side of the house model.

- **Servo motors:** 4 x screws. Place them from the inside of the house towards the outside and mount them using the screws included in the package.
- **PIR Motion sensor:** 2 x bolt and 2 x nuts. Mount it through the top left and bottom right mounting holes.
- **RFID sensor:** 2 x bolt and 2 x nuts. Mount it through the top left and bottom right mounting holes.
- **OLED display:** 2 x bolt and 2 x nuts. Mount it through the top left and bottom right mounting holes.
- **Traffic lights module:** 2 x bolt and 2 x nuts. Insert the module from the inside of the house towards the outside. Then mount it through the two mounting holes on the right side.
- **Ultrasonic sensor:** Insert it from the inside of the house towards the outside. Friction will keep it in place.
- **LED:** Insert the LED in the mounting hole and friction will keep it in place.



Finally, the **raindrop sensor** should be installed on the roof. Use 2 x bolts and 2 x nuts and mount it through the mounting holes on the top-left and bottom-right



After all electronics and sensors are mounted in place, you will need to wire them using the jumper cables that are included in the package. However, this procedure will be explained in the tutorials section which will explain the different GPIO pins of the Raspberry Pi Pico and the pins of each electronic component or sensor.

In the package you will also find 6 AA batteries and a battery holder. Place the batteries on the battery holder and keep it aside for now. In a later tutorial you will learn where you should place it and how it should be connected with the MB-102 power module.

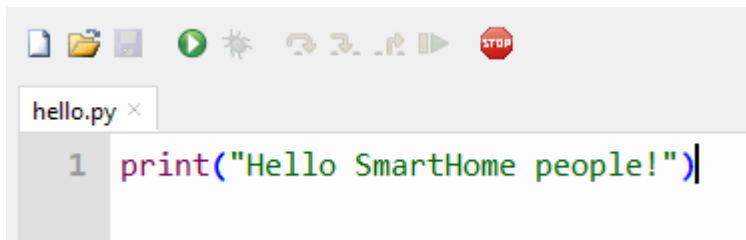
Basic Tutorials

0. “Hello SmartHome people!”

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

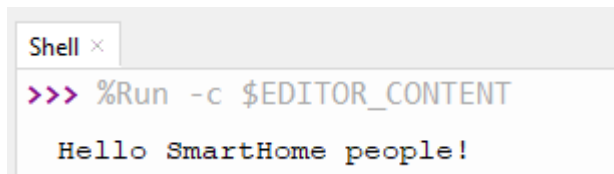
The print() function

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.



```
hello.py x
1 print("Hello SmartHome people!")
```

2. Check the shell window.



```
Shell x
>>> %Run -c $EDITOR_CONTENT
Hello SmartHome people!
```

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.

```

hello.py x
1 print(1,2,3,4,5) #This is a comment!
2 print("I am ",2,"awesome") #1 line
3 print("Python is") #1 line
4 print("amazing") #1 line
5 print("I cant wait.....\n to learn more") # 2 lines of output!
  
```

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.

```

Shell x
>>> %Run hello.py

1 2 3 4 5
I am 2 awesome
Python is
amazing
I cant wait.....
  to learn more
  
```

Exercise

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

1. Control an LED

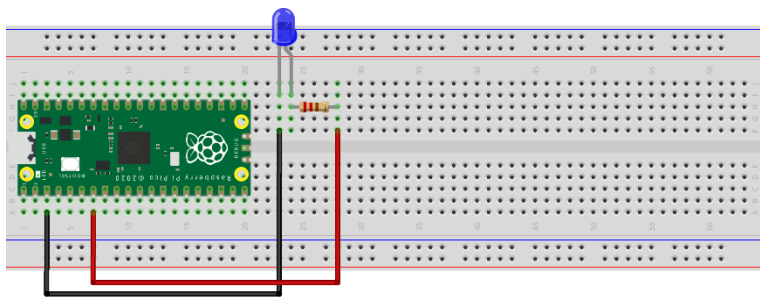
Description

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 materials

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

Wiring diagram



fritzing

- 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

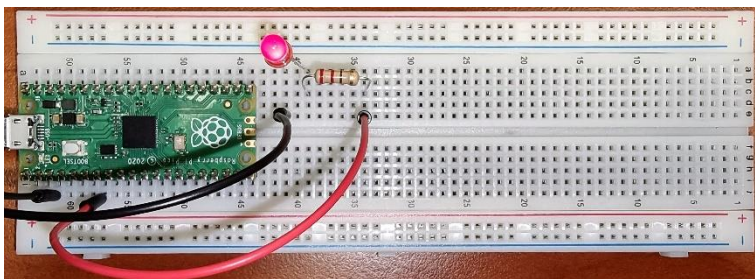
[ led.py ] x
1 from machine import Pin #import Pin from library machine
2 from time import sleep #import sleep from library time
3
4 led = Pin(5, Pin.OUT)
5
6 while True:
7     led.toggle()
8     sleep(1)
9 |

Shell x
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:



2. Push Button

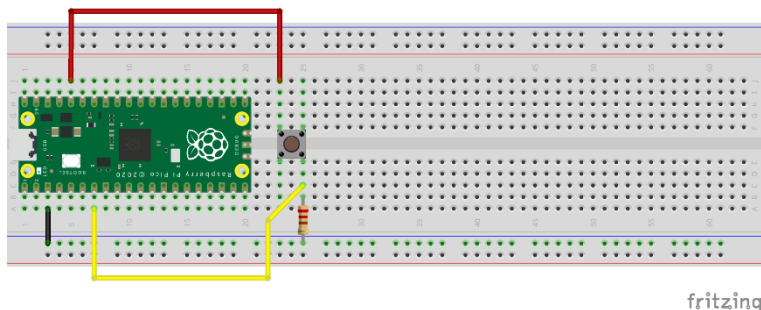
Description

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 materials

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

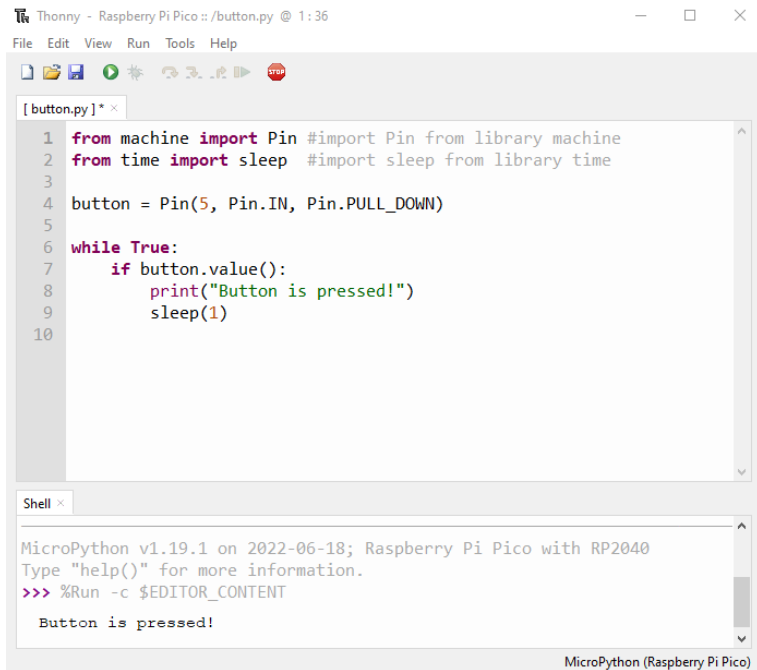
Wiring diagram



- 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:

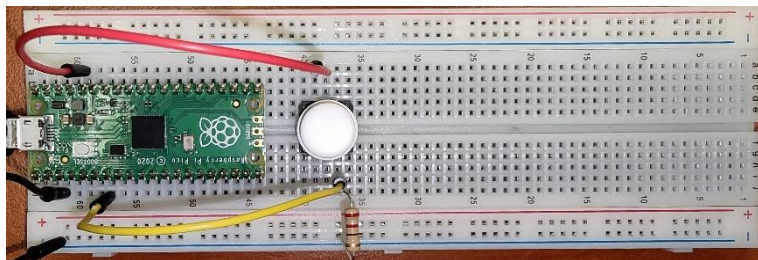


```

Thonny - Raspberry Pi Pico :: /button.py @ 1:36
File Edit View Run Tools Help
[button.py]* x
1 from machine import Pin #import Pin 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 x
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
Button is pressed!
MicroPython (Raspberry Pi Pico)
  
```

Sample image

Image of how the tutorial looks using the provided hardware:



3. Buzzer

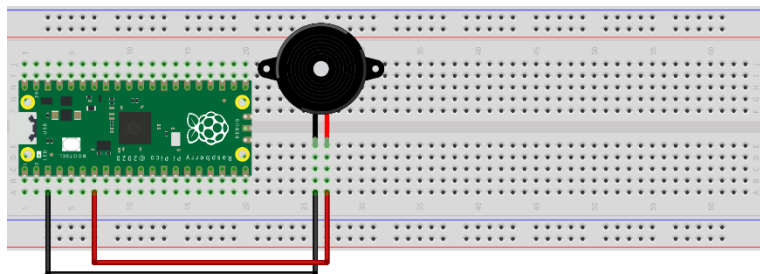
Description

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 materials

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

Wiring diagram

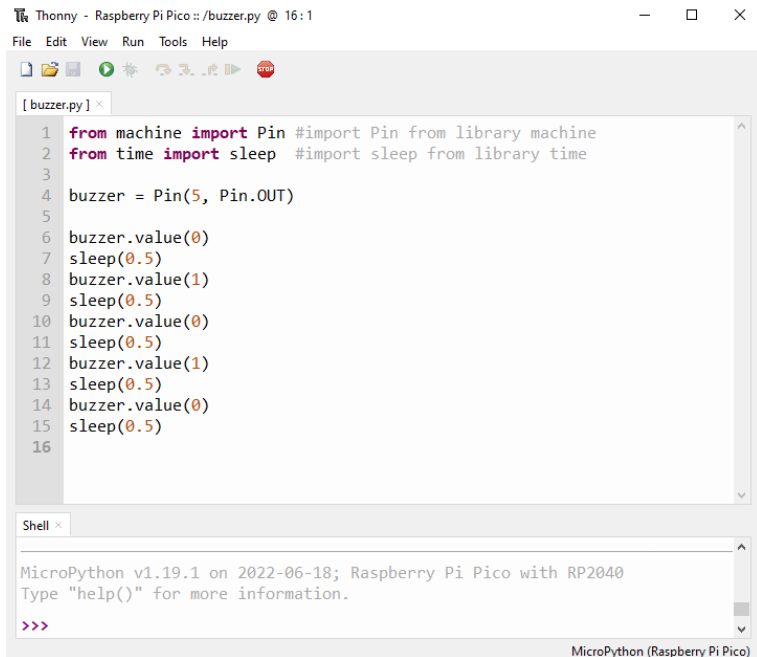


fritzing

- 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

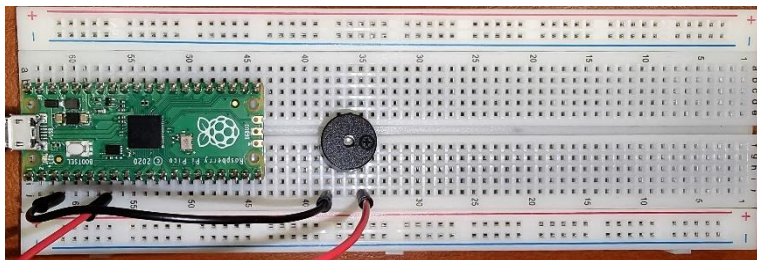
[ buzzer.py ] x
1 from machine import Pin #import Pin from library machine
2 from time import sleep #import sleep from library time
3
4 buzzer = Pin(5, Pin.OUT)
5
6 buzzer.value(0)
7 sleep(0.5)
8 buzzer.value(1)
9 sleep(0.5)
10 buzzer.value(0)
11 sleep(0.5)
12 buzzer.value(1)
13 sleep(0.5)
14 buzzer.value(0)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

Shell x
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:



4. Potentiometer

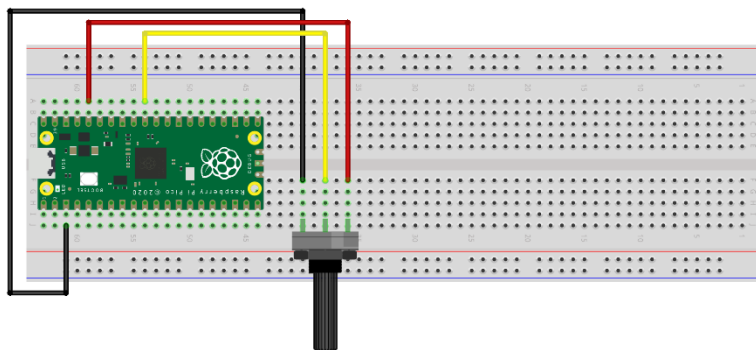
Description

In this tutorial you will learn how to connect and control a 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 materials

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

Wiring diagram

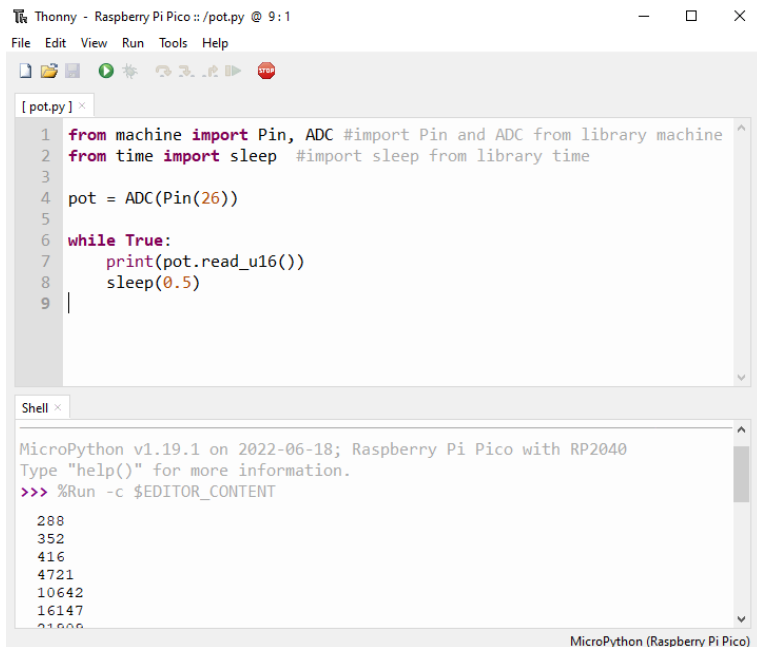


fritzing

- black cable should be connected to GND (Pin 3) pin
- yellow cable should be connected to GPIO26 ADC 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:



```

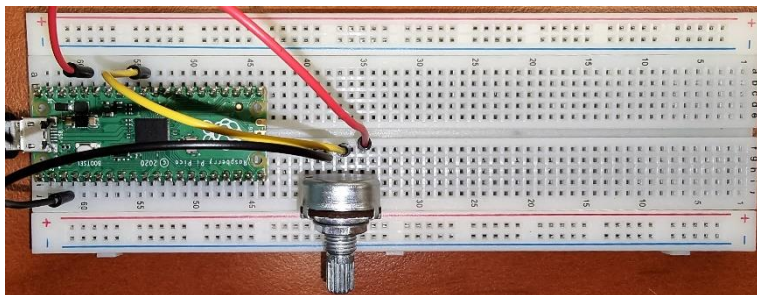
Thonny - Raspberry Pi Pico :: /pot.py @ 9:1
File Edit View Run Tools Help

[ pot.py ] x
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 pot = ADC(Pin(26))
5
6 while True:
7     print(pot.read_u16())
8     sleep(0.5)
9

Shell x
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
288
352
416
4721
10642
16147
21000
MicroPython (Raspberry Pi Pico)
  
```

Sample image

Image of how the tutorial looks using the provided hardware:



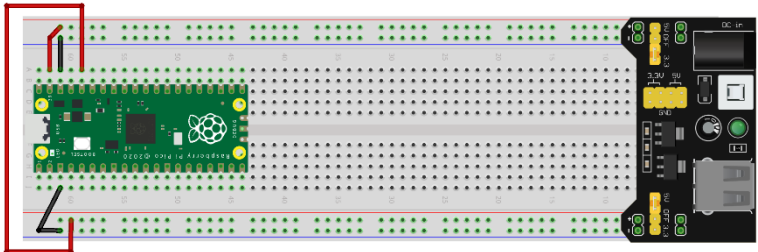
Advanced Tutorials

Before proceeding to this section and the next, we need to make a few modifications to our development board. 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

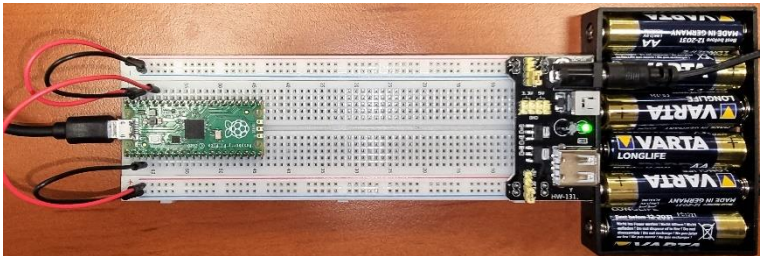
Please follow the schematic to connect the new components:



fritzing

- 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)

Sample image



5. LED Traffic Lights Module

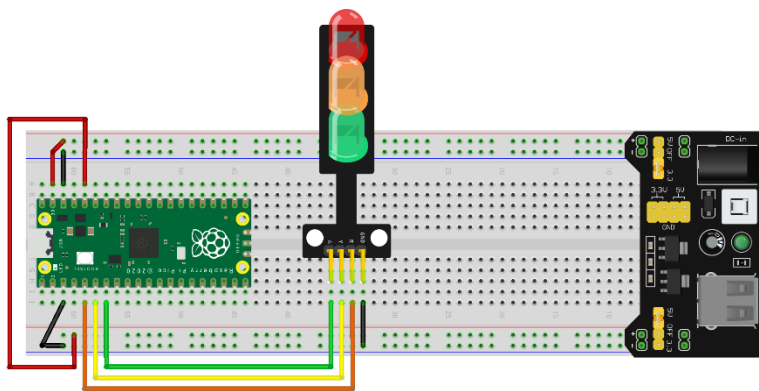
Description

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

Required materials

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

Wiring diagram

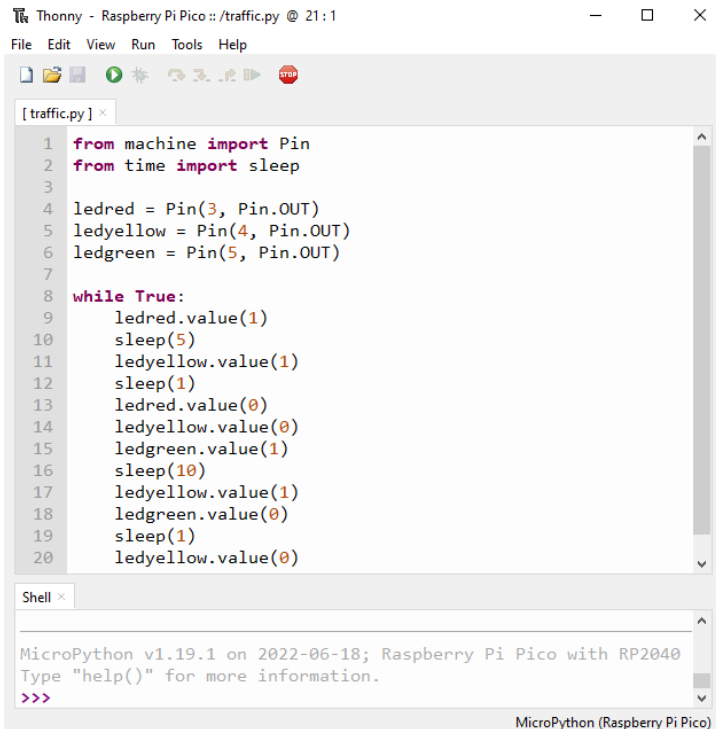


fritzing

- orange cable (R) is connected to GPIO3 pin
- yellow cable (Y) is connected to GPIO4 pin
- green cable (G) is connected to GPIO5 pin
- black cable (GND) is connected to GND rail ((-) black)

Code

MicroPython code for the tutorial:



```

Thonny - Raspberry Pi Pico :: /traffic.py @ 21: 1
File Edit View Run Tools Help

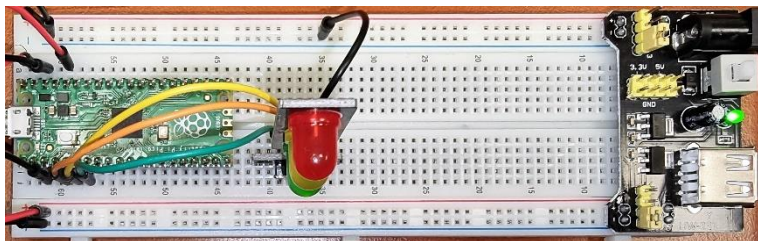
[ traffic.py ] x
1 from machine import Pin
2 from time import sleep
3
4 ledred = Pin(3, Pin.OUT)
5 ledyellow = Pin(4, Pin.OUT)
6 ledgreen = Pin(5, Pin.OUT)
7
8 while True:
9     ledred.value(1)
10    sleep(5)
11    ledyellow.value(1)
12    sleep(1)
13    ledred.value(0)
14    ledyellow.value(0)
15    ledgreen.value(1)
16    sleep(10)
17    ledyellow.value(1)
18    ledgreen.value(0)
19    sleep(1)
20    ledyellow.value(0)

Shell x
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:



6. LDR Photoresistor

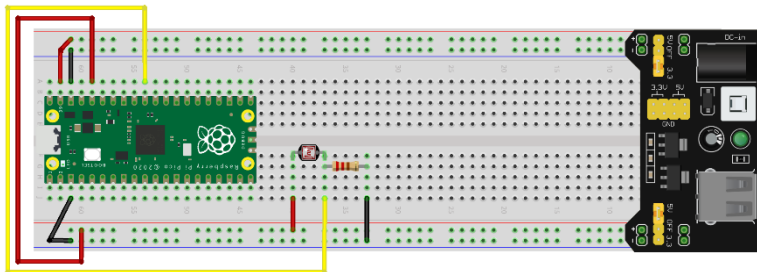
Description

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 materials

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

Wiring diagram

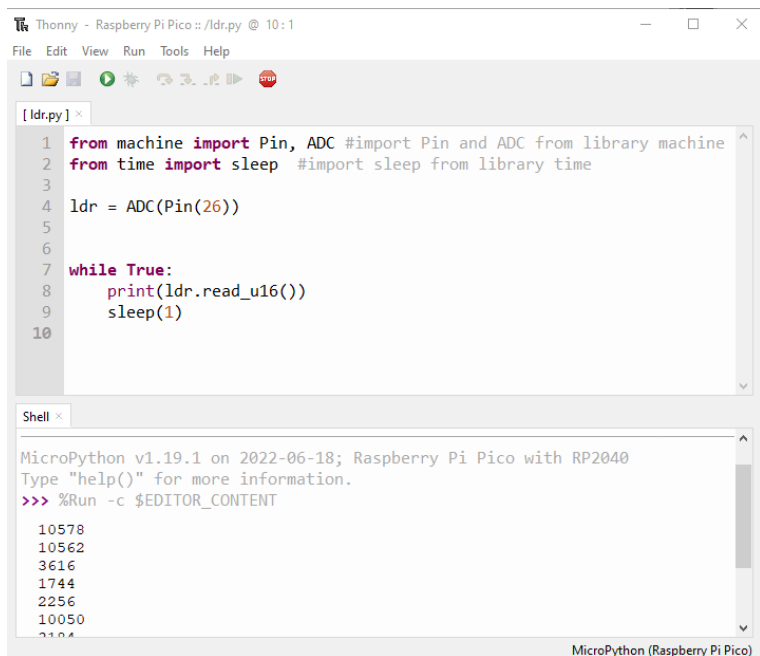


fritzing

- 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:



```

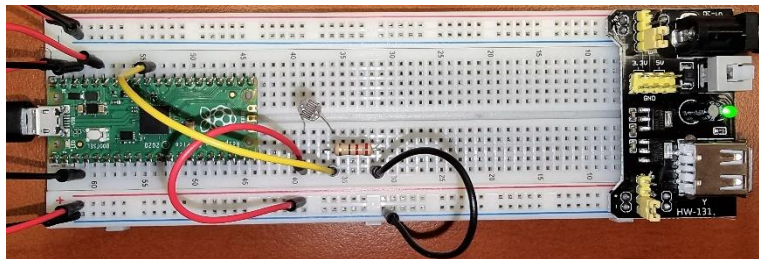
Thonny - Raspberry Pi Pico :: /ldr.py @ 10:1
File Edit View Run Tools Help

[ ldr.py ] x
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 x
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
10578
10562
3616
1744
2256
10050
2104
MicroPython (Raspberry Pi Pico)
  
```

Sample image

Image of how the tutorial looks using the provided hardware:



7. DC Motor (small fan)

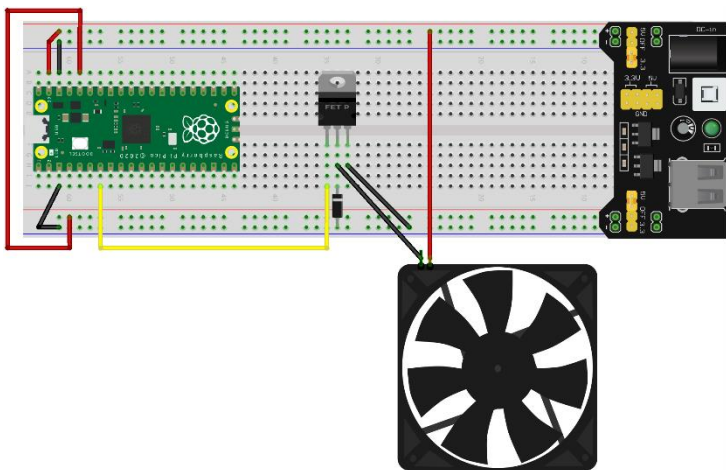
Description

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

Required materials

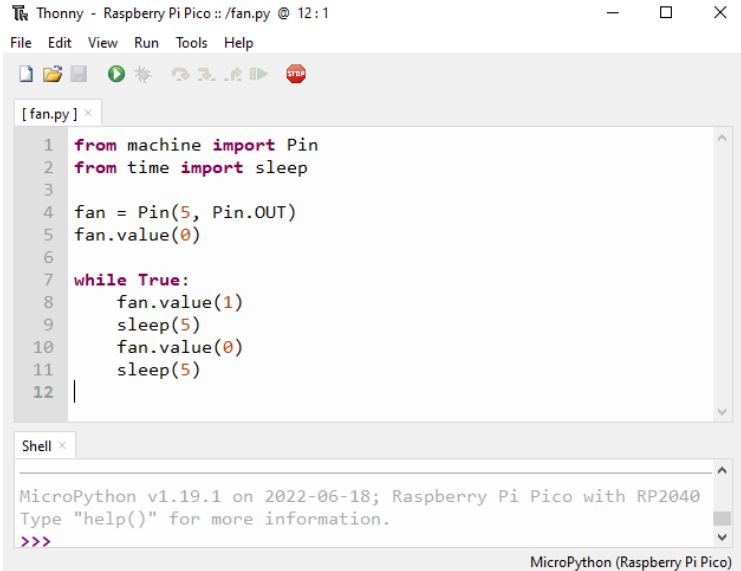
- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 1 x Diode
- 3 x Male-to-male jumper wires
- 1 x DC motor (small fan)
- 1 x TIP-120 transistor

Wiring diagram



Code

MicroPython code for the tutorial:



```

Thonny - Raspberry Pi Pico :: /fan.py @ 12:1
File Edit View Run Tools Help

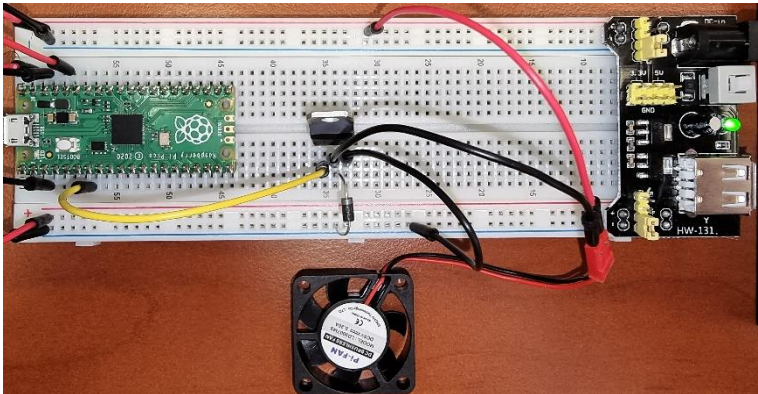
[ fan.py ] x
1 from machine import Pin
2 from time import sleep
3
4 fan = Pin(5, Pin.OUT)
5 fan.value(0)
6
7 while True:
8     fan.value(1)
9     sleep(5)
10    fan.value(0)
11    sleep(5)
12 |

Shell x
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:



8. SG-90 Servo Motor

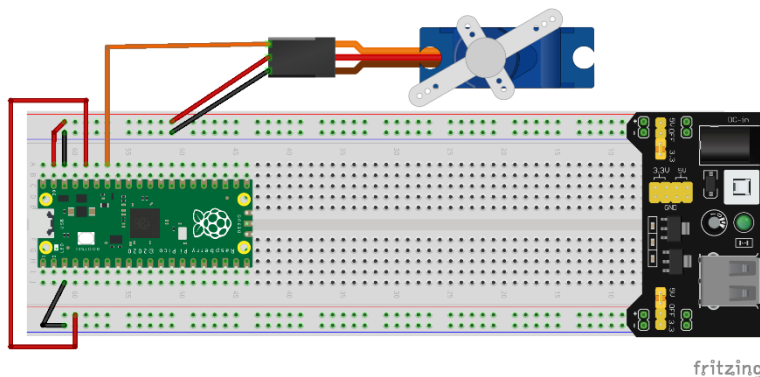
Description

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 materials

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

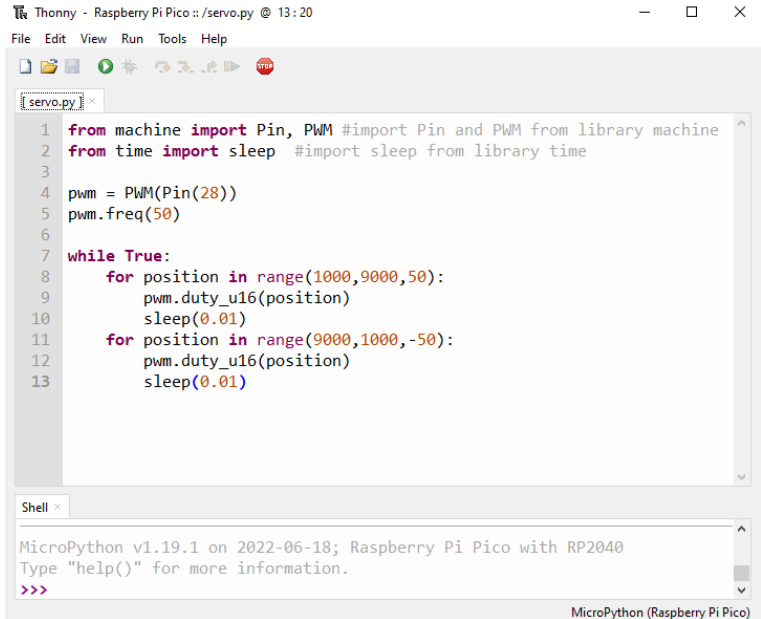
Wiring diagram



- 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

```

1 from machine import Pin, PWM #import Pin and PWM from library machine
2 from time import sleep #import sleep from library time
3
4 pwm = PWM(Pin(28))
5 pwm.freq(50)
6
7 while True:
8     for position in range(1000,9000,50):
9         pwm.duty_u16(position)
10        sleep(0.01)
11    for position in range(9000,1000,-50):
12        pwm.duty_u16(position)
13        sleep(0.01)
  
```

Shell

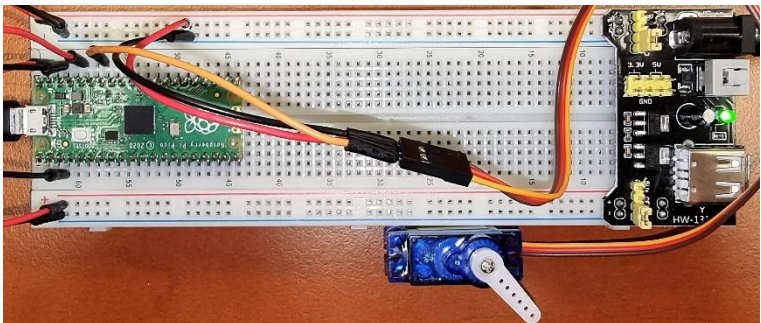
```

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:



9. OLED I2C SSD1306 Display

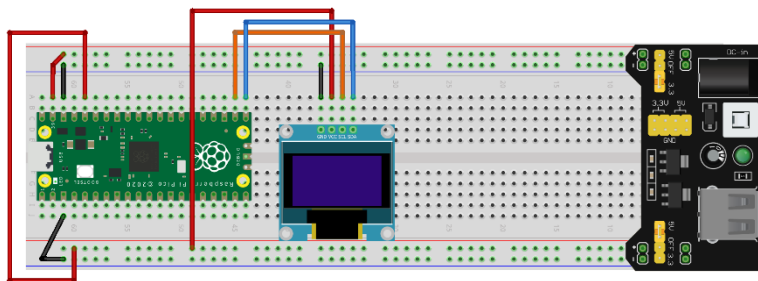
Description

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 materials

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

Wiring diagram



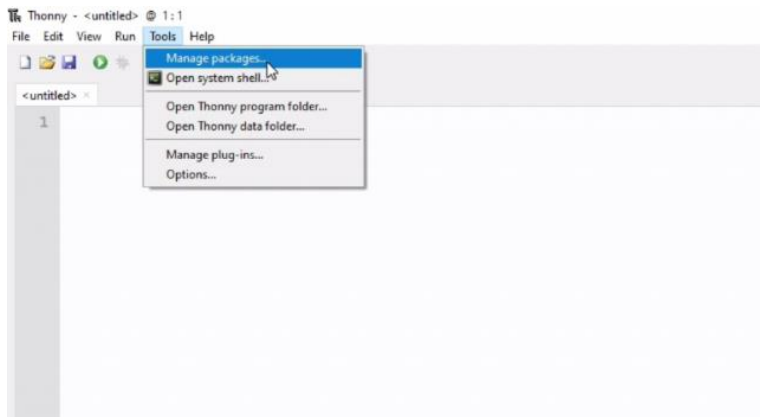
fritzing

- 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

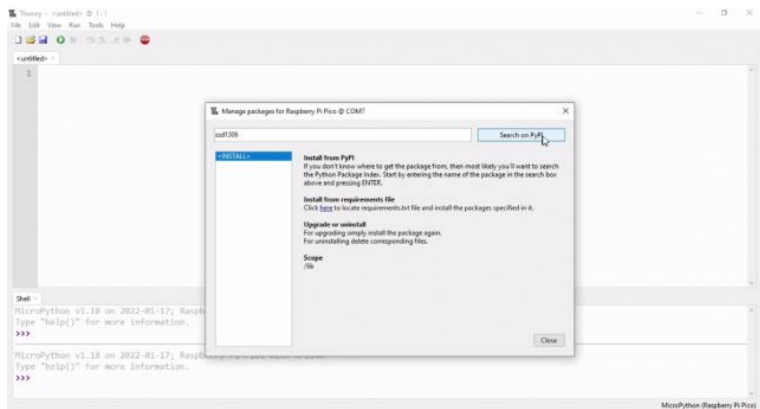
Code

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:

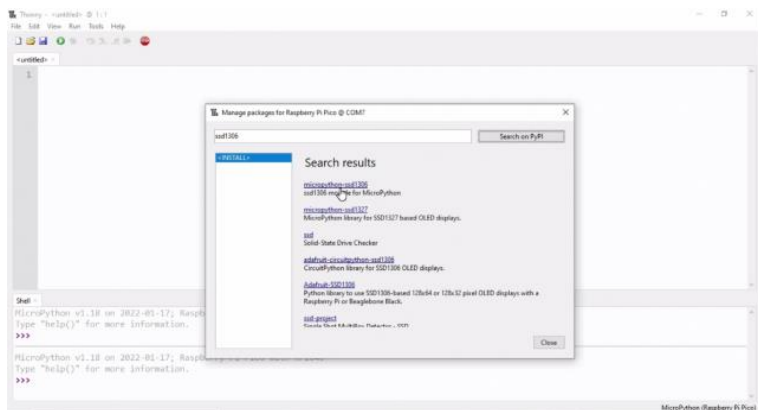
1. Open Thonny and go to **Tools** → **Manage packages...**



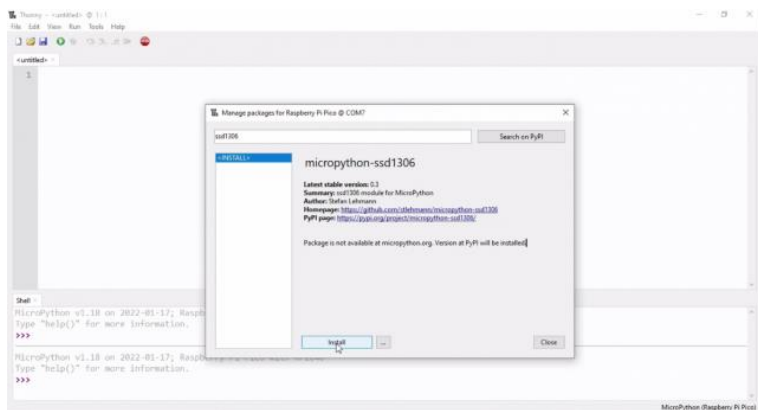
2. In the manage packages window, type **SSD1306** and click **Search**.



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



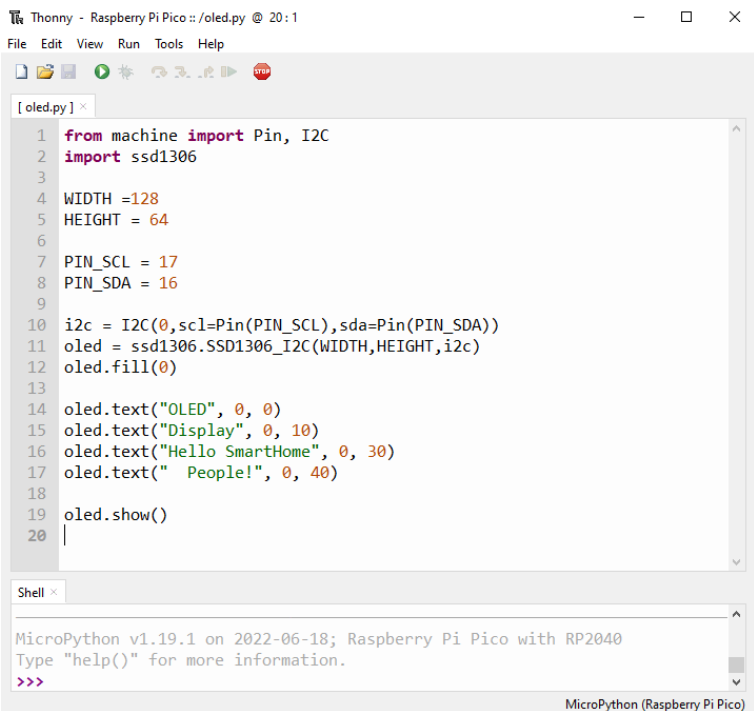
4. On the next window, click *Install*.



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] x
1  from machine import Pin, I2C
2  import ssd1306
3
4  WIDTH =128
5  HEIGHT = 64
6
7  PIN_SCL = 17
8  PIN_SDA = 16
9
10 i2c = I2C(0,scl=Pin(PIN_SCL),sda=Pin(PIN_SDA))
11 oled = ssd1306.SSD1306_I2C(WIDTH,HEIGHT,i2c)
12 oled.fill(0)
13
14 oled.text("OLED", 0, 0)
15 oled.text("Display", 0, 10)
16 oled.text("Hello SmartHome", 0, 30)
17 oled.text(" People!", 0, 40)
18
19 oled.show()
20 |

Shell x
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:



10. RFID Reader RC522

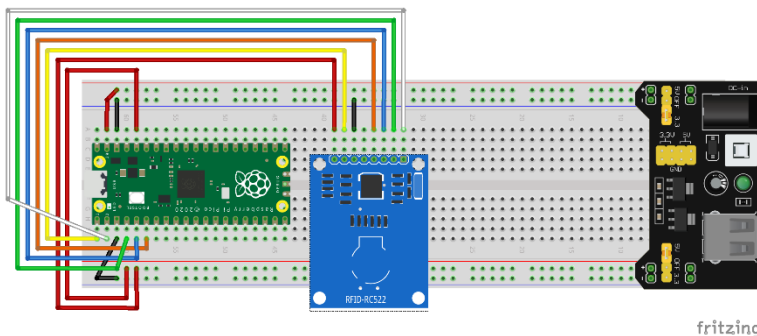
Description

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

Required materials

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 7 x Male-to-male jumper wires
- 1 x RFID RC522 module
- 1 x RFID Tag

Wiring diagram



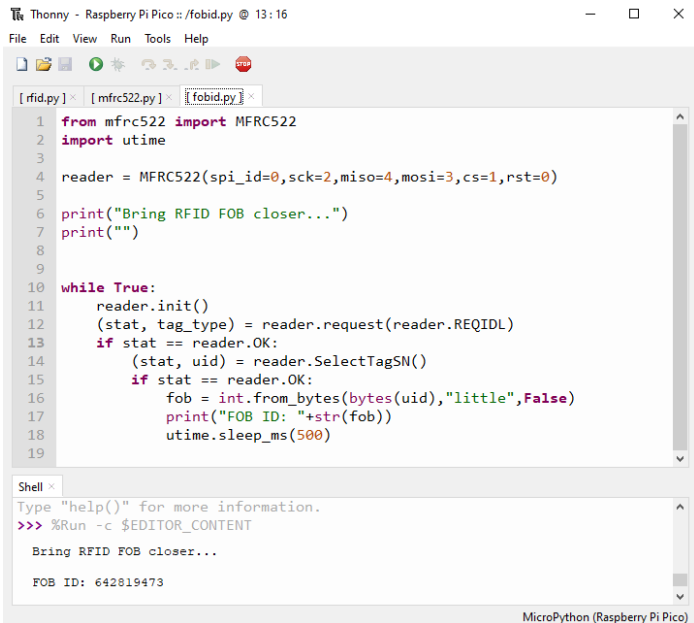
- 3v3 (red cable) is connected to 3v3 rail (+)
- GND (black cable) is connected to GND rail (-)
- RST (yellow cable) is connected to GPIO0 pin
- SDA (white cable) is connected to GPIO1 pin
- SCK (green cable) is connected to GPIO2 pin

- MOSI (blue cable) is connected to GPIO3 pin
- MISO (orange cable) is connected to GPIO4 pin

Code

Similar to the OLED display, we need an additional library that controls the RFID module, namely the MFRC522 library. We can download it from <https://github.com/pimylifeup/MFRC522-python/blob/master/mfrc522/MFRC522.py>. Download the file and open it in Thonny Python. Then click File → Save as... choose Raspberry Pi Pico and save your file under the name `mfrc522.py`.

Then, you need to identify the fob's ID to make the RFID reader to work. To do that you need to create a program that reads the RFID Fob and gives its ID. Check the program below:



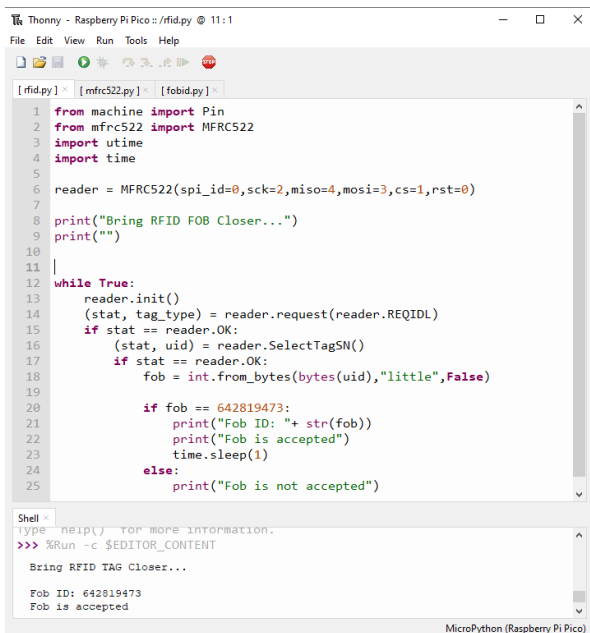
```

Thonny - Raspberry Pi Pico :: /fobid.py @ 13:16
File Edit View Run Tools Help

[rfid.py] x [mfrc522.py] x [fobid.py] x

1 from mfrc522 import MFRC522
2 import utime
3
4 reader = MFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=1,rst=0)
5
6 print("Bring RFID FOB closer...")
7 print("")
8
9
10 while True:
11     reader.init()
12     (stat, tag_type) = reader.request(reader.REQIDL)
13     if stat == reader.OK:
14         (stat, uid) = reader.SelectTagSN()
15         if stat == reader.OK:
16             fob = int.from_bytes(bytes(uid),"little",False)
17             print("FOB ID: "+str(fob))
18             utime.sleep_ms(500)
19
Shell x
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
Bring RFID FOB closer...
FOB ID: 642819473
MicroPython (Raspberry Pi Pico)
  
```

Click Play and scan the fob. This will give you its ID. Now, back to `rfid.py` program, you should change the number in line 20 so it matches the ID of your fob. Then click Save and run your program. MicroPython code for the tutorial:



```

Thonny - Raspberry Pi Pico :: rfid.py @ 11:1
File Edit View Run Tools Help

[rfid.py] x [mfr522.py] x [fobid.py] x
1 from machine import Pin
2 from mfr522 import MFRC522
3 import utime
4 import time
5
6 reader = MFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=1,rst=0)
7
8 print("Bring RFID FOB Closer...")
9 print("")
10
11 |
12 while True:
13     reader.init()
14     (stat, tag_type) = reader.request(reader.REQIDL)
15     if stat == reader.OK:
16         (stat, uid) = reader.SelectTagSN()
17         if stat == reader.OK:
18             fob = int.from_bytes(bytes(uid), "little", False)
19
20             if fob == 642819473:
21                 print("Fob ID: "+ str(fob))
22                 print("Fob is accepted")
23                 time.sleep(1)
24             else:
25                 print("Fob is not accepted")

Shell x
TYPE --help() for more information.
>>> %Run -c $EDITOR_CONTENT

Bring RFID TAG Closer...

Fob ID: 642819473
Fob is accepted

MicroPython (Raspberry Pi Pico)
  
```

Sample image

Image of how the tutorial looks using the provided hardware:



Tutorials with Sensors

11. Raindrop sensor

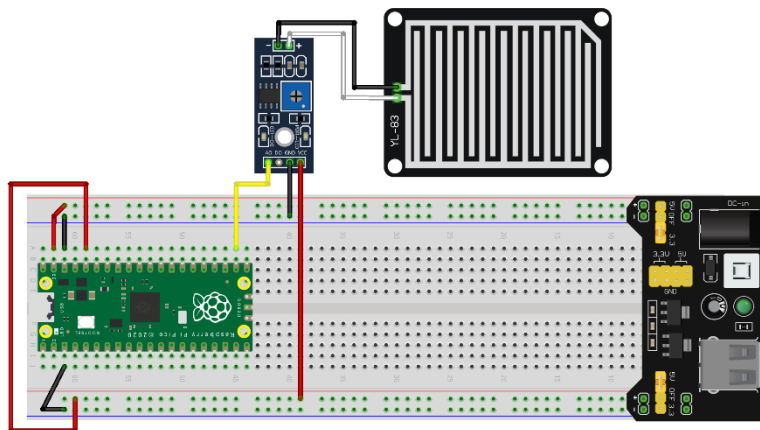
Description

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 materials

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

Wiring diagram

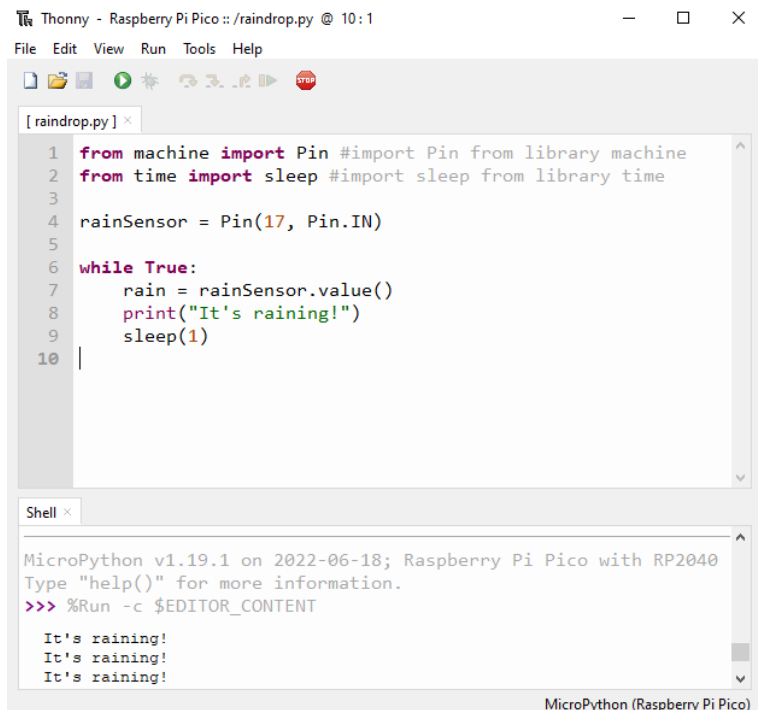


fritzing

- 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 @ 10: 1
File Edit View Run Tools Help

[ raindrop.py ] x
1 from machine import Pin #import Pin from library machine
2 from time import sleep #import sleep from library time
3
4 rainSensor = Pin(17, Pin.IN)
5
6 while True:
7     rain = rainSensor.value()
8     print("It's raining!")
9     sleep(1)
10 |

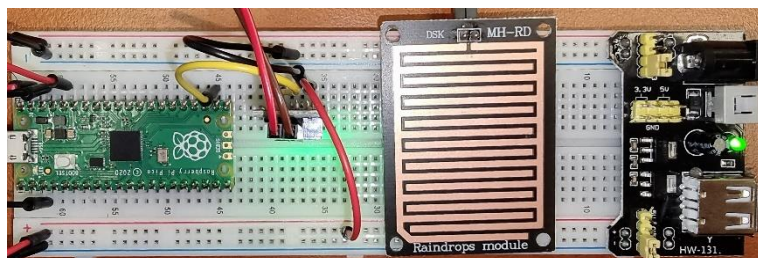
Shell x
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

It's raining!
It's raining!
It's raining!

MicroPython (Raspberry Pi Pico)
  
```

Sample image

Image of how the tutorial looks using the provided hardware:



12. HC-SR04 Ultrasonic Sensor

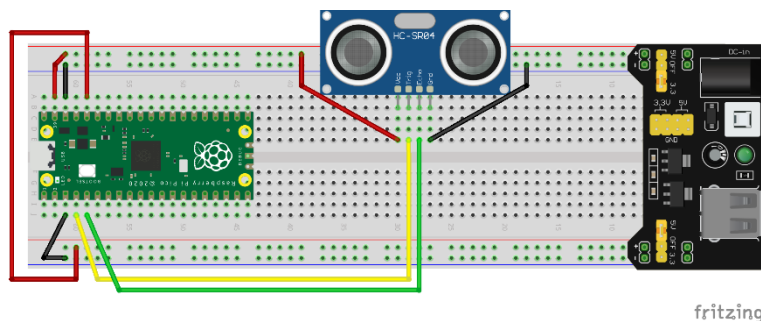
Description

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 materials

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

Wiring diagram

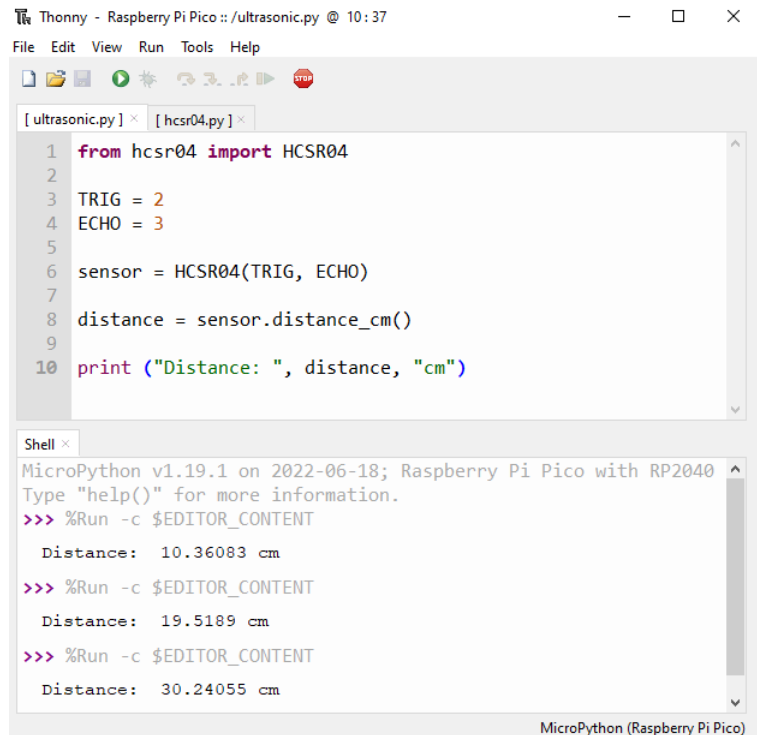


- VCC (red cable) is connected to 5V rail (+)
- GND (black cable) is connected to GND rail (-)
- TRIG (orange cable) is connected to GPIO2 pin
- ECHO (green cable) is connected to GPIO3 pin

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:



```

Thonny - Raspberry Pi Pico :: /ultrasonic.py @ 10:37
File Edit View Run Tools Help

[ ultrasonic.py ] x [ hcsr04.py ] x

1 from hcsr04 import HCSR04
2
3 TRIG = 2
4 ECHO = 3
5
6 sensor = HCSR04(TRIG, ECHO)
7
8 distance = sensor.distance_cm()
9
10 print ("Distance: ", distance, "cm")

Shell x
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
    Distance:  10.36083 cm
>>> %Run -c $EDITOR_CONTENT
    Distance:  19.5189 cm
>>> %Run -c $EDITOR_CONTENT
    Distance:  30.24055 cm

MicroPython (Raspberry Pi Pico)
  
```


MicroPython code without existing library:

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

File Edit View Run Tools Help

```

[ultrasonic.py] x
1  from machine import Pin
2  import utime
3
4  TRIG = Pin(2, Pin.OUT)
5  ECHO = Pin(3, Pin.IN)
6  def ultra():
7      TRIG.low()
8      utime.sleep_us(2)
9      TRIG.high()
10     utime.sleep_us(5)
11     TRIG.low()
12     while ECHO.value() == 0:
13         signaloff = utime.ticks_us()
14     while ECHO.value() == 1:
15         signalon = utime.ticks_us()
16     timepassed = signalon - signaloff
17     distance = (timepassed * 0.0343) / 2
18     print("The distance from object is ",distance,"cm")
19 while True:
20     ultra()
21     utime.sleep(1)
  
```

Shell x

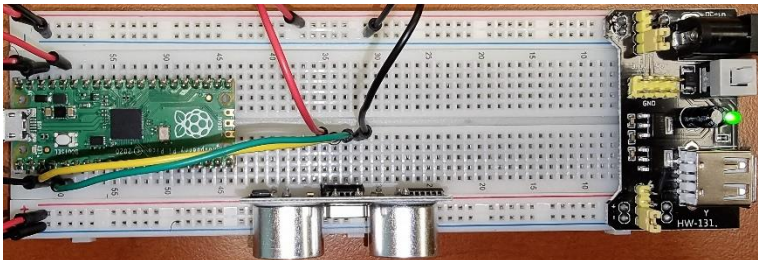
```

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:



13. PIR Motion Sensor

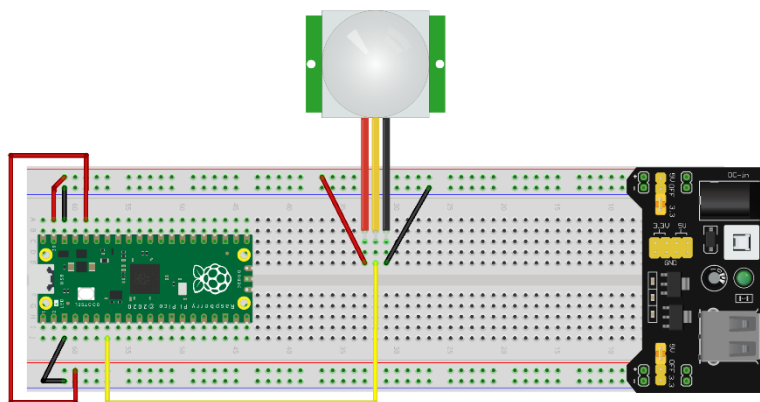
Description

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 materials

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

Wiring diagram

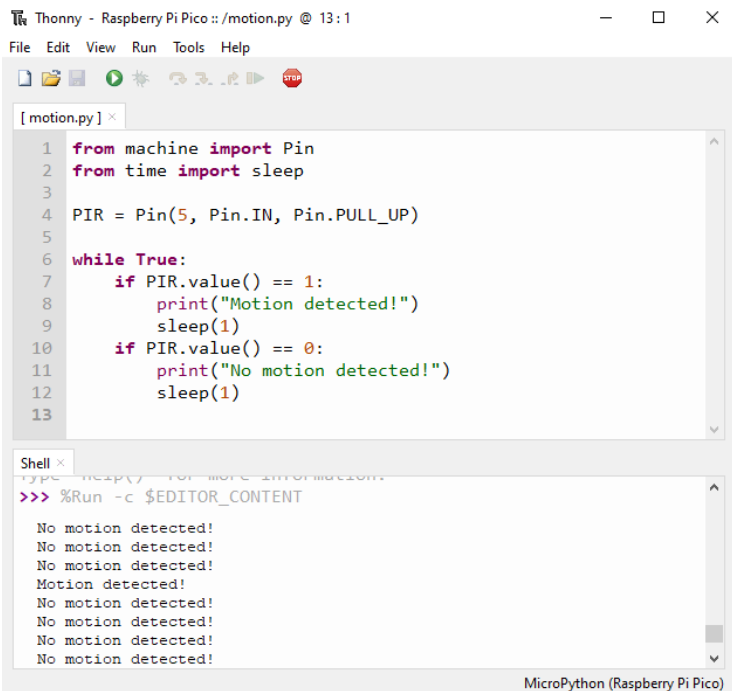


fritzing

- VCC (red cable) is connected to 5V rail (+)
- GND (black cable) is connected to GND rail (-)
- OUT (orange cable) is connected to GPIO5 pin

Code

MicroPython code for the tutorial:

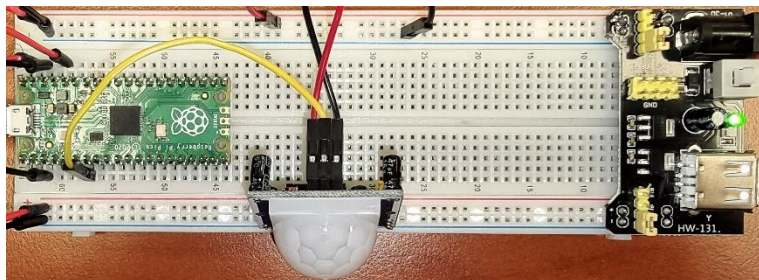


```

Thonny - Raspberry Pi Pico :: /motion.py @ 13:1
File Edit View Run Tools Help
[motion.py] x
1 from machine import Pin
2 from time import sleep
3
4 PIR = Pin(5, 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)
13
Shell x
Type help() for more information.
>>> %Run -c $EDITOR_CONTENT
No motion detected!
No motion detected!
No motion detected!
Motion detected!
No motion detected!
No motion detected!
No motion detected!
No motion detected!
No motion detected!
MicroPython (Raspberry Pi Pico)
  
```

Sample image

Image of how the tutorial looks using the provided hardware:



14. DHT11 Sensor

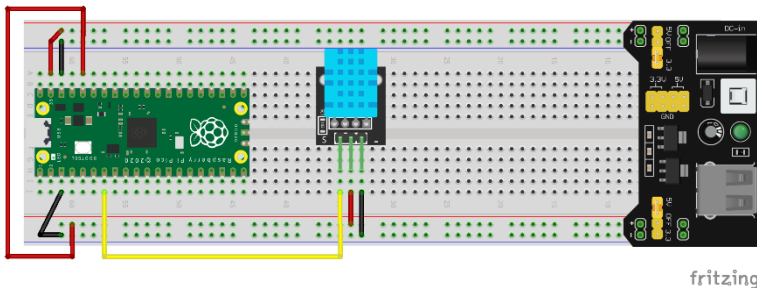
Description

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 materials

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

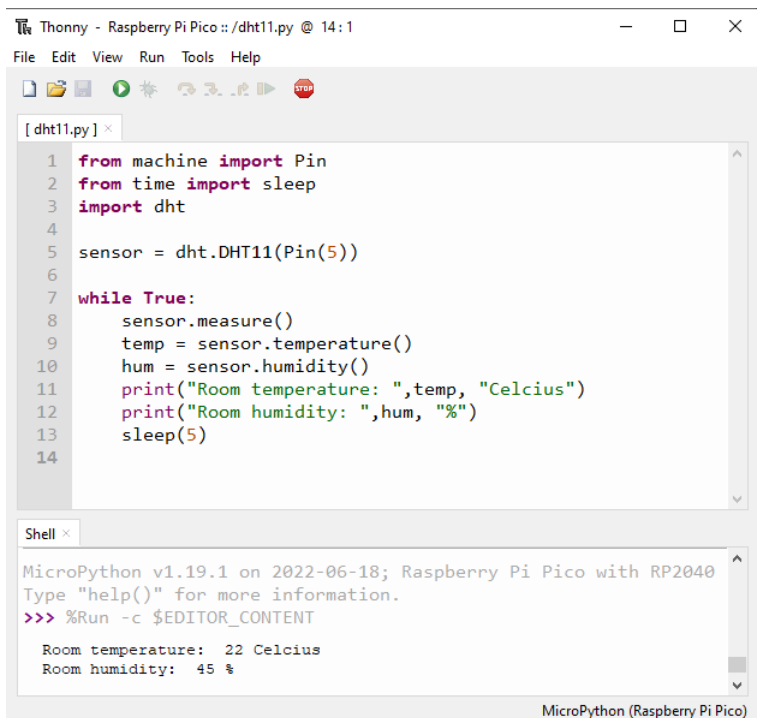
Wiring diagram



- VCC (red cable) is connected to 3v3 rail (+)
- GND (black cable) is connected to GND rail (-)
- S (green cable) is connected to GPIO5 pin

Code

MicroPython code for the tutorial:



```

Thonny - Raspberry Pi Pico :: /dht11.py @ 14: 1
File Edit View Run Tools Help

[dht11.py] x
1 from machine import Pin
2 from time import sleep
3 import dht
4
5 sensor = dht.DHT11(Pin(5))
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)
14

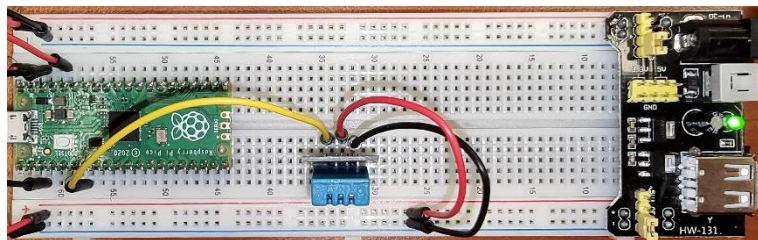
Shell x
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Room temperature: 22 Celcius
Room humidity: 45 %

MicroPython (Raspberry Pi Pico)
  
```

Sample image

Image of how the tutorial looks using the provided hardware:



15. Flame Sensor

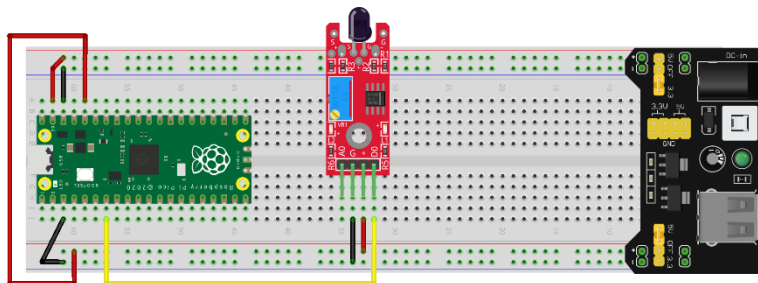
Description

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 materials

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

Wiring diagram

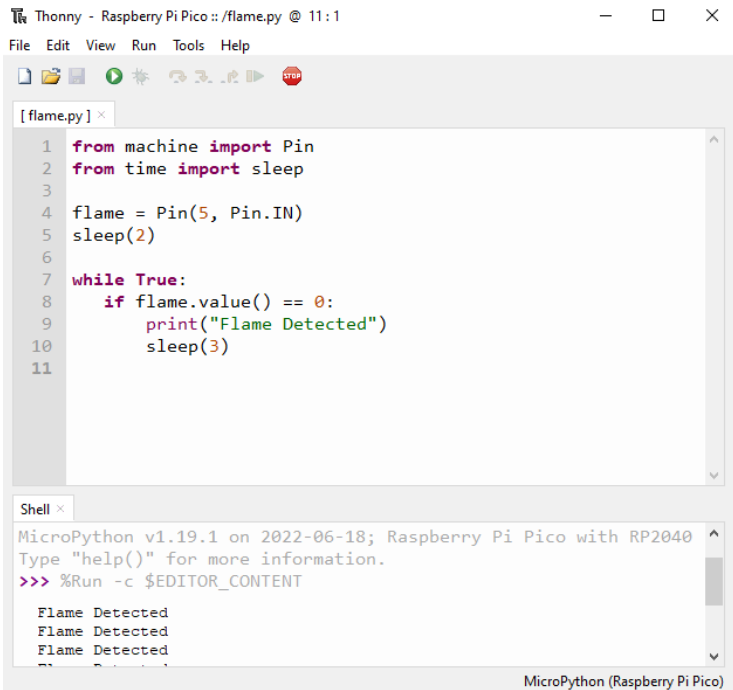


fritzing

- VCC (red cable) is connected to 5V rail (+)
- GND (black cable) is connected to GND rail (-)
- DO (green cable) is connected to GPIO5 pin

Code

MicroPython code for the tutorial:



Thonny - Raspberry Pi Pico :: /flame.py @ 11:1

File Edit View Run Tools Help

```
[flame.py] x
1 from machine import Pin
2 from time import sleep
3
4 flame = Pin(5, Pin.IN)
5 sleep(2)
6
7 while True:
8     if flame.value() == 0:
9         print("Flame Detected")
10        sleep(3)
11
```

Shell x

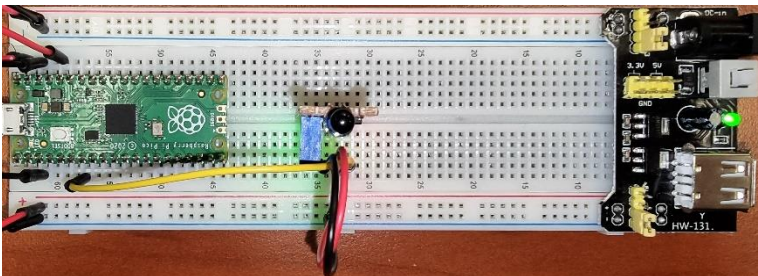
```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Flame Detected
Flame Detected
Flame Detected
```

MicroPython (Raspberry Pi Pico)

Sample image

Image of how the tutorial looks using the provided hardware:



16. MQ-135 Gas Detection Sensor

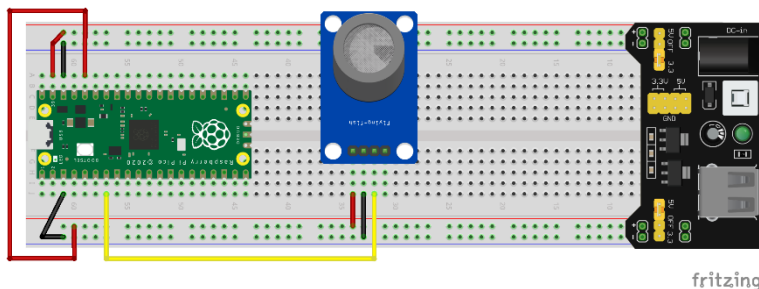
Description

In this tutorial you will learn how to connect and control the MQ-135 gas detection sensor. Open Thonny Python, then go to File → Save as..., choose Raspberry Pi Pico, and save your file under the name `gas.py`. Then it is time to connect the electronics and write your program. Please follow the instructions below.

Required materials

- 1 x Raspberry Pi Pico
- 1 x Full size breadboard
- 1 x Micro-USB cable
- 3 x Male-to-male jumper wires
- 1 x MQ-135 gas detection

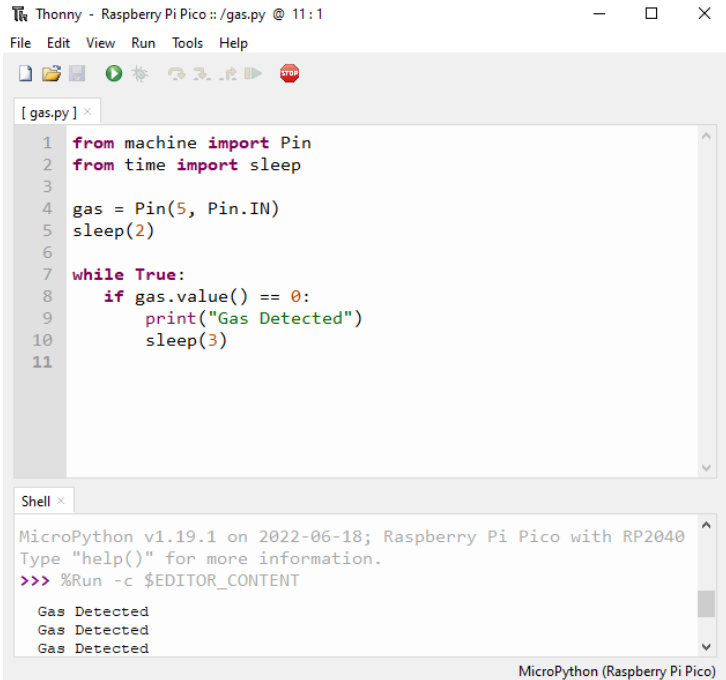
Wiring diagram



- VCC (red cable) is connected to 3v3 rail (+)
- GND (black cable) is connected to GND rail (-)
- DO/OUT (yellow cable) is connected to GPIO5 pin

Code

MicroPython code for the tutorial:



```

Thonny - Raspberry Pi Pico :: /gas.py @ 11:1
File Edit View Run Tools Help

[ gas.py ] x
1 from machine import Pin
2 from time import sleep
3
4 gas = Pin(5, Pin.IN)
5 sleep(2)
6
7 while True:
8     if gas.value() == 0:
9         print("Gas Detected")
10        sleep(3)
11

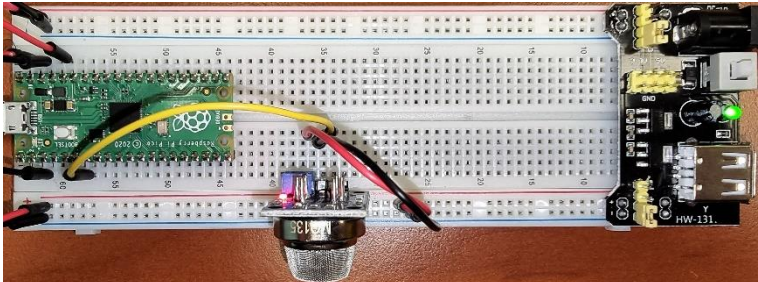
Shell x
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Gas Detected
Gas Detected
Gas 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	<code>from machine import Pin</code>	
Initialisation of the digital output object	<code>led = Pin(pin_value, Pin.OUT)</code>	<code>pin_value</code> from 0 to 40
Open digital output (3.3V output)	<code>led.value(1)</code>	ON
Close digital output (0V output)	<code>led.value(0)</code>	OFF

Digital Input		
Calling the Pin class	<code>from machine import Pin</code>	
Initialisation of the digital output object	<code>button = Pin(pin_value, Pin.IN)</code>	<code>pin_value</code> from 0 to 40
	<code>button = Pin(pin_value), Pin.IN, Pin.PULL_UP)</code>	Activation of PULL UP resistance
	<code>button = Pin(pin_value), Pin.IN, Pin.PULL_DOWN)</code>	Activation of PULL DOWN resistance
Input reading	<code>value = button.value(1)</code>	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	<code>from machine import PWM</code>	
Initialisation of the analog output	<code>led = PWM(Pin(pin_value), frequency)</code>	<code>pin_value</code> from 0 to 40 <code>frequency</code> in HZ, from 0 to 78125
Input reading	<code>led.duty(duty_cycle)</code>	<code>duty_cycle</code> from 0 to 1023 (0V output to 3.3V output respectively)

Analog Input		
Calling the ADC class	<code>from machine import ADC</code>	
Initialisation of the analog input	<code>pot = ADC(Pin(pin_value))</code>	pin_value can be GPIO26, GPIO27 and GPIO28
Declaration at which voltage the input will give its maximum value (in ESP32 usually 3.3V)	<code>pot.atten(ADC.ATTN_11DB)</code>	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)	<code>pot.width(ADC.WIDTH_10BIT)</code>	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	<code>value = pot.readl()</code>	value is an integer from 0 to the maximum of the range specified by the <code>ADC.WIDTH_#BIT</code> statement (see previous)

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

If statement structure	
<code>if <expr1>: <statement1> elif <expr2>:</code>	<expr#>: the control condition that must return True or False

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

While loop structure

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

For loop structure

<pre> for <var> in <iterable>: <statement (s)> </pre>	<p><iterable>: a collection of objects, for example a list containing numbers, alphanumerics, etc.</p> <p><var>: a variable to which the value of the next item in the collection <iterable> is assigned.</p> <p><statement(s)>: a set of instructions executed at each iteration</p>
<pre> for <var> in range(<start>, <end>, <step>): <statement (s)> </pre>	<p>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).</p> <p><var>: variable to which the value of the next element of the sequence produced by range is assigned.</p> <p><statement(s)>: a set of instructions executed in each iteration</p>

Miscellaneous		
DHT11	<code>import dht</code>	Importing the DHT library
	<code>sensor = dht.DHT11(Pin(pin_number))</code>	Initialisation of the sensor variable with its associated <code>pin_number</code> .
	<code>sensor.measure()</code>	Updating sensor values
	<code>temp = sensor.temperature()</code>	Saving current temperature value
	<code>hum = sensor.humidity()</code>	Saving current humidity value
OLED DISPLAY	<code>from machine import I2C</code>	Importing the I2C library
	<code>import ssd1306</code>	Importing the ssd1306 library
	<code>i2c = I2C(-1, scl=Pin(1), sda=Pin(0))</code>	Initialization of the i2c variable on the SCL & SDA pins of Pico
	<code>oled_width = 128</code> <code>oled_height = 64</code> <code>oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)</code>	Initializing the screen
	<code>oled.text('Hello, World 1!', 0, 0)</code> <code>oled.text('Hello, World 2!', 0, 10)</code> <code>oled.text('Hello, World 3!', 0, 20)</code>	Storing messages in the screen buffer
	<code>oled.show()</code>	Showing messages (necessary to display messages stored in the screen buffer)
	<code>display.pixel(3, 4, 1)</code>	Set the pixel located at position (x,y) on the screen, with x=3 & y=4, to state 1 (i.e. display)