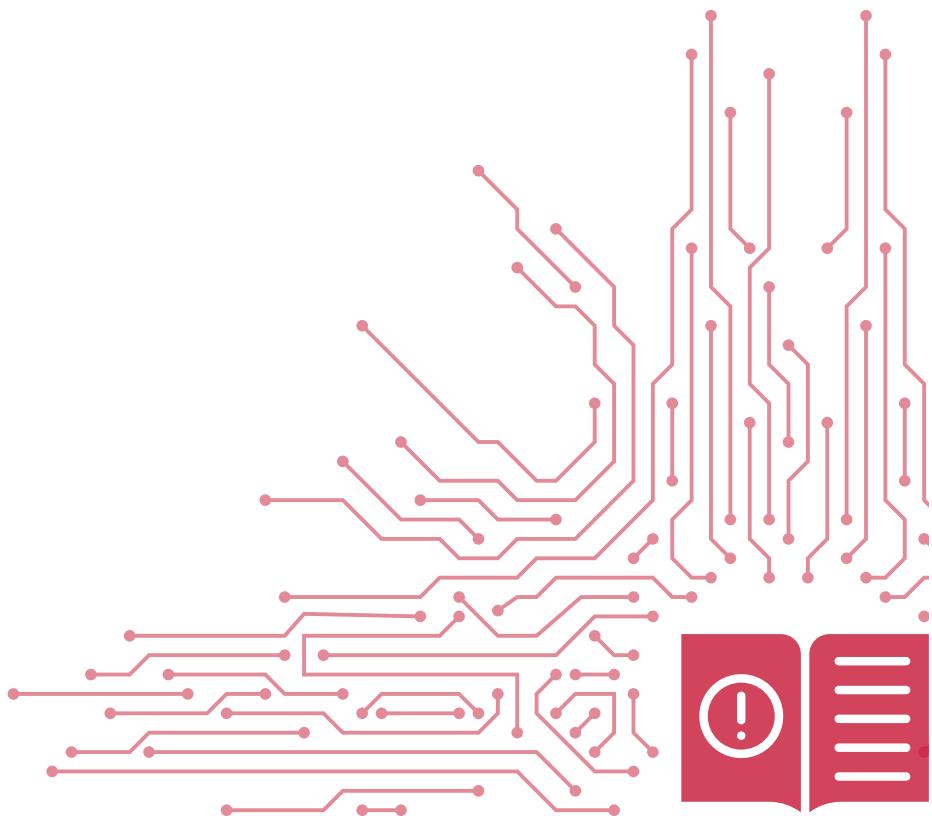


Raspberry Pi Pico

GUIDE



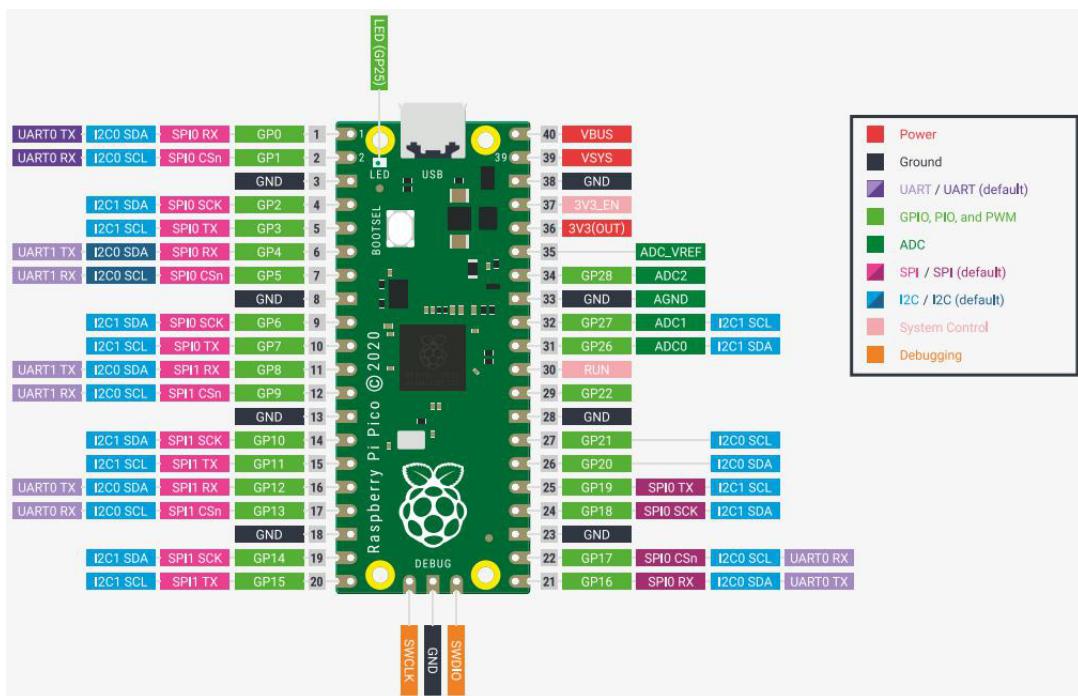
CONTENTS

1. En savoir plus sur la carte mère Raspberry Pi Pico+	FR-3
1.1 De quoi avez-vous besoin	FR-3
1.1.1 Carte mère de Raspberry Pi Pico	FR-4
1.1.2 Téléchargeur Micro USB	FR-4
1.1.3 Planche à pain	FR-4
1.1.4 Cavalier	FR-6
2. Préparation à l'apprentissage de Raspberry Pi Pico	FR-6
2.1 Introduction à la bouche d'E/S du Raspberry Pi Pico	FR-6
2.2 Qu'est-ce que la programmation Micro Python.....	FR-7
2.2.1 Raspberry Pi Pico installe le micrologiciel de prise en charge de Micro Python	FR-7
2.2.3 Introduction à la bibliothèque Micro Python de la carte mère Raspberry Pi Pico	FR-8
3. Utilisez le logiciel thonny pour le développement Python	FR-9
3.1 Sélectionnez la bonne programmation Raspberry Pi Pico.....	FR-13
3.2 Modifier la taille de la police.....	FR-14
3.3 Écrire le programme de clignotement des voyants à l'aide de Thonny	FR-15
4. Programmation de cas.....	FR-19
4.1 Expériences de diodes électroluminescentes	FR-19
4.2 Expérience de ventilateur de moteur à courant continu	FR-20
4.3 Bande LED	FR-21
4.4 Capteur d'humidité	FR-22
4.5 Capteur de niveau de liquide	FR-23
4.6 Capteur UV.....	FR-24
4.7 Capteur de sortie laser.....	FR-25
4.8 Capteur d'inclinaison.....	FR-26
4.9 Capteur de son.....	FR-27
4.10 Capteur PIR.....	FR-28
4.11 Capteur à effet Hall	FR-29
4.12 Capteur de gaz	FR-30
4.13 Capteur infrarouge réfléchissant.....	FR-31
4.14 Capteur de flamme.....	FR-32
4.15 Capteur de température-humidité.....	FR-33
4.16 Capteur d'horloge	FR-34
4.17 Capteur de rotation	FR-36
4.18 Capteur de température	FR-37
4.19 Capteur à ultrasons	FR-38
4.20 Moteur de direction	FR-40
4.21 Télécommande infrarouge	FR-41
4.22 Capteur de couleur	FR-42
4.23 Relais	FR-43

1. En savoir plus sur la carte mère Raspberry Pi Pico+

Le 21 janvier 2021, la Fondation Raspberry Pi a lancé la carte mère Raspberry Pi Pico, un produit à microprocesseur. Le produit est basé sur le nouveau processeur RP2040 auto-développé par la Fondation Raspberry Pi, qui ne coûte que 4 \$. Raspberry Pi Pico est signalé comme un microprocesseur, il est spécialisé dans la communication E/S à faible latence et l'entrée de signal analogique, à faible consommation d'énergie, peut compenser la faiblesses de la carte mère Raspberry Pi dans le domaine du microprocesseur.

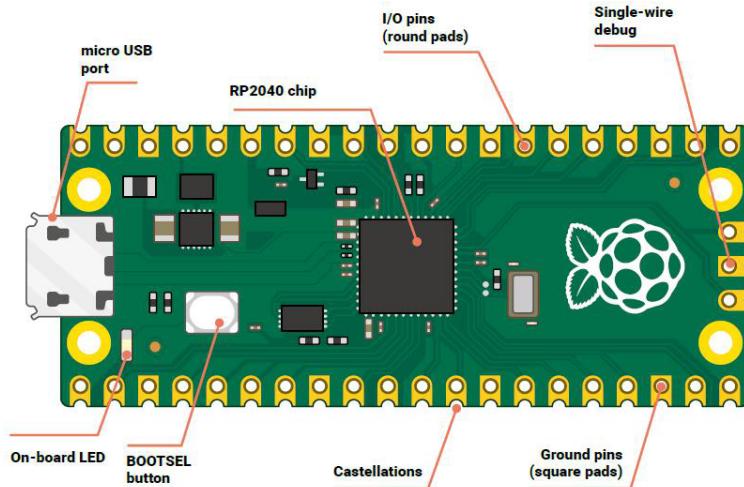
La carte mère Raspberry Pi Pico est devenue un nouveau favori des fondateurs du monde, en particulier sur la base des nouvelles fonctionnalités de la programmation Micro Python actuellement à la mode, nous permettant d'apprendre la programmation Python à faible coût entièrement via la carte mère Raspberry Pi Pico.



1.1 De quoi avez-vous besoin

Pour créer un système complet d'apprentissage et de développement de Raspberry Pi Pico, vous avez besoin d'au moins une carte mère Raspberry Pi Pico, d'un câble de données Micro USB (carte d'extension de capteur Pico Block en option) et d'un simple port d'E/S et une carte d'essai, afin de mieux utiliser le système de développement de Raspberry Pi Pico, vous pouvez également assortir des accessoires tels que des modules de capteurs, des moniteurs LCD, des claviers, etc., allons-y un par un :

1.1.1 Carte mère de Raspberry Pi Pico



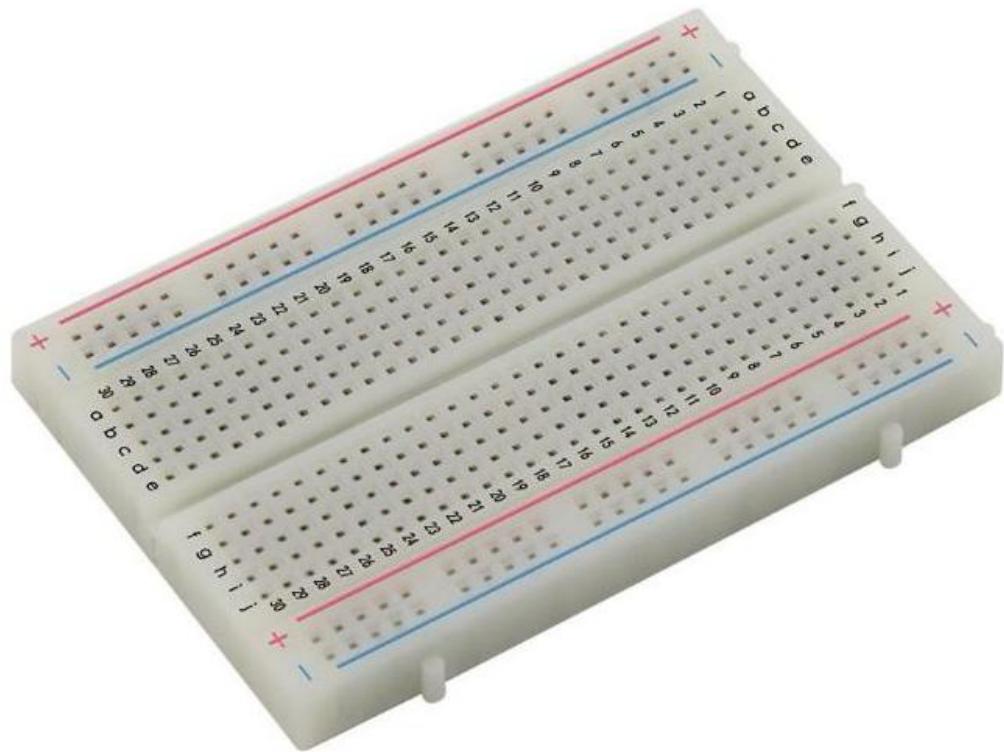
- ◆ Processeur: ARM Cortex-M0+ double cœur 32 bits à 48 MHz, configurable jusqu'à 133 MHz
- ◆ RAM: 264 Ko de SRAM dans six banques configurables indépendamment
- ◆ Stockage: 2 Mo de mémoire flash externe
- ◆ GPIO: 26 broches
- ◆ ADC: 3 broches ADC 12 bits
- ◆ PWM: Huit tranches, deux sorties par tranche pour 16 au total
- ◆ Horloge: Horloge et minuterie sur puce précises avec année, mois, jour, jour de la semaine, heure, seconde et calcul automatique des années bissextilles
- ◆ Capteurs: Capteur de température sur puce connecté au canal ADC 12 bits
- ◆ LED: LED embarquée adressable par l'utilisateur
- ◆ Connectivité de bus: 2 × UART, 2 × SPI, 2 × I2C, entrée/sortie programmable (PIO)
- ◆ Débogage matériel: débogage à fil unique (SWD)
- ◆ Options de montage: broches traversantes et crénélées (non peuplées) avec 4 orifices de montage
- ◆ Alimentation: 5 V via un connecteur micro USB, 3,3 V via une broche 3V3 ou 2–5 V via une broche VSYS

1.1.2 Téléchargeur Micro USB

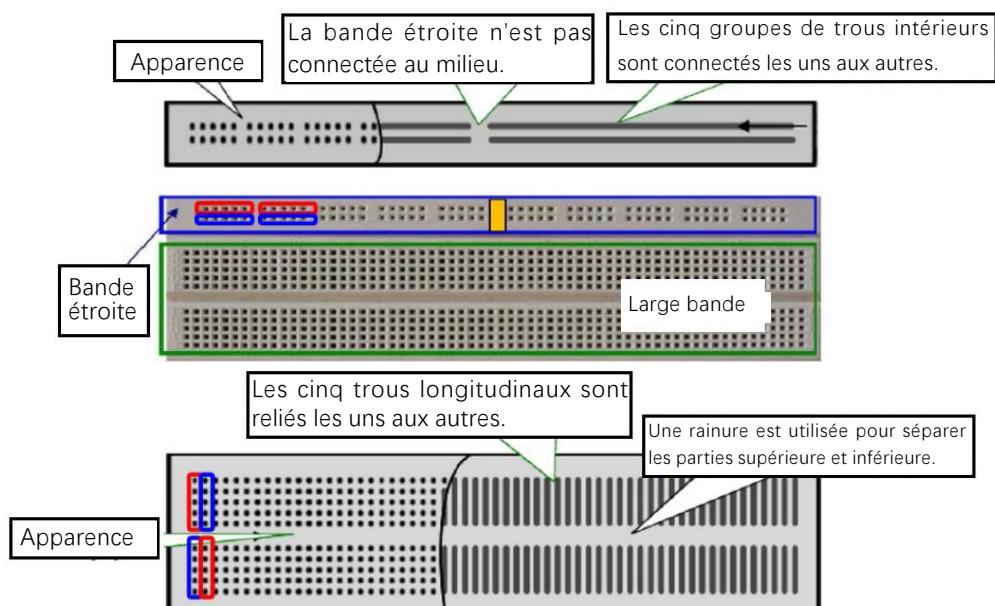
Téléchargement du programme de la carte mère Raspberry Pi Pico et interface d'alimentation USB pour l'interface Micro USB, nous devons donc utiliser un câble Micro USB de meilleure qualité pour le téléchargement du programme, si le côté PC lors de l'utilisation, il est préférable de ne pas utiliser l'extension USB Hub, car cela conduire au téléchargement instable des programmes, il est recommandé d'utiliser l'USB avec le côté PC pour le téléchargement du programme et l'alimentation, nécessite l'USB pour fournir un 5V stable, Dans les expériences de capteur ultérieures, un courant suffisant est nécessaire pour alimenter l'alimentation.

1.1.3 Planche à pain

La planche à pain est conçue pour les expériences non soudées dans les circuits électroniques car il y a beaucoup de petites prises sur la carte. Parce qu'une variété de composants électroniques peuvent être insérés ou débranchés selon les besoins, et pour éliminer le soudage, économiser du temps d'assemblage de circuits et que les composants peuvent être réutilisés, la planche à pain est donc idéale pour l'assemblage, la mise en service et la formation de circuits électroniques.

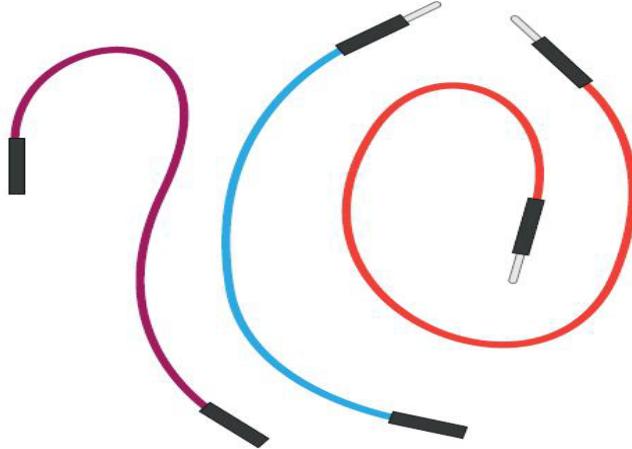


La structure interne de la planche à pain



1.1.4 Cavalier

Il existe trois types de cavaliers : mâle à mâle, femelle à femelle et mâle/femelle.



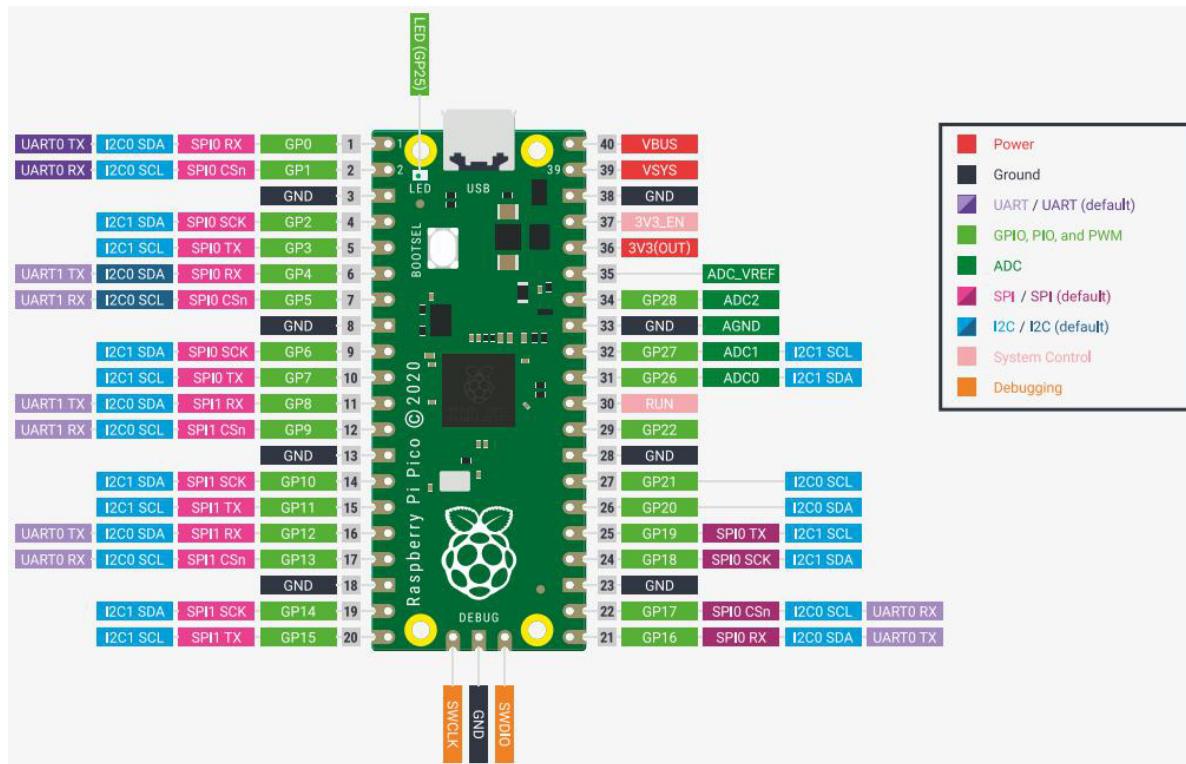
2. Préparation à l'apprentissage de Raspberry Pi Pico

La carte mère Raspberry Pi Pico, qui est officiellement lancée par la carte Raspberry Pi Pico, est programmée en langage Python et dispose d'un analyseur Micro python installé dans le processeur RP2040, nous pouvons donc programmer à l'aide de la carte mère Python Pico, et la méthode de programmation officielle est également disponible en C/C++, un choix de lecteur avancé qui nécessite beaucoup de logiciels pour le supporter.

2.1 Introduction à la bouche d'E/S du Raspberry Pi Pico

Raspberry Pi Pico propose des ports 40 broches à utiliser, avec 26 broches GPIO multifonctions, 2 SPI et 2 contrôleurs I2C, 2 UART et 16 PWM interfaces de contrôle. 3 interfaces d'acquisition analogique ADC 12 bits.

La définition de la broche est la suivante:



2.2 Qu'est-ce que la programmation Micro Python

Micro Python est un logiciel complet pour le langage de programmation Python3, écrit en C et exécuté sur un microcontrôleur, et Micro Python est un compilateur Python complet et un système d'exploitation fonctionnant sur le matériel du microcontrôleur. Fournit à l'utilisateur une invite interactive (REPL) pour LISENG TECHNOLOGY LIMITED exécuter immédiatement les commandes prises en charge. En plus de la bibliothèque Python principale sélectionnée, Micro Python inclut les modules matériels sous-jacents qui donnent accès aux programmeurs.

Les détails peuvent être consultés sur : <https://www.micropython.org>

2.2.1 Raspberry Pi Pico installe le micrologiciel de prise en charge de Micro Python

Si la carte mère Raspberry Pi Pico prend en charge la programmation Micro Python, nous devons télécharger le fichier UF2 sur le site officiel et le copier sur la clé USB de Pico sur l'ordinateur et aller sur le site officiel : <https://www.raspberrypi.org/documentation/rp2040/demarrage/>



Welcome to Raspberry Pi RP2040

Welcome to Raspberry Pi RP2040, a microcontroller designed here at Raspberry Pi.

Whether you have a Raspberry Pi Pico or another RP2040-based microcontroller board, everything you need to get started is here. You'll find support for getting started with C/C++ or MicroPython on Raspberry Pi Pico, and links to resources for other boards that use RP2040. There are also links to the technical documentation for both the Raspberry Pi Pico microcontroller board and our RP2040 microcontroller chip.

RP2040 On Board About Raspberry Pi Pico Getting started MicroPython Getting started C/C++

Getting started with MicroPython

Drag and drop MicroPython

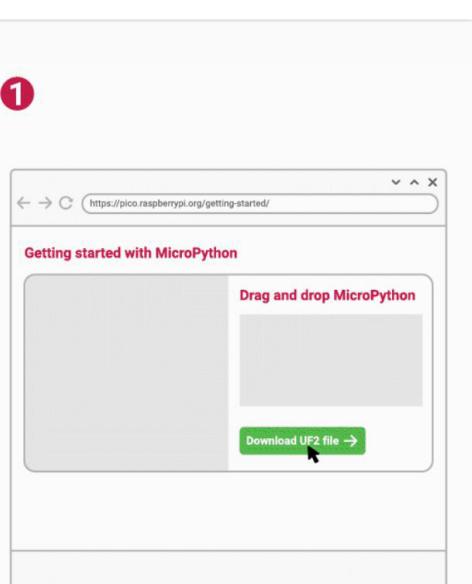
You can program your Pico by connecting it to a computer via USB, then dragging and dropping a file onto it, so we've put together a [downloadable UF2](#) file to let you install MicroPython more easily.

1. Download the MicroPython UF2 file by clicking the button below.
2. Push and hold the BOOTSEL button and plug your Pico into the USB port of your Raspberry Pi or other computer. Release the BOOTSEL button after your Pico is connected.
3. It will mount as a Mass Storage Device called RPI-RP2.
4. Drag and drop the MicroPython UF2 file onto the RPI-RP2 volume. Your Pico will reboot. You are now running MicroPython.

You can access the REPL via USB Serial. Our [MicroPython documentation](#) contains step-by-step instructions for connecting to your Pico and programming it in MicroPython.

Download UF2 file →

1



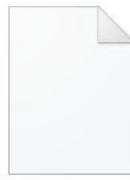
Getting started with MicroPython

Drag and drop MicroPython

Download UF2 file →

Select download UF2 file to the local disk

L'image suivante est un fichier UF2 obtenu, qui peut être officiellement mis à jour, donc le nom changera:



rp2-pico-20210
205-unstable-v
1.14-8-g1f800c
ac3.uf2

Connectez l'interface USB de la carte mère Pico avec le câble Micro USB et l'interface USB côté PC, si la carte mère sans graver le firmware produira une clé USB avec le nom : RPI-RP2 ;

Étape 1 : maintenez d'abord le "BOOTSEL" enfoncé, puis insérez le port USB côté ordinateur ;

Étape 2: insérez l'extrémité de l'ordinateur et relâchez le bouton de "BOOTSEL", vous pouvez afficher le nom de la clé USB RPI-RP2, puis le fichier UF2 précédemment téléchargé copié sur la clé USB, la clé USB disparaîtra automatiquement ;

Étape 3 : Ouvrez PC Device Manager et nous pouvons trouver le périphérique série virtuel dans le port, comme indiqué ci-dessous:



Noter:

Si nous devons confirmer s'il s'agit de COM4 illustré ci-dessus, nous pouvons débrancher l'appareil pour voir si le port disparaît, et si c'est le cas, cela signifie que l'appareil est notre carte mère Pico et prend en charge la programmation par Micro Python.

2.3 Introduction à la bibliothèque Micro Python de la carte mère Raspberry Pi Pico

La Fondation Raspberry Pi fournit un document PDF SDK Python très détaillé qui détaille les différentes fonctions intégrées, ainsi que l'utilisation spécifique des fonctions correspondantes et les notes de cas fournies, que nous devons utiliser pour interroger les différentes fonctions intégrées pour la mise en œuvre.

Documentation

Documentation for Raspberry Pi Pico and other RP2040-based boards.

- **Raspberry Pi Pico Datasheet**
An RP2040-based microcontroller board
- **RP2040 Datasheet**
A microcontroller by Raspberry Pi
- **Hardware design with RP2040**
Using RP2040 microcontrollers to build boards and products

- **Getting started with Raspberry Pi Pico**
C/C++ development with Raspberry Pi Pico and other RP2040-based microcontroller boards
- **Raspberry Pi Pico C/C++ SDK**
Libraries and tools for C/C++ development on RP2040 microcontrollers
- **Raspberry Pi Pico Python SDK**
A MicroPython environment for RP2040 microcontrollers

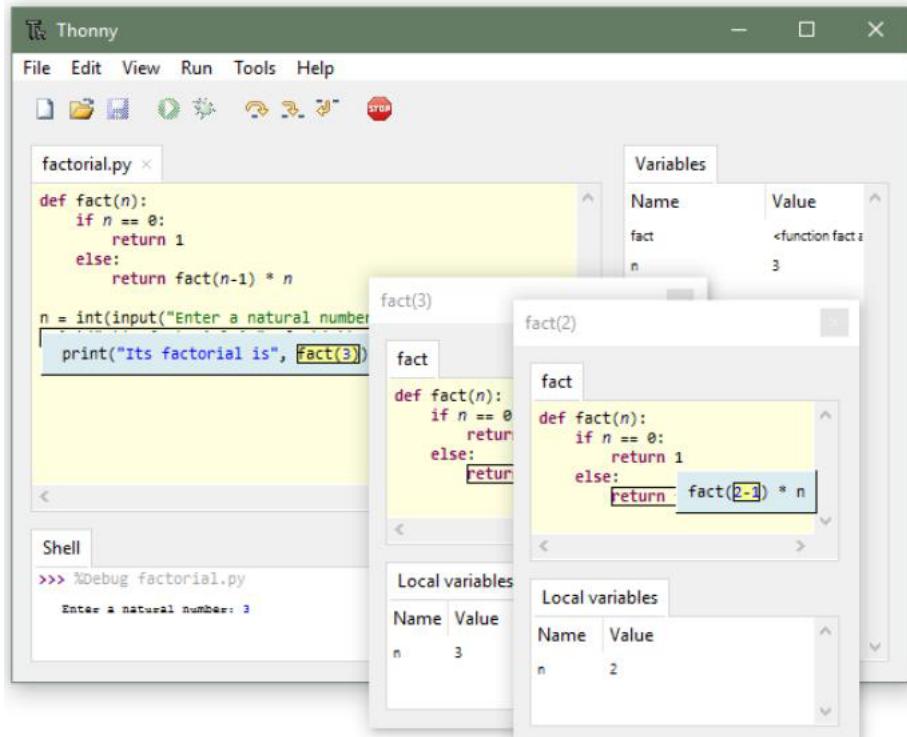
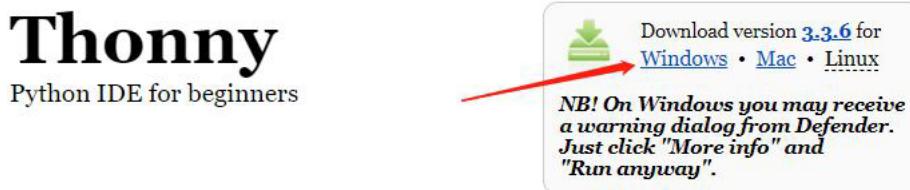
The API level Doxygen documentation for the Raspberry Pi Pico C/C++ SDK is available [as a micro-site](#), and frequent questions are answered in the [Frequently Asked Questions \(FAQ\)](#) document.

3. Utilisez le logiciel thonny pour le développement Python

La programmation Python a évolué jusqu'à nos jours avec un certain nombre de logiciels de programmation IDE très puissants, tels que PyCharm DE, Eclipse, Spyder et Visual Studio Code, mais pour les débutants, choisir un IDE léger et facile à utiliser est la meilleure option, et Thonny est ce que nous appelons un petit IDE, sous Windows Disponible sur, MAC ou Linux, il prend en charge la coloration de la syntaxe, l'auto-complément du code, le débogage, et surtout, le compilateur Python sur la carte mère Raspberry Pi Pico.

Site officiel : <https://thonny.org/>

Accédez au site Web pour télécharger la dernière version du logiciel Thonny, comme suit:

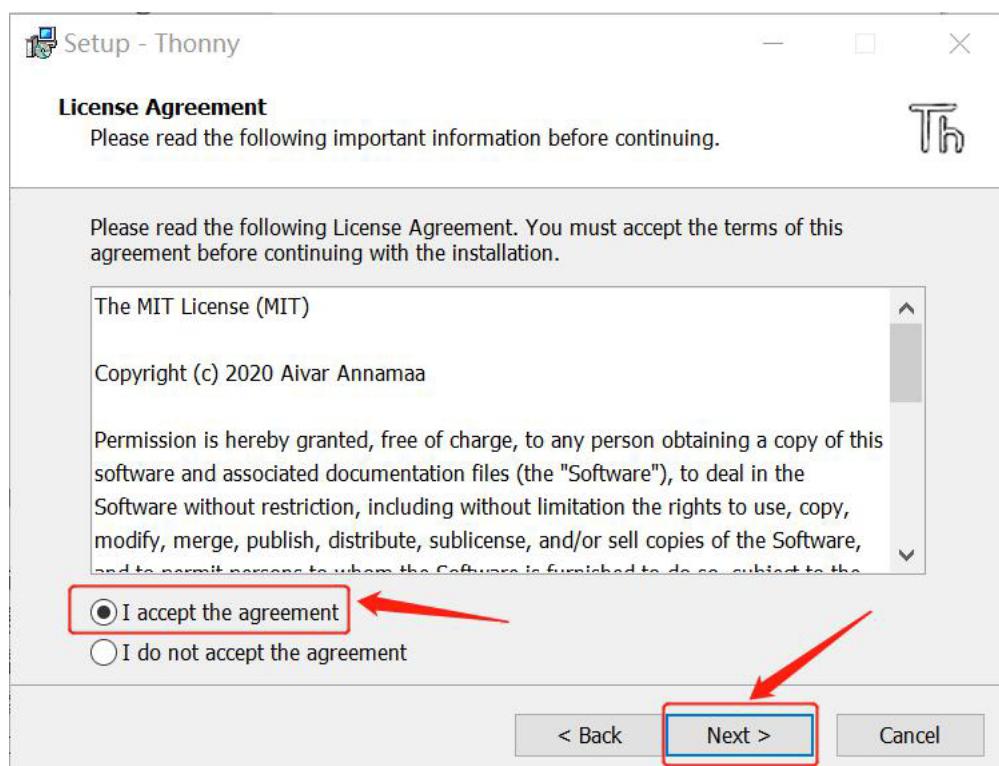
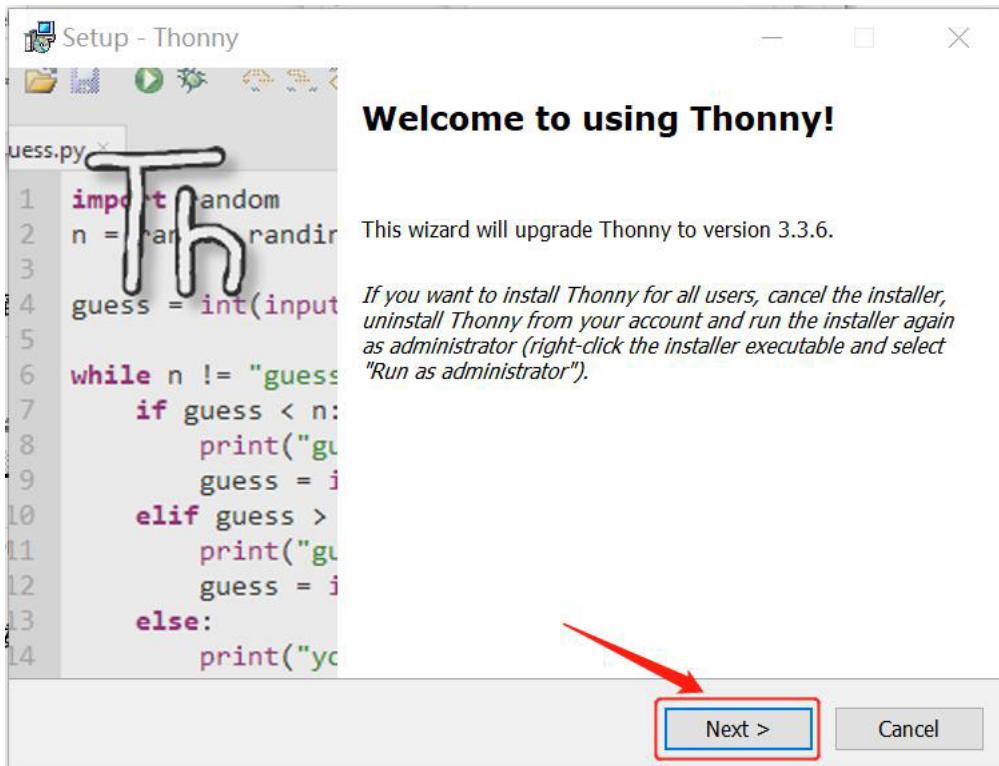


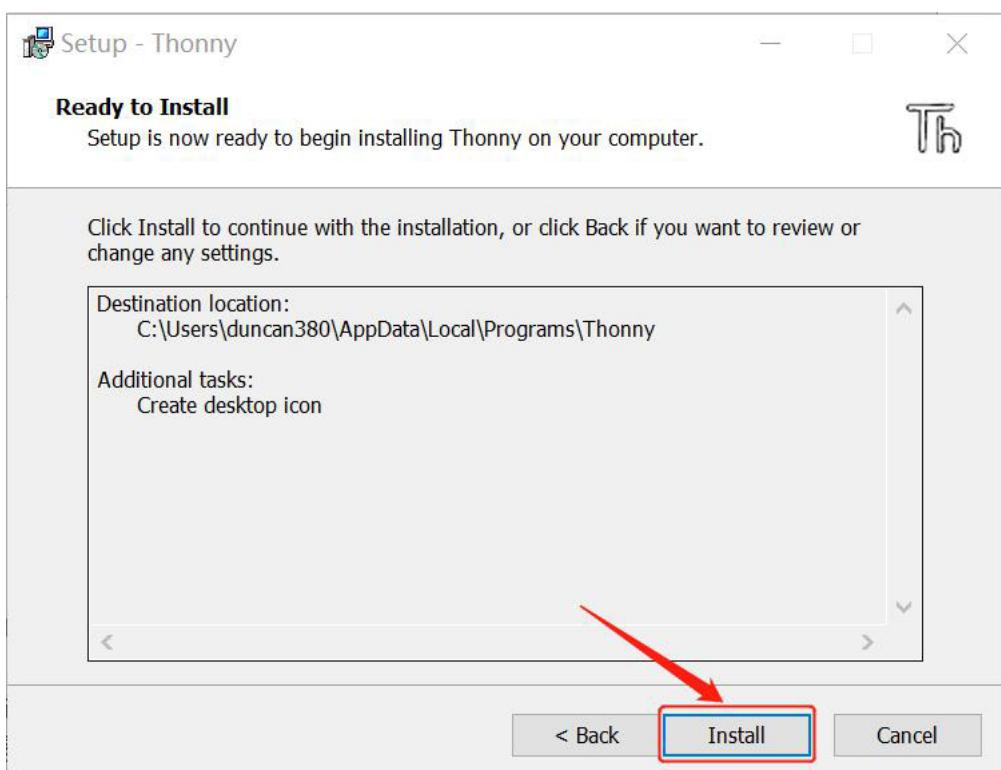
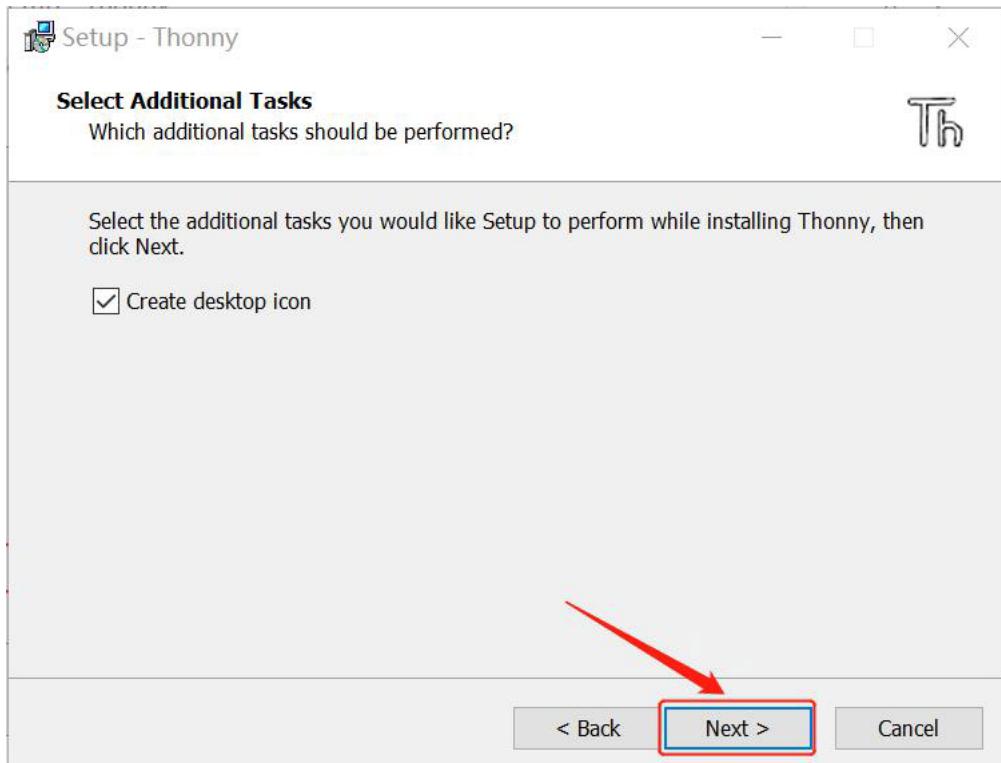
Après le téléchargement dans la zone locale, sélectionnez le programme d'installation correspondant, comme indiqué ci-dessous:

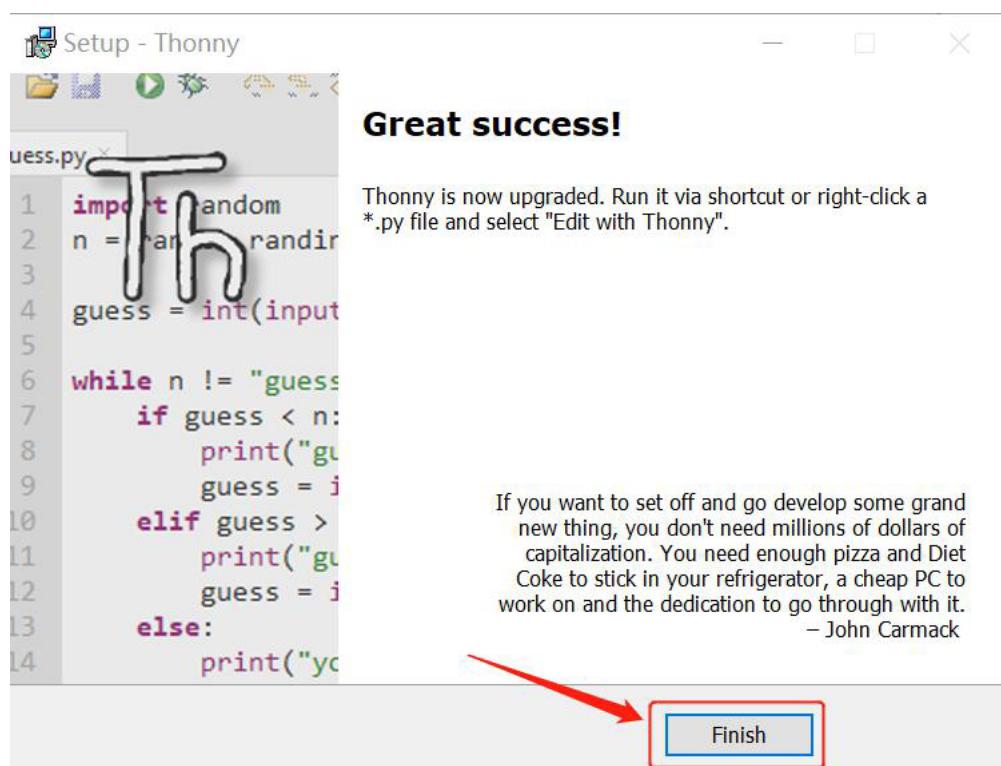
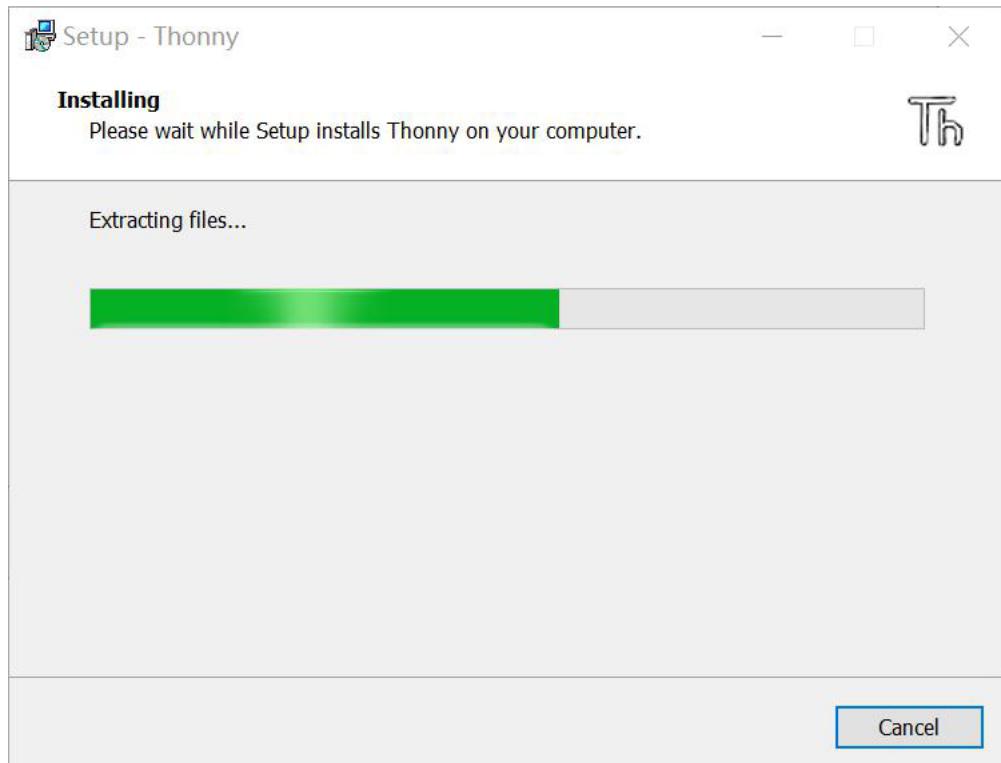


thonny-3.3.6

Double-cliquez sur le programme d'installation et affichez l'interface d'installation suivante, pointez sur la touche NEXT:







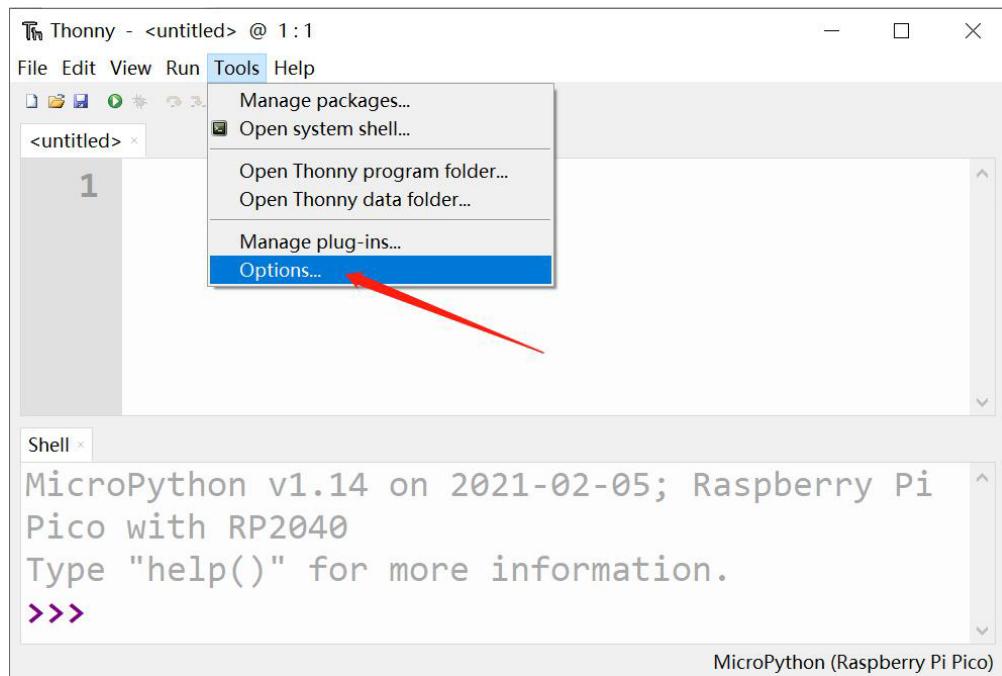
Double-cliquez sur le raccourci suivant sur le bureau du PC pour ouvrir le logiciel, sélectionnez la langue appropriée et entrez dans l'interface du logiciel:

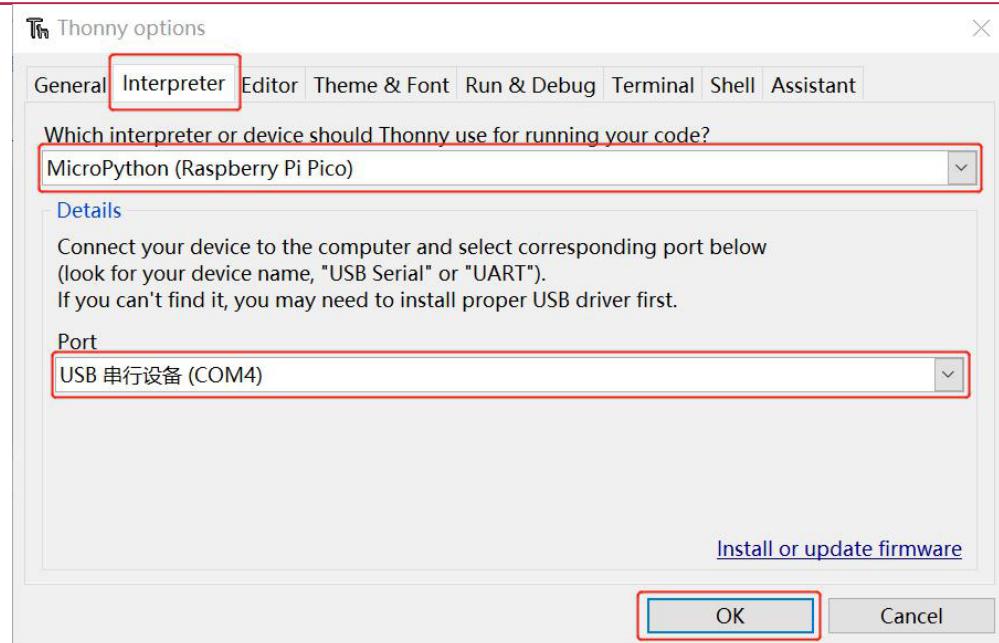


The screenshot shows the Thonny IDE interface. At the top, the title bar reads "Thonny - <untitled> @ 1:1". The menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations. A code editor window titled "<untitled>" contains the number "1". To the right of the code editor is a "Shell" window displaying MicroPython version information: "MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040" and the instruction "Type 'help()' for more information.". A red arrow points to the "MicroPython (Raspberry Pi Pico)" tab at the bottom of the shell window.

3.1 Sélectionnez la bonne programmation Raspberry Pi Pico

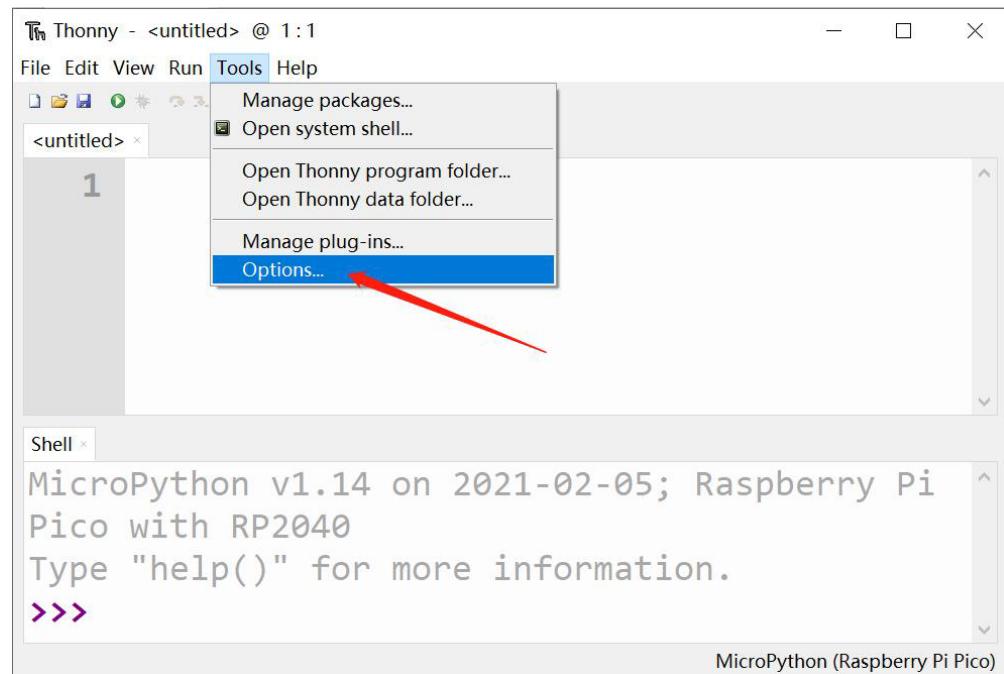
Sélectionnez Tools/Settings dans la barre de menu et cliquez sur la fenêtre contextuelle des paramètres:

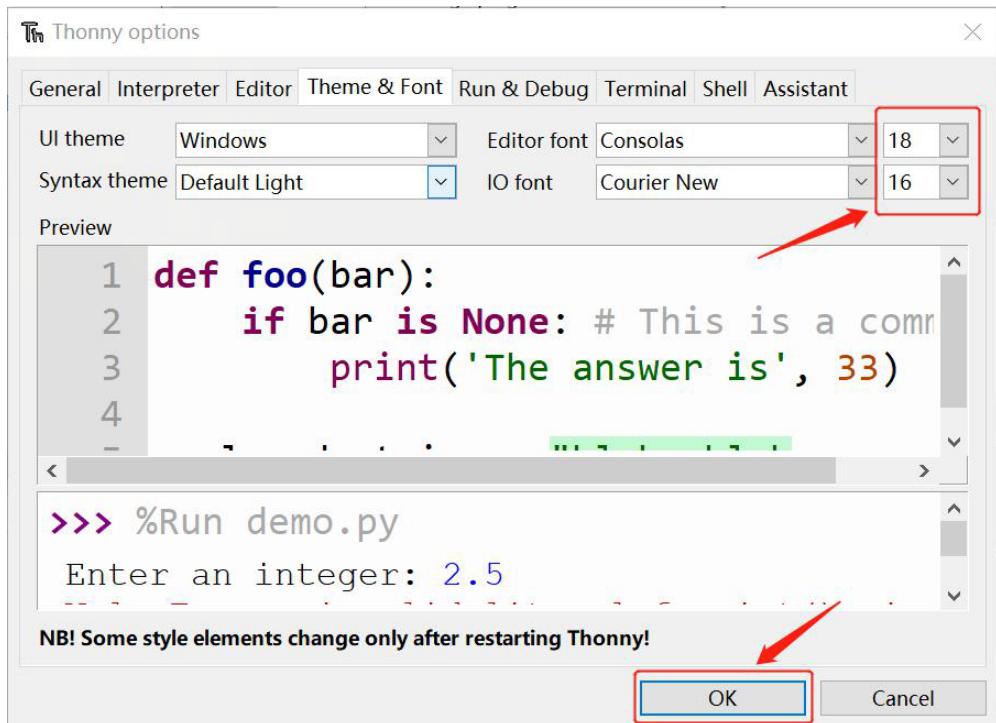




3.2 Modifier la taille de la police

Sélectionnez Tools/Settings/Subjects et adresses



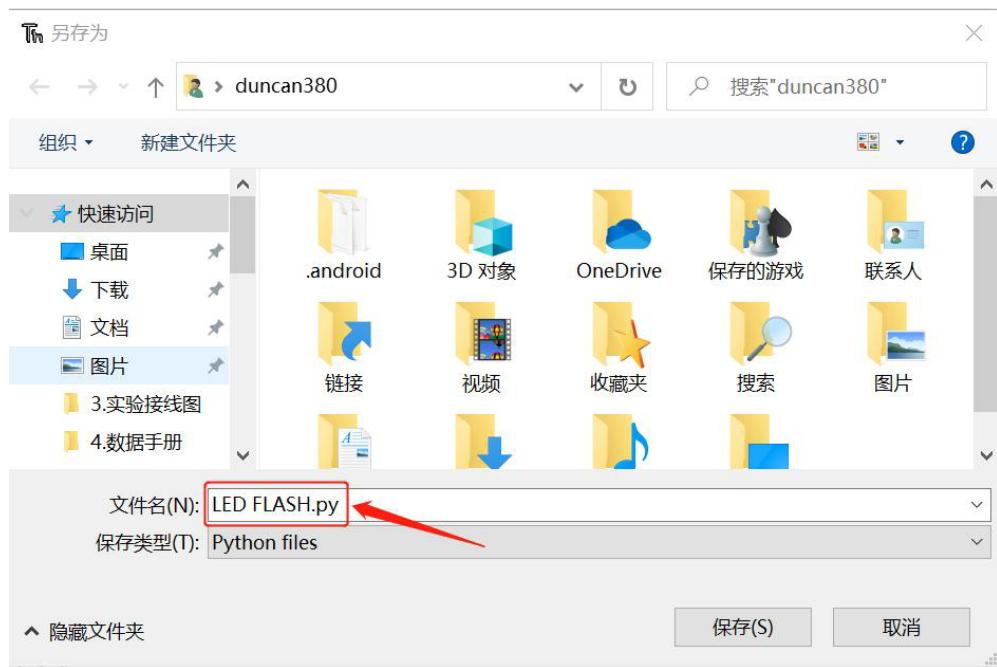
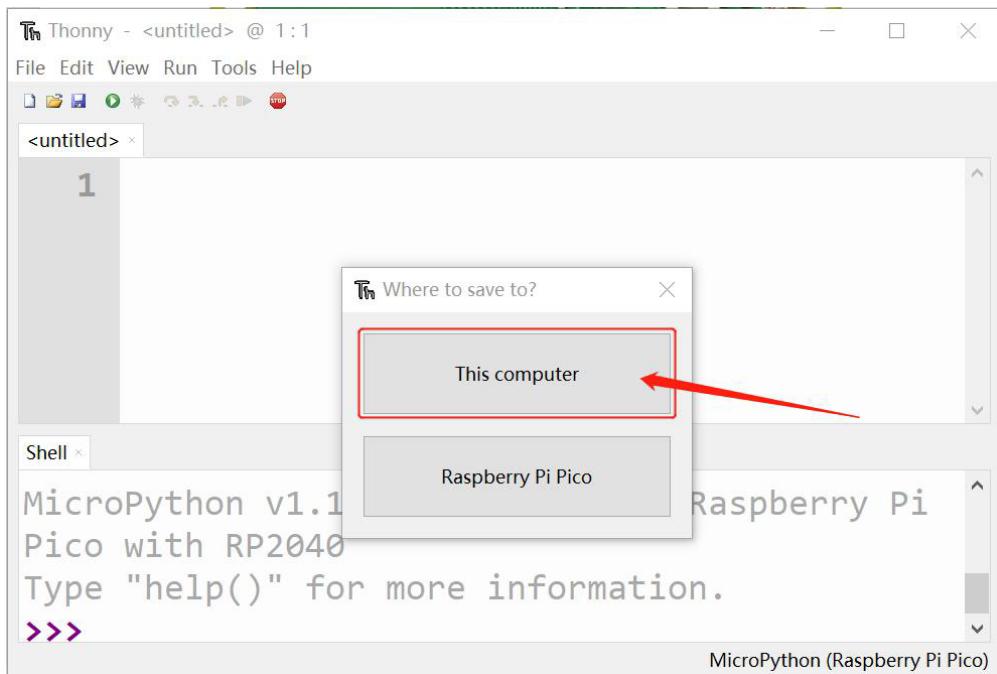


3.3 Écrire le programme de clignotement des voyants à l'aide de Thonny

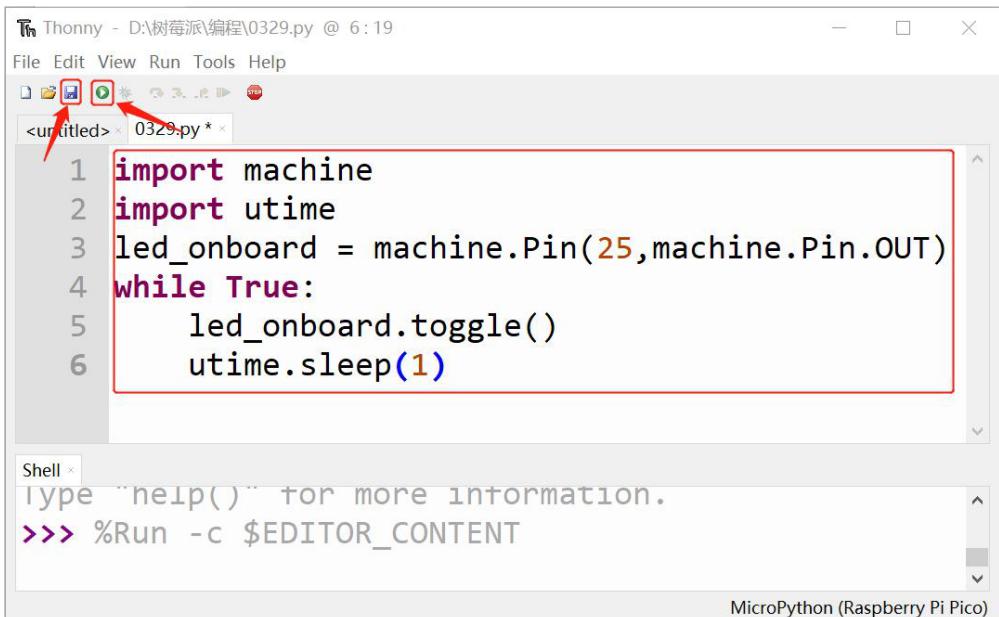
Pour allumer la lumière LED, vous devez donner un niveau élevé, éteindre pour donner un niveau bas, et maintenant écrire un logiciel Thonny qui permet au voyant LED (GP25) de la carte Pico de clignoter, comme illustré ci-dessous:



Étape 1: Ouvrez le logiciel Thonny et créez un nouveau fichier nommé « **.py »



Étape 2: Après avoir écrit le code suivant, cliquez sur le bouton ENREGISTRER:



```
import machine
import utime
led_onboard = machine.Pin(25,machine.Pin.OUT)
while True:
    led_onboard.toggle()
    utime.sleep(1)
```

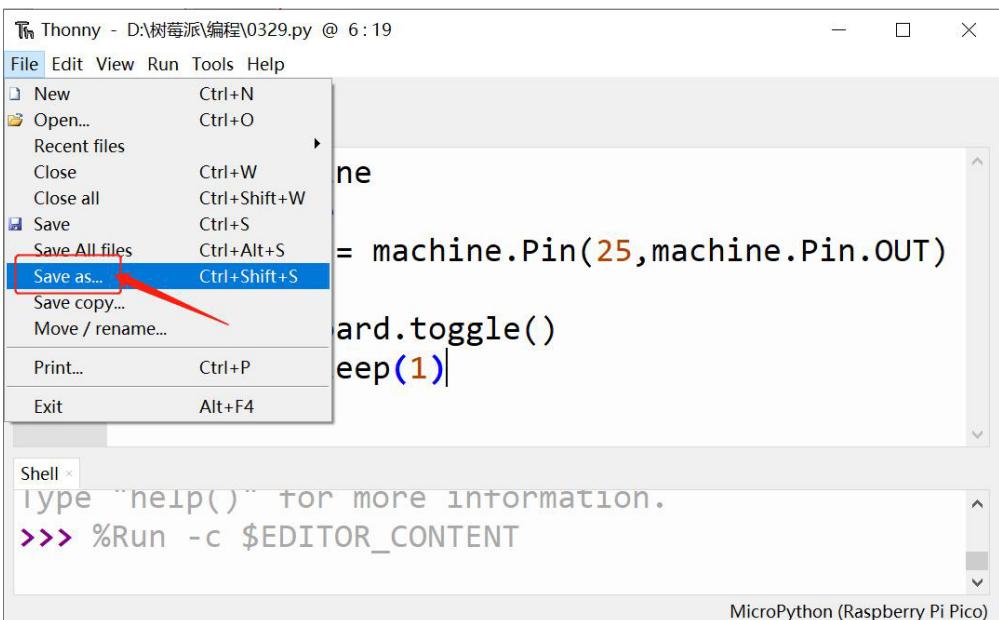
Shell >
Type `help()` for more information.
`>>> %Run -c $EDITOR_CONTENT`

MicroPython (Raspberry Pi Pico)

Étape 3: Cliquez sur le bouton RUN pour exécuter le programme, et le voyant LED clignotera en fonction du programme.

Note: Le programme enregistré selon la méthode ci-dessus est sur l'ordinateur, et si vous débranchez l'USB à ce moment-là, la LED ne clignotera pas. Si vous souhaitez réparer le programme sur la carte mère Pico, vous pouvez exécuter automatiquement le programme tant que vous l'allumez sur l'appareil. Voici comment:

Tout ce que vous avez à faire est de sélectionner Enregistrer sous et enregistrer sous m dans le menu du fichier [main.py]



Thonny - D:\树莓派\编程\0329.py @ 6:19

File Edit View Run Tools Help

untitled > 0329.py *

```
1 import machine
2 import utime
3 led_onboard = machine.Pin(2, machine.Pin.OUT)
4 while True:
5     led_onboard.toggle()
6     utime.sleep(0.5)
```

Where to save to?

This computer

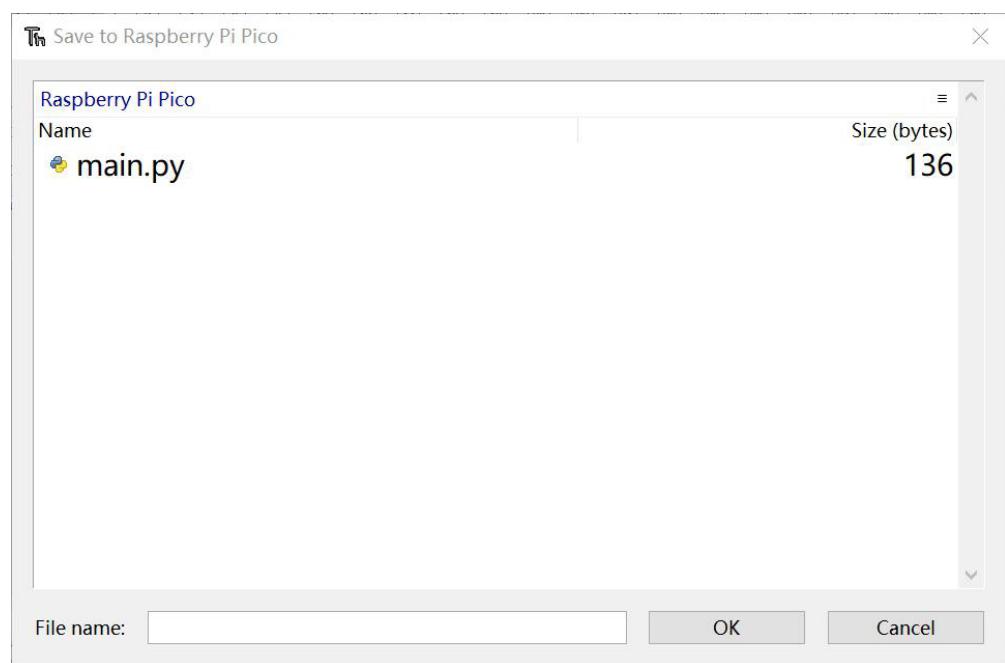
Raspberry Pi Pico

Shell >

Type "help()" for more information.

>>> %Run -c \$EDITOR_CONTENT

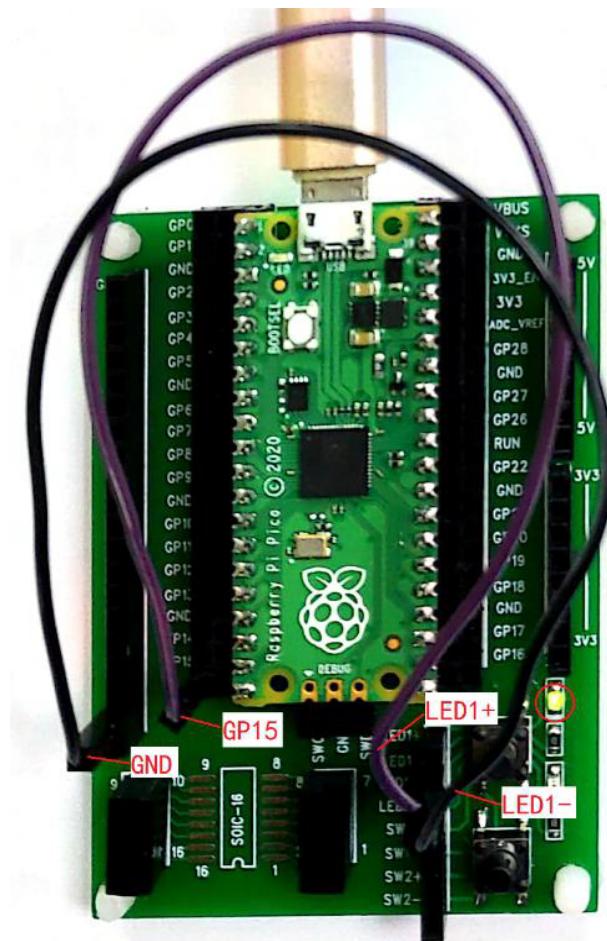
MicroPython (Raspberry Pi Pico)



4. Programmation de cas

4.1 Expériences de diodes électroluminescentes

Le câblage est le suivant:



Le code du programme est le suivant:

```
1 import machine
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 while True:
6     led_external.toggle()
7     utime.sleep(1)
```

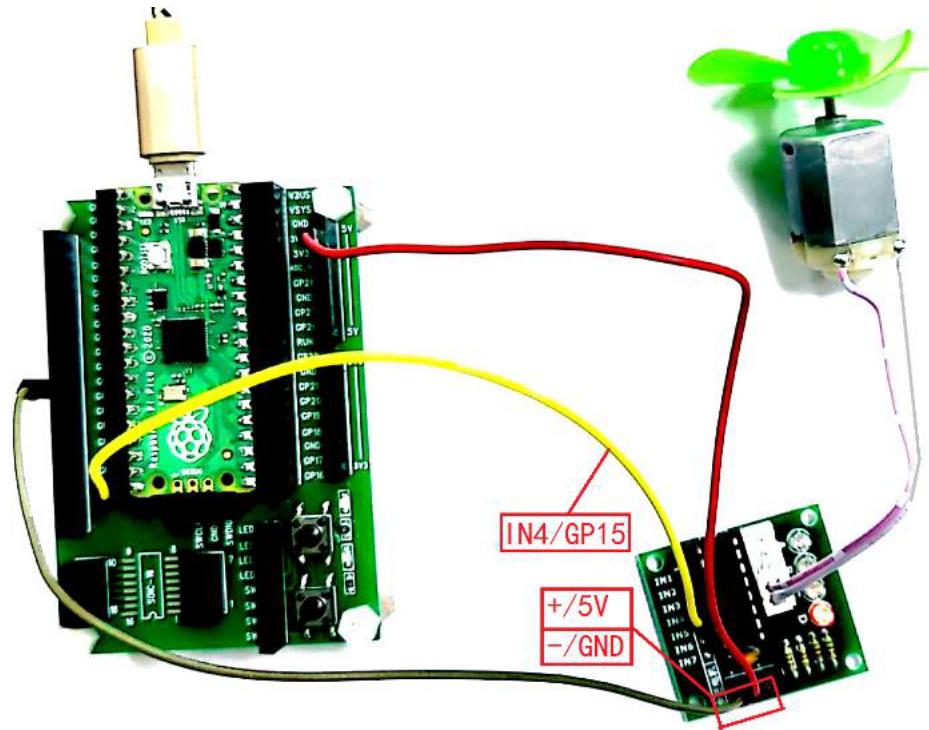
Shell <

```
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico w
ith RP2040
Type "help()" for more information.
>>>
```

MicroPython (Raspberry Pi Pico)

4.2 Expérience de ventilateur de moteur à courant continu

Le câblage est le suivant:



Le code du programme est le suivant:

```
Thonny - E:\Raspberry_pico\入门资料\2.案例程序\8.直流电机风扇实验\main.py @ 14 : 21
```

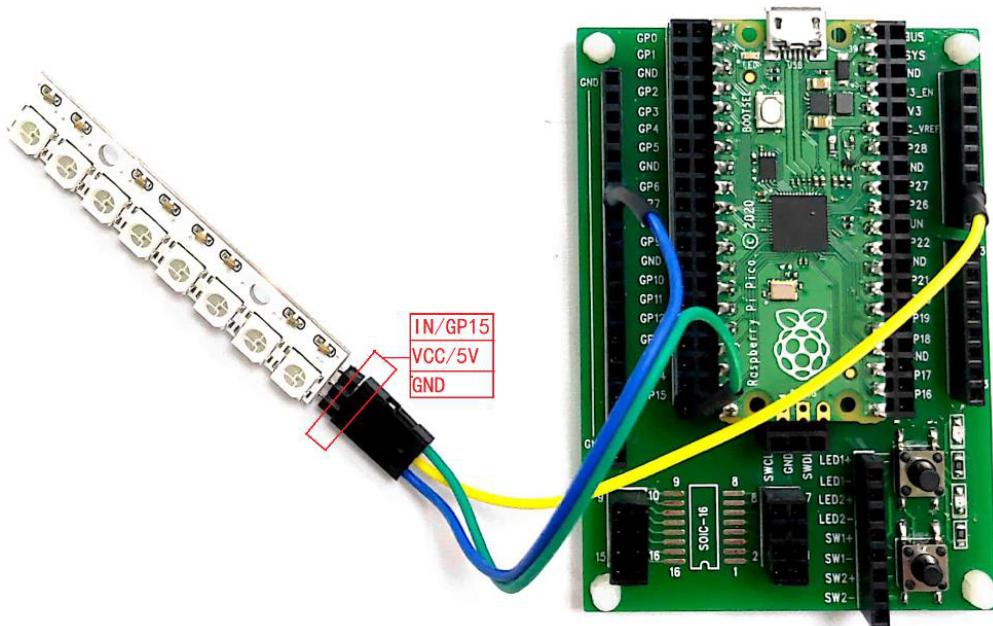
The screenshot shows the Thonny IDE interface with the following details:

- Title Bar:** Thonny - E:\Raspberry_pico\入门资料\2.案例程序\8.直流电机风扇实验\main.py @ 14 : 21
- Menu Bar:** File Edit View Run Tools Help
- Tool Buttons:** New, Open, Save, Run, Stop, Run Selection, Stop Selection, Run All, Stop All.
- Code Editor:** The file main.py contains the following Python code:

```
1 from machine import Pin, PWM
2 from time import sleep
3
4 pwm = PWM(Pin(15))
5
6 pwm.freq(1000)
7
8 while True:
9     for duty in range(10,65025,100):
10         pwm.duty_u16(duty)
11         sleep(0.001)
12     for duty in range(65025, 10, -100):
13         pwm.duty_u16(duty)
14         sleep(0.001)
```
- Variables View:** Shows variables defined in the script: PWM, Pin, duty, machine, pwm, rp2, sleep.
- Shell View:** Displays the command "KeyboardInterrupt:" followed by a prompt ">>>".
- Status Bar:** MicroPython (Raspberry Pi Pico)

4.3 Bande LED

Le câblage est le suivant:



Le code du programme est le suivant:

The screenshot shows the Thonny IDE interface with the file "rainbow.py" open. The code defines a function to initialize the LED strip and sets up color variables for various hues. The shell window shows the MicroPython interpreter running on the Pico, indicating the program has been successfully executed.

```
1 import time
2 import ws2812b
3
4 numpix = 8
5 pin = 15
6 strip = ws2812b.ws2812b(numpix, 0, pin)
7
8 RED = (255, 0, 0)
9 ORANGE = (255, 165, 0)
10 YELLOW = (255, 150, 0)
11 GREEN = (0, 255, 0)
12 BLUE = (0, 0, 255)
13 INDIGO = (75, 0, 130)
14 VIOLET = (138, 43, 226)
15 COLORS = (RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET)      #define LED color change order
16
```

Shell

```
MicroPython v1.13-290-g556ae7914 on 2021-01-21; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
```

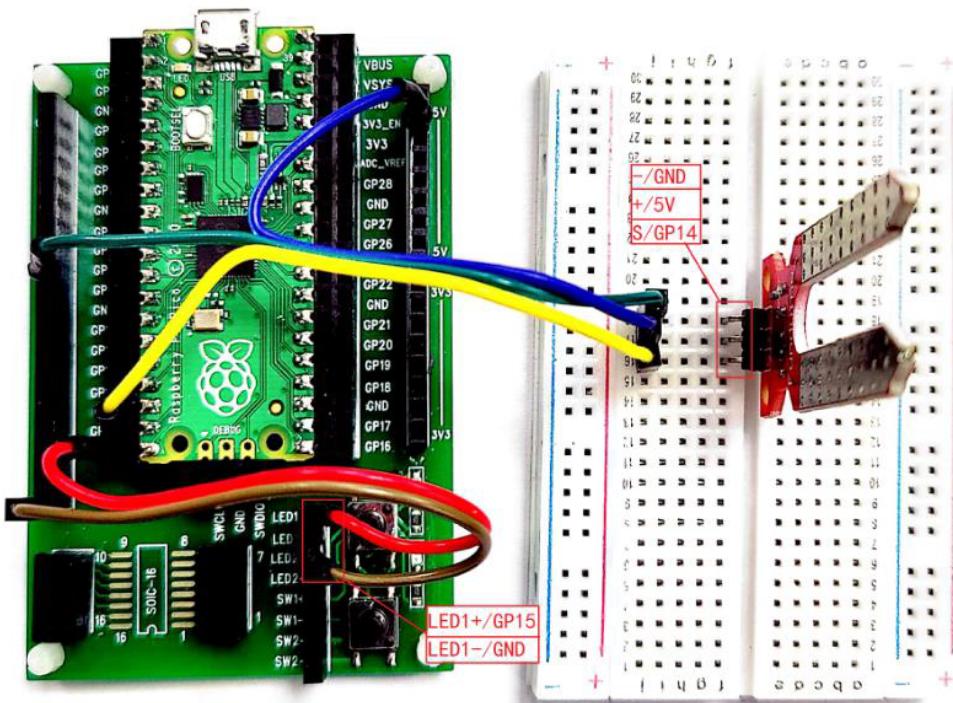
MicroPython (Raspberry Pi Pico)

The screenshot shows the Thonny IDE interface with the following details:

- Title Bar:** Thonny - Raspberry Pi Pico :: /ws2812b.py @ 14:49
- Menu Bar:** File Edit View Run Tools Help
- Toolbar:** Standard file operations (New, Open, Save, etc.)
- Code Editor:** The file "rainbow.py" is open, containing Python code for a WS2812b LED strip. The code uses the rp2 library to define a bitstream for a 3x3 matrix of LEDs. It includes labels for the bitloop and zero states, and a jmp("bitloop") instruction.
- Shell:** MicroPython v1.13-290-g556ae7914 on 2021-01-21; Raspberry Pi Pico with RP2040. It shows the command %%Run -c \$EDITOR_CONTENT and a message indicating the backend terminated or disconnected.
- Status Bar:** MicroPython (Raspberry Pi Pico)

4.4 Capteur d'humidité

Le câblage est le suivant:



Le code du programme est le suivant:

The screenshot shows the Thonny IDE interface with the file 'main.py' open. The code is as follows:

```
1 import utime
2
3 led_external = machine.Pin(15, machine.Pin.OUT)
4 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
5
6 flag = 0
7
8 while True:
9     if aout.value() == 1:
10         led_external.value(1)
11         if flag:
12             print('led on')
13             flag = 0
14         utime.sleep(0.2)
15     else:
16         led_external.value(0)
17         if flag == 0:
18             print('led off')
19             flag = 1
```

The code uses a PULL_UP configuration for pin 14 (AOUT). It prints 'led on' when the AOUT pin is high and 'led off' when it is low. A red box highlights the main loop, and two red arrows point to the print statements.

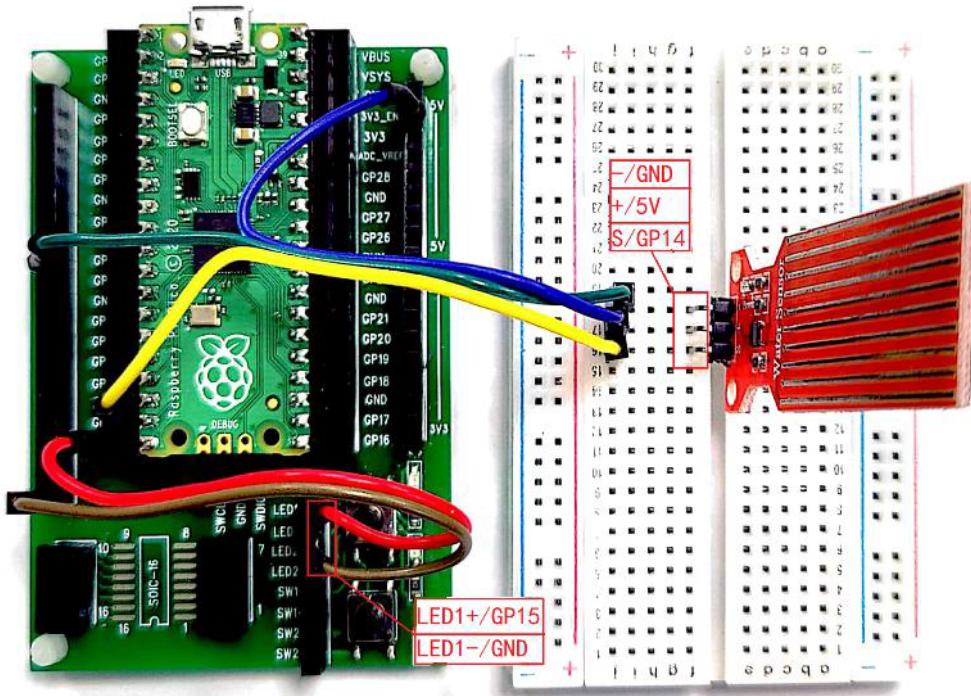
At the bottom, the shell shows:

```
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
```

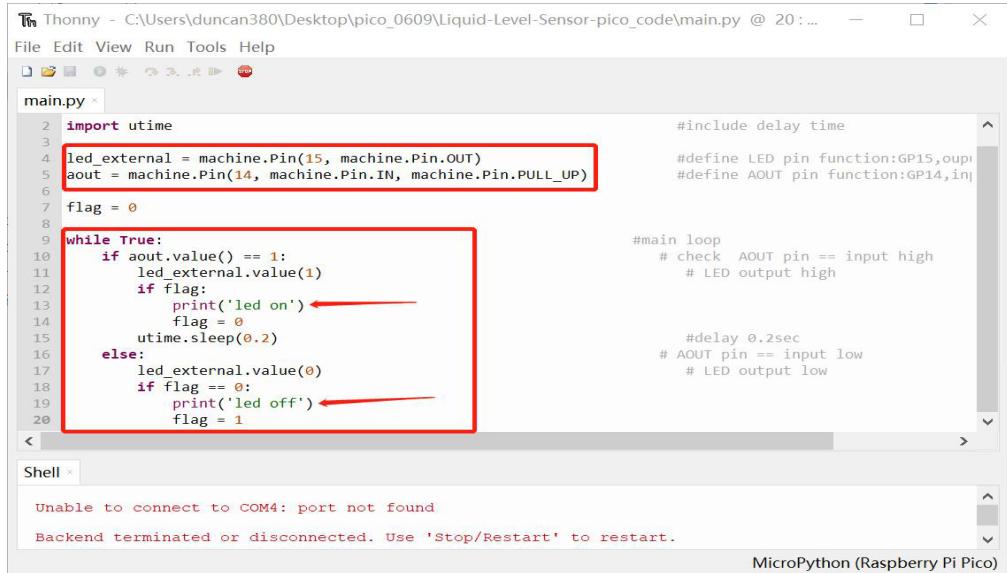
MicroPython (Raspberry Pi Pico)

4.5 Capteur de niveau de liquide

Le câblage est le suivant:



Le code du programme est le suivant:



```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Liquid-Level-Sensor-pico_code\main.py @ 20 : ...
```

```
File Edit View Run Tools Help
```

```
main.py
```

```
1 import utime
2
3 led_external = machine.Pin(15, machine.Pin.OUT)
4 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
5
6 flag = 0
7
8 while True:
9     if aout.value() == 1:
10         led_external.value(1)
11         if flag:
12             print('led on') ←
13             flag = 0
14             utime.sleep(0.2)
15     else:
16         led_external.value(0)
17         if flag == 0:
18             print('led off') ←
19             flag = 1
20
```

```
#include delay time
#define LED pin function:GP15,output
#define AOUT pin function:GP14,input and pull up

#main loop
# check AOUT pin == input high
# LED output high
#delay 0.2sec

#AOUT pin == input low
# LED output low
```

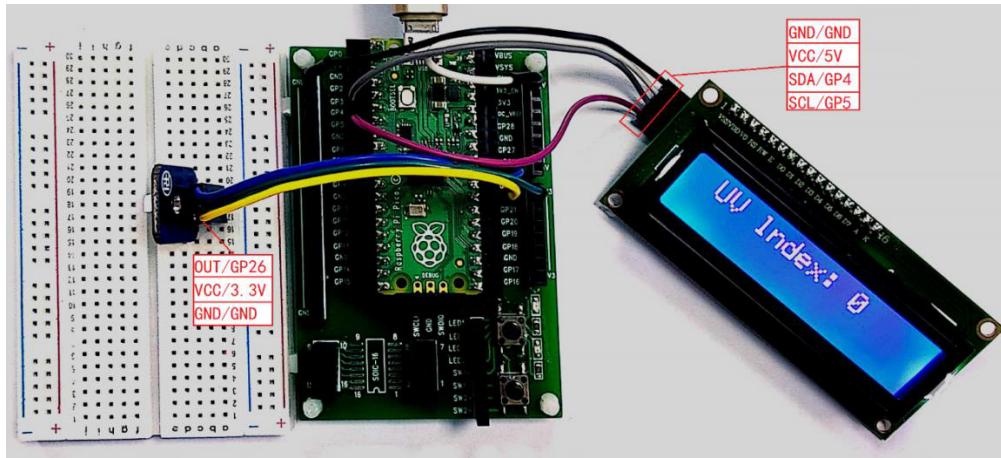
```
Shell
```

```
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
```

```
MicroPython (Raspberry Pi Pico)
```

4.6 Capteur UV

Le câblage est le suivant:



Le code du programme est le suivant:

Opération "main.py" Enregistrez d'abord le programme "lcd1602_i2c.py" sur raspberry pi Pico, la méthode de fonctionnement se réfère à 3.3/step3.



```
Thonny - D:\树莓派\Moisture-Sensor-pico_code\main.py @ 14 : ...
```

```
File Edit View Run Tools Help
```

```
main.py
```

```
1 import machine
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7
8 while True:
9     if aout.value() == 1:
10         led_external.value(1)
11         utime.sleep(0.2)
12     else:
13         led_external.value(0)
```

```
#include hardware devices
#include delay time

#define LED pin function:GP15,output function
#define AOUT pin function:GP14,input and pull up

#main loop
# check AOUT pin == input high
# LED output high
#delay 0.2sec

# AOUT pin == input low
# LED output low
```

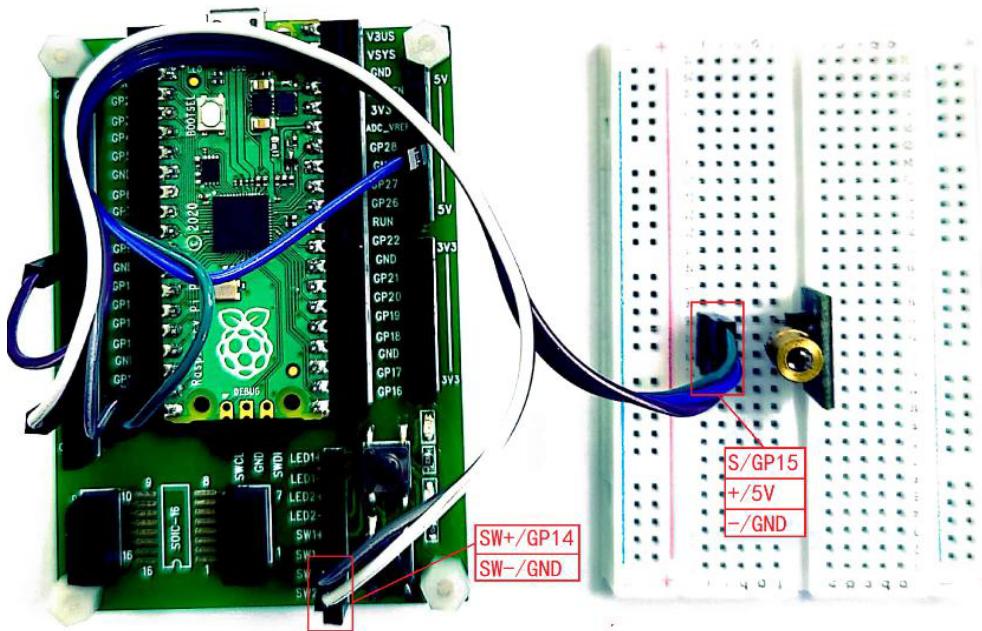
```
Shell
```

```
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
```

```
MicroPython (Raspberry Pi Pico)
```

4.7 Capteur de sortie laser

Le câblage est le suivant:



Le code du programme est le suivant:

The screenshot shows the Thonny IDE interface with the following details:

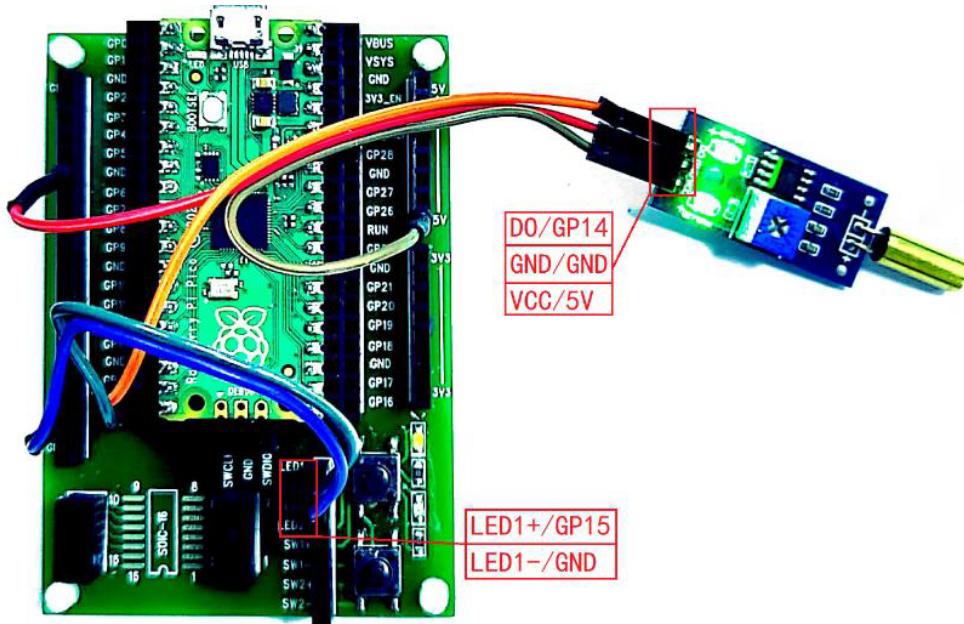
- Title Bar:** Thonny - D:\树莓派\Raspberry-Pi-Pico-Sensor\万科盛0527\Laser_out_pico_code\main.py @ 14 : 5
- File Menu:** File Edit View Run Tools Help
- Code Editor:** The `main.py` file contains the following Python code:

```
1 import machine
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 key = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7
8 while True:
9     if key.value() == 0:
10         led_external.value(1)
11         utime.sleep(0.2)
12     else:
13         led_external.value(0)
14
```
- MicroPython Shell:** The shell shows the following output:

```
Shell
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
Connection lost (EOF)
Use Stop/Restart to reconnect.
```
- Status Bar:** MicroPython (Raspberry Pi Pico)

4.8 Capteur d'inclinaison

Le câblage est le suivant:



Le code du programme est le suivant:

```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Tilt_sensor_pico_code\main.py @ 20 : 21
File Edit View Run Tools Help
main.py
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 0
8
9 while True:
10     if aout.value() == 1:
11         led_external.value(1)
12         if flag:
13             print('led on')
14             flag = 0
15             utime.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off')
20             flag = 1
#main loop
# check AOUT pin == input high
# LED output high
#delay 0.2sec
# AOUT pin == input low
# LED output low
#include delay time
#define LED pin function:GP15,out
#define AOUT pin function:GP14,in
```

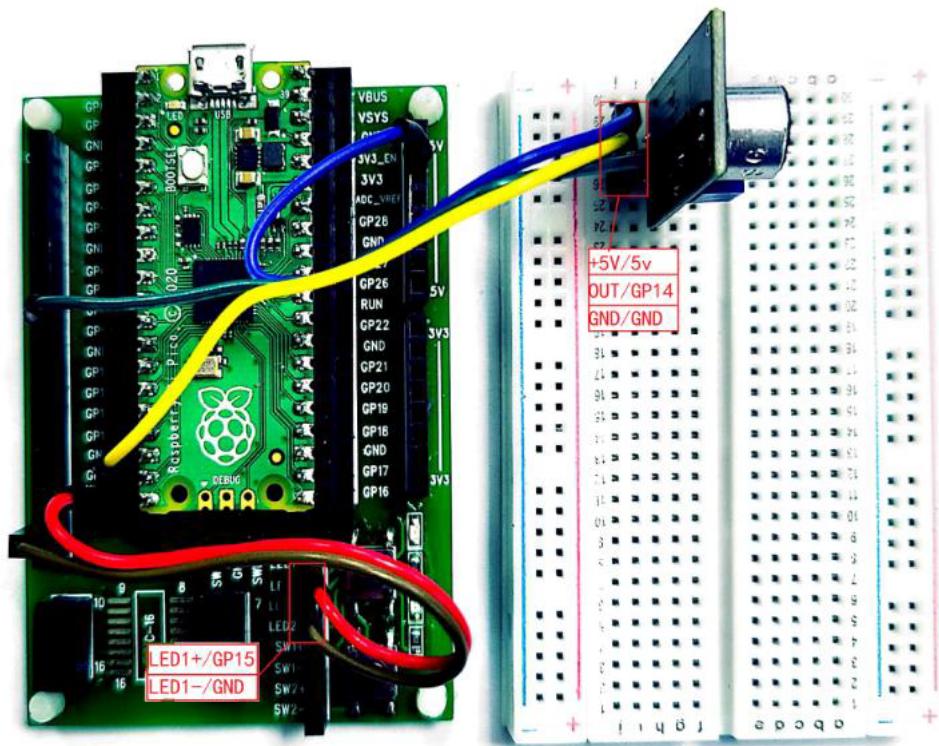
Shell

```
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
```

MicroPython (Raspberry Pi Pico)

4.9 Capteur de son

Le câblage est le suivant:



Le code du programme est le suivant:

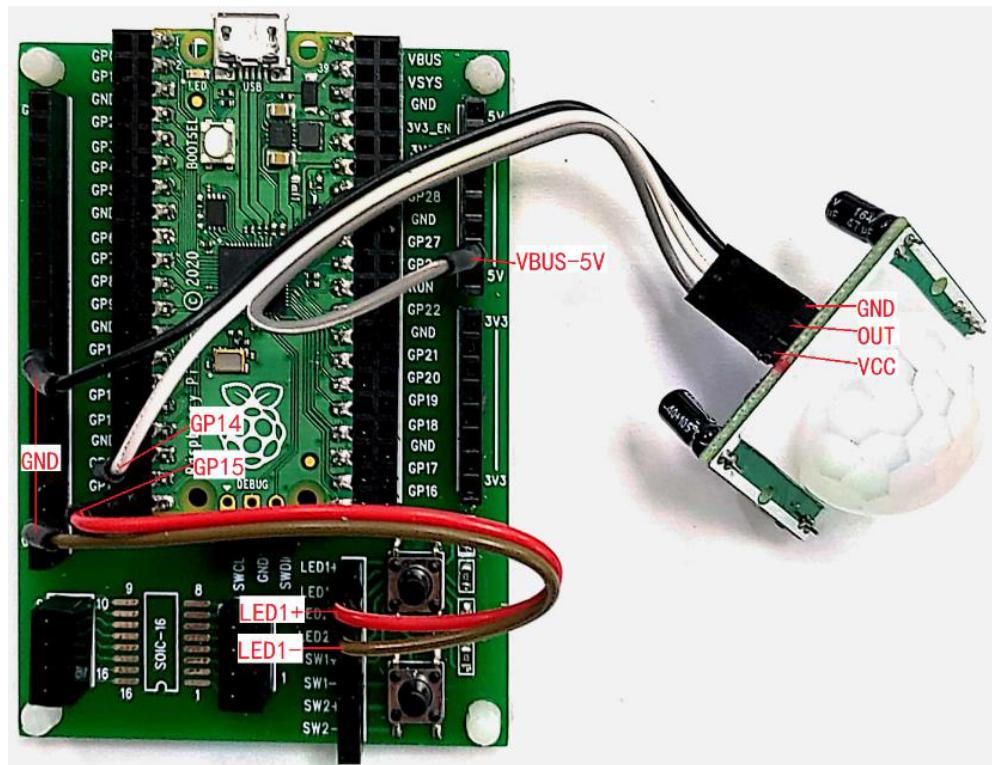
```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Sound-Sensor-pico_code\main.py @ 20:21
File Edit View Run Tools Help
main.py x
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 0
8
9 while True:
10     if aout.value() == 0:
11         led_external.value(1)
12         if flag:
13             print('led on') ←
14             flag = 0
15             utime.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off') ←
20             flag = 1
#main loop
# check AOUT pin == input high
# LED output high

#delay 0.2sec
# AOUT pin == input low
# LED output low

Shell x
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

4.10 Capteur PIR

Le câblage est le suivant:



Le code du programme est le suivant:

```
Thonny - C:\Users\duncan380\Desktop\pico_0609\PIR-pico_code\main.py @ 20:21
File Edit View Run Tools Help
main.py
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 1
8
9 while True:
10     if aout.value() == 1:
11         led_external.value(1)
12         if flag:
13             print('led on')
14             flag = 0
15             utime.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off')
20             flag = 1
# include delay time
#define LED pin function:GP15,out
#define AOUT pin function:GP14,in

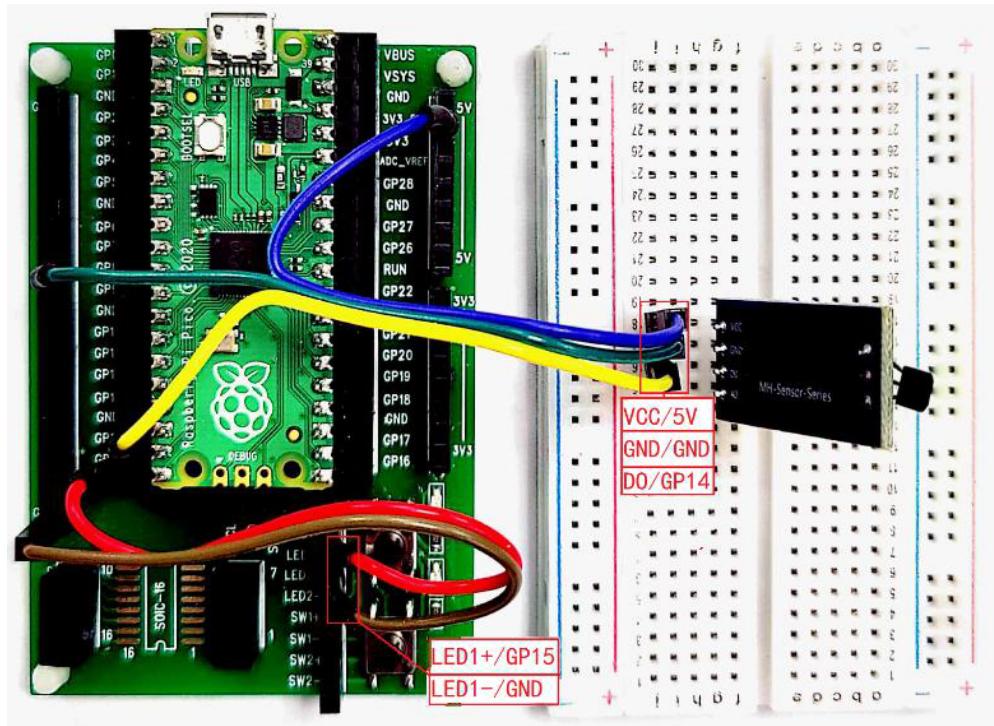
#main loop
# check AOUT pin == input high
# LED output high

#delay 0.2sec
# AOUT pin == input low
# LED output low

Shell
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

4.11 Capteur à effet Hall

Le câblage est le suivant:



Le code du programme est le suivant:

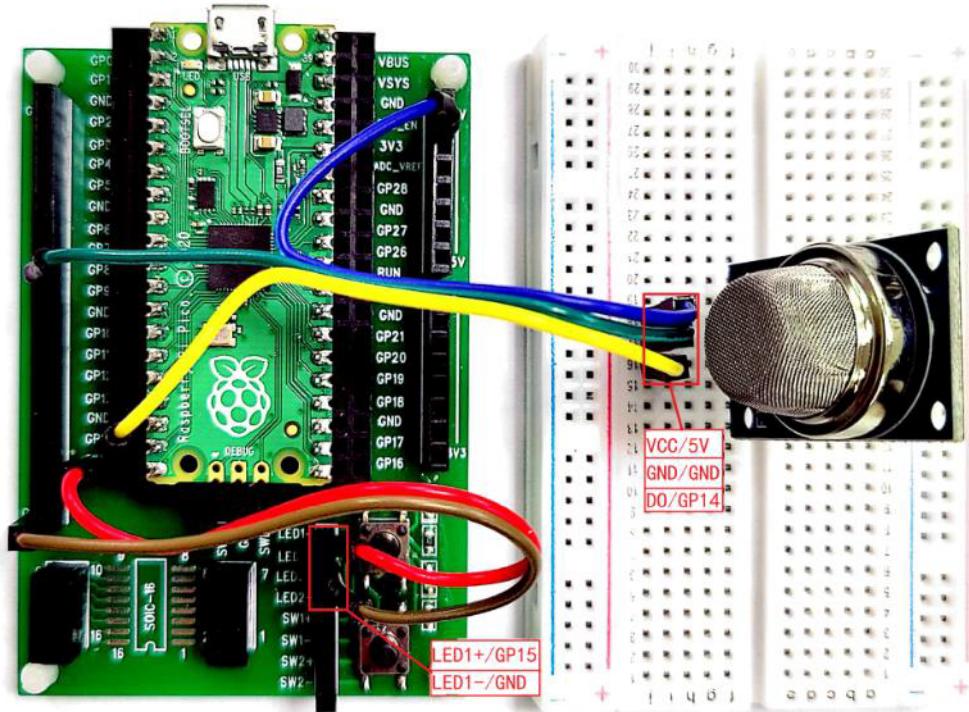
```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Hall-Sensor-pico-code\main.py @ 20:21
File Edit View Run Tools Help
main.py x
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 0
8
9 while True:
10     if aout.value() == 1:
11         led_external.value(1)
12         if flag:
13             print('led on') ←
14             flag = 0
15         utime.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off') ←
20             flag = 1
# include delay time
#define LED pin function:GP15,out
#define AOUT pin function:GP14,in
#main loop
# check AOUT pin == input high
# LED output high

#delay 0.2sec
# AOUT pin == input low
# LED output low

Shell x
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

4.12 Capteur de gaz

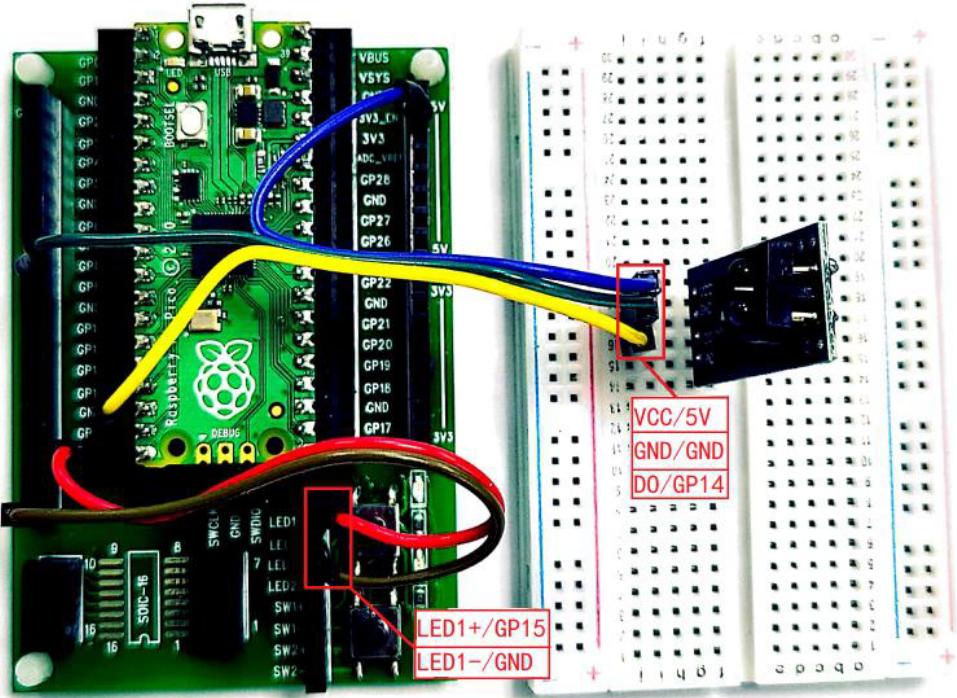
Le câblage est le suivant:



Le code du programme est le suivant:

4.13 Capteur infrarouge réfléchissant

Le câblage est le suivant:

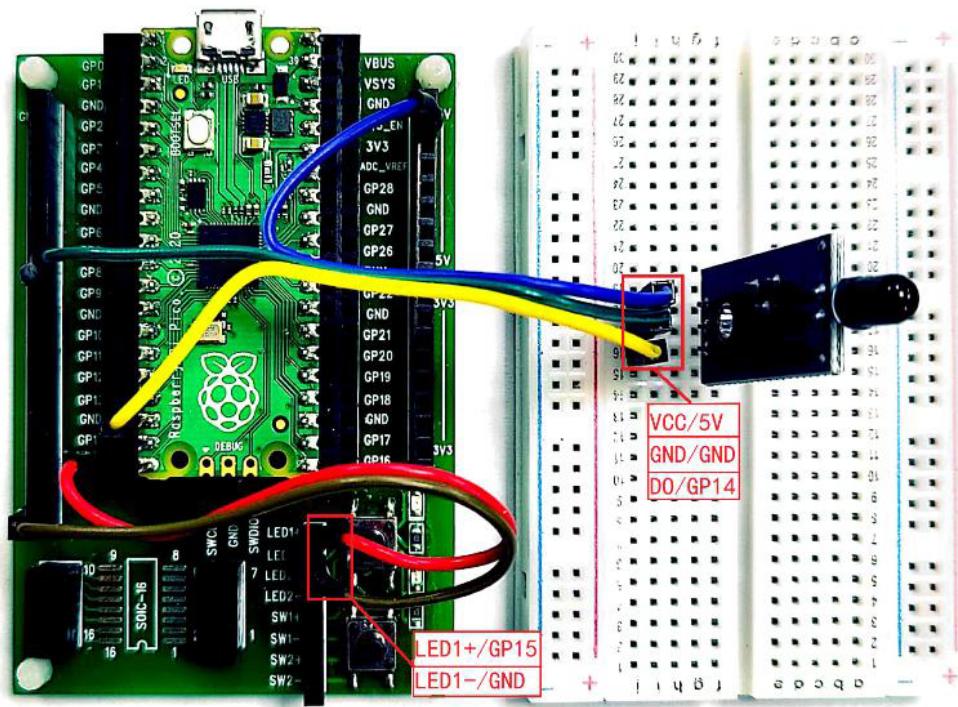


Le code du programme est le suivant:

```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Infrared_Reflective_sensor_pico_code\main.py @... — □ ×  
File Edit View Run Tools Help  
main.py x  
2 import utime  
3  
4 led_external = machine.Pin(15, machine.Pin.OUT)  
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)  
6  
7 flag = 0  
8  
9 while True:  
10     if aout.value() == 1:  
11         led_external.value(1)  
12         if flag:  
13             print('led on') ←  
14             flag = 0  
15         utime.sleep(0.2)  
16     else:  
17         led_external.value(0)  
18         if flag == 0:  
19             print('led off') ←  
20             flag = 1  
#include delay time  
#define LED pin function:GP15,out  
#define AOUT pin function:GP14,in  
  
#main loop  
# check AOUT pin == input high  
# LED output high  
  
#delay 0.2sec  
# AOUT pin == input low  
# LED output low  
  
Shell x  
Unable to connect to COM4: port not found  
Backend terminated or disconnected. Use 'Stop/Restart' to restart.  
MicroPython (Raspberry Pi Pico)
```

4.14 Capteur de flamme

Le câblage est le suivant:



Le code du programme est le suivant:

```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Flame-Sensor-pico-code\main.py @ 9:1
File Edit View Run Tools Help
main.py
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 0
8
9 while True:
10     if aout.value() == 1:
11         led_external.value(1)
12         if flag:
13             print('led on')
14             flag = 0
15         utime.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off')

#define LED pin function:GP15,out
#define AOUT pin function:GP14,in

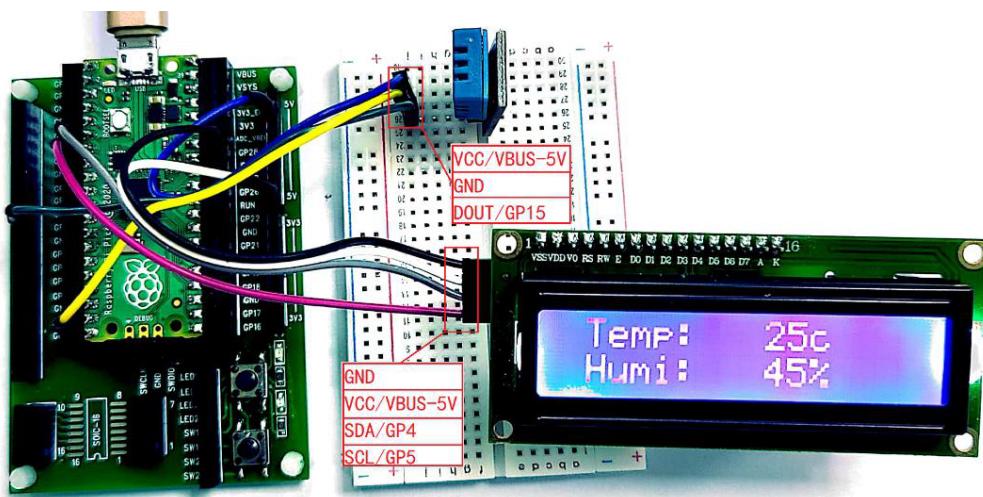
#main loop
# check AOUT pin == input high
# LED output high

#delay 0.2sec
# AOUT pin == input low
# LED output low

Shell
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

4.15 Capteur de température-humidité

Le câblage est le suivant:



Le code du programme est le suivant:

Opération "main.py" Sauvegardez d'abord le programme "dht11.py" et "lcd1602_i2c.py" sur raspberry pi Pico, la méthode de fonctionnement se réfère à 3.3/step3.

Note: Le nom du fichier doit être cohérent avec le nom du programme, les lettres majuscules et les symboles doivent être cohérents.

The screenshot shows the Thonny IDE interface. The menu bar includes File, Edit, View, Run, Tools, Help. The toolbar has icons for file operations like Open, Save, Run, and Stop. There are two tabs open: 'main.py' and '[lcd1602_i2c.py]'. The code editor displays the following Python script:

```
from time import sleep_ms
# The PCF8574 has a jumper selectable address: 0x20 - 0x27
DEFAULT_I2C_ADDR = 0x27

# Defines shifts or masks for the various LCD line attached to the PCF8574
MASK_RS = 0x01
MASK_RW = 0x02
MASK_E = 0x04
SHIFT_BACKLIGHT = 3
SHIFT_DATA = 4

class LcdApi:
```

The code for the LCD API is highlighted with a red box. The shell window at the bottom shows repeated sensor readings:

```
temperature is 28 -wet is 78 %
Sensor is working
temperature is 28 -wet is 78 %
Sensor is working
temperature is 28 -wet is 78 %
Sensor is working
temperature is 28 -wet is 78 %
```

The screenshot shows the Thonny IDE interface for a Raspberry Pi Pico project. The menu bar includes File, Edit, View, Run, Tools, and Help. The title bar indicates the file is dht11.py and the current time is 65:48. The toolbar contains icons for file operations like Open, Save, and Run. The code editor has tabs for main.py and dht11.py, with dht11.py currently selected. The code for dht11.py defines a class DHT11 with methods for initialization and reading data. The shell window at the bottom displays repeated sensor readings: temperature is 28 - wet is 83 %, indicating the sensor is working.

```
#include hardware device
#include hardware device
#include delay time

1 #import pyb
2 from machine import UART
3 from machine import Pin
4 from time import sleep_ms,sleep_us
5
6
7
8 class DHT11:
9     def __init__(self,pin_name):
10         sleep_ms(1000)
11         self.N1 = Pin(pin_name, Pin.OUT)
12         self.PinName=pin_name
13         sleep_ms(10)
14     def read_data(self):
15         self.__init__(self.PinName)

Shell *
temperature is 28 - wet is 83 %
Sensor is working
temperature is 28 - wet is 83 %
Sensor is working
temperature is 28 - wet is 83 %
Sensor is working
temperature is 28 - wet is 83 %
```

The screenshot shows the Thonny IDE interface with the following details:

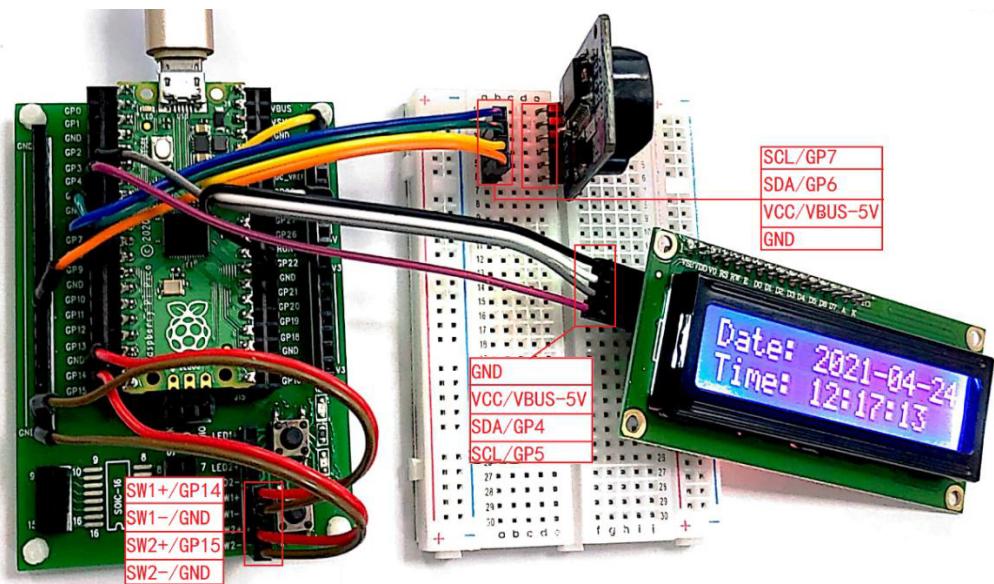
- Title Bar:** Thonny - D\树莓派\Temperature-Humidity-Sensor-Pico_code\main.py @ 35:1
- Menu Bar:** File Edit View Run Tools Help
- Toolbar:** Includes icons for file operations like Open, Save, Run, and Stop.
- Code Editor:** Displays the Python script `main.py`. The code imports necessary modules (`time`, `machine`, `lcd1602_i2c`, `dht11`) and defines a function `readDHTdata()` to read temperature and humidity from a DHT11 sensor connected via I2C. It also includes comments defining the I2C address as `0x27` and the DHT11 pin as `GP15`.
- Shell:** Shows the output of the script execution, displaying repeated readings of temperature and humidity: "temperature is 27 -wet is 81 %", "Sensor is working", and "temperature is 28 -wet is 82 %".
- Status Bar:** MicroPython (Raspberry Pi Pico)

4.16 Capteur d'horloge

Opération "main.py" Sauvegardez d'abord le programme "DS3231.py" et "lcd1602_i2c.py" sur raspberry pi Pico, la méthode de fonctionnement se réfère à 3.3/step3.

Note: Le nom du fichier doit être cohérent avec le nom du programme, les lettres majuscules et les symboles doivent être cohérents.

Le câblage est le suivant:



Le code du programme est le suivant:

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. The main window has tabs for [ds3231.py]*, lcd1602_i2c.py, and main.py. The code in main.py is as follows:

```
1 import machine
2 from machine import I2C,Pin
3
4 DS3231_ADDR = 0x68 #const(0x68)
5 DS3231_REG_SEC = b'\x00'
6 DS3231_REG_MIN = b'\x01'
7 DS3231_REG_HOUR = b'\x02'
8 DS3231_REG_WEEKDAY= b'\x03'
9 DS3231_REG_DAY = b'\x04'
10 DS3231_REG_MONTH = b'\x05'
11 DS3231_REG_YEAR = b'\x06'
12 DS3231_REG_A1SEC = b'\x07'
13 DS3231_REG_A1MIN = b'\x08'
14 DS3231_REG_A1HOUR = b'\x09'
15 DS3231_REG_A1DAY = 0x0A
```

The Shell tab shows the output of the code execution:

```
File "<stdin>", line 22, in <module>
File "lcd1602_i2c.py", line 132, in move_to
File "lcd1602_i2c.py", line 258, in hal_write_command
OSError: 5
>>>
```

The status bar at the bottom right indicates "MicroPython (Raspberry Pi Pico)".

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, Help. The main window has tabs for [ds3231.py]*, lcd1602_i2c.py, and main.py. The code in main.py is as follows:

```
1 from time import sleep_ms
2
3 # The PCF8574 has a jumper selectable address: 0x20 - 0x27
4 DEFAULT_I2C_ADDR = 0x27#0x27#
5
6 # Defines shifts or masks for the various LCD line attached to the PCF8574
7
8 MASK_RS = 0x01
9 MASK_RW = 0x02
10 MASK_E = 0x04
11 SHIFT_BACKLIGHT = 3
12 SHIFT_DATA = 4
13
14
15 class LcdApi:
```

The Shell tab shows the output of the code execution:

```
File "<stdin>", line 22, in <module>
File "lcd1602_i2c.py", line 132, in move_to
File "lcd1602_i2c.py", line 258, in hal_write_command
OSError: 5
>>>
```

The status bar at the bottom right indicates "MicroPython (Raspberry Pi Pico)".

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, Help. The main window has tabs for [ds3231.py]*, lcd1602_i2c.py, and main.py. The code in main.py is as follows:

```
1 from machine import I2C, Pin
2 import time
3 from ds3231 import DS3231
4 from lcd1602_i2c import I2cLcd
5
6 DEFAULT_I2C_ADDR = 0x27
7
8
9 if __name__ == "__main__":
10     i2c = I2C(0,scl=Pin(5), sda=Pin(4), freq=100000)
11     lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
12     lcd.clear()
13     ds=DS3231()
14     ds.DATE([21,4,19])
15     ds.TIME([14,26,00])
```

The Shell tab shows the output of the code execution:

```
File "<stdin>", line 22, in <module>
File "lcd1602_i2c.py", line 132, in move_to
File "lcd1602_i2c.py", line 258, in hal_write_command
OSError: 5
>>>
```

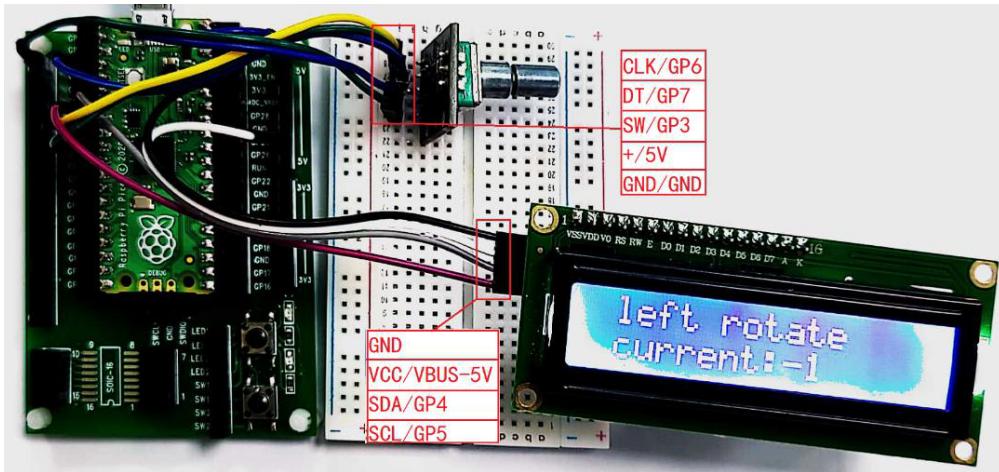
The status bar at the bottom right indicates "MicroPython (Raspberry Pi Pico)".

4.17 Capteur de rotation

Opération "main.py" Enregistrez d'abord le programme "lcd1602_i2c.py" sur raspberry pi Pico, la méthode de fonctionnement se réfère à 3.3/step3.

Note: Le nom du fichier doit être cohérent avec le nom du programme, les lettres majuscules et les symboles doivent être cohérents.

Le câblage est le suivant:



Le code du programme est le suivant:

The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - D:\树莓派\Rotation-Sensor-pico_code\lcd1602_i2c.py @ 277:1". The menu bar includes File, Edit, View, Run, Tools, and Help. The toolbar has icons for file operations like Open, Save, and Run. The left sidebar lists "main.py" and "lcd1602_i2c.py". The main code editor window displays the following Python code:

```
1 from time import sleep_ms
2
3 # The PCF8574 has a jumper selectable address: 0x20 - 0x27
4 DEFAULT_I2C_ADDR = 0x27#0x27#
5
6 # Defines shifts or masks for the various LCD line attached to the PCF8574
7
8 MASK_RS = 0x01
9 MASK_RW = 0x02
10 MASK_E = 0x04
11 SHIFT_BACKLIGHT = 3
12 SHIFT_DATA = 4
13
14
15 class LcdApi:
```

The bottom-left pane is titled "Shell" and shows the following command-line history:

```
current = -1
right rotate
current = 0
right rotate
current = 1
left rotate
current = 0
```

Thonny - D:\树莓派\Raspberry-Pi-Pico-Sensor\push_up_R\main.py @ 9:4

File Edit View Run Tools Help

main.py < lcd1602_i2c.py

```

1 from machine import I2C, Pin, Timer
2 #import time
3 from time import sleep_ms, sleep_us
4 from lcd1602_i2c import I2cLcd
5
6 DEFAULT_I2C_ADDR = 0x27
7
8 SW = machine.Pin(3, machine.Pin.IN,machine.Pin.PULL_UP)
9 CLK = machine.Pin(6, machine.Pin.IN,machine.Pin.PULL_UP)
10 DT = machine.Pin(7, machine.Pin.IN,machine.Pin.PULL_UP)
11
12 dis_cnt = 0
13
14 def tick(timer):
15     global dis_cnt
16     dis_cnt += 1

```

Shell <

```

left rotate
current = -1
fh_1 is 0
left rotate
current = -2
fh_1 is 1

```

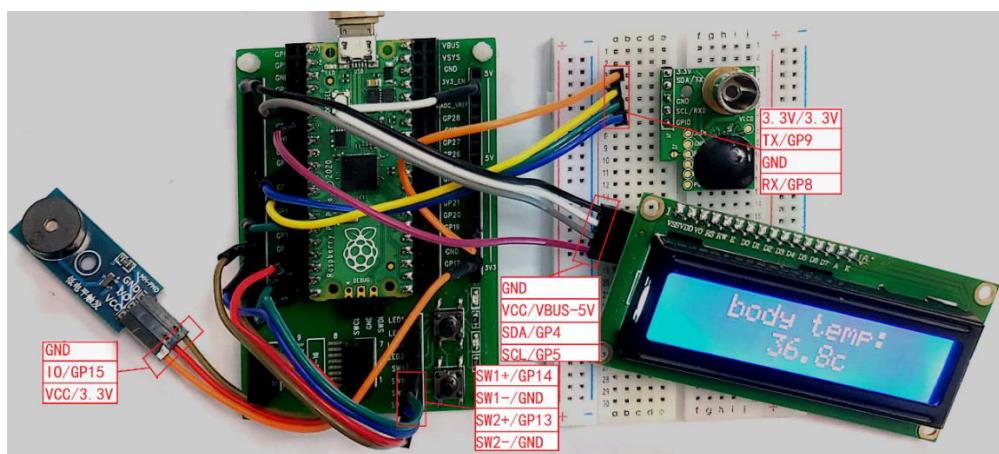
MicroPython (Raspberry Pi Pico)

4.18 Capteur de température

Opération "main.py" Enregistrez d'abord le programme "lcd1602_i2c.py" sur raspberry pi Pico, la méthode de fonctionnement se réfère à 3.3/step3.

Note: Le nom du fichier doit être cohérent avec le nom du programme, les lettres majuscules et les symboles doivent être cohérents.

Le câblage est le suivant:



Le code du programme est le suivant:

The image contains two side-by-side screenshots of the Thonny Python IDE interface, both titled "Thonny - D:\树莓派\LS_TM04\lcd1602_i2c.py @ 8:15" and "Thonny - D:\树莓派\Raspberry-Pi-Pico-Sensor\LS_TM01\main.py @ 8:22".

Screenshot 1 (Top):

```

from time import sleep_ms
# The PCF8574 has a jumper selectable address: 0x20 ~ 0x27
DEFAULT_I2C_ADDR = 0x27#0x27

# Defines shifts or masks for the various LCD line attached to the PCF8574
MASK_RS = 0x01
MASK_RW = 0x02
MASK_E = 0x04
SHIFT_BACKLIGHT = 3
SHIFT_DATA = 4

class LcdApi:
    pass

uart1: 11
uart1: 121
uart1: 0
uart1: 8
uart1: 0
uart1: 0
uart1: 210
uart1: 10

```

Screenshot 2 (Bottom):

```

from machine import I2C,UART, Pin, Timer
import utime
from lcd1602_i2c import I2CLcd
#VCC = 3.3V

DEFAULT_I2C_ADDR = 0x27
buz = machine.Pin(15, machine.Pin.OUT)
key1 = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
key2 = machine.Pin(13, machine.Pin.IN, machine.Pin.PULL_UP)

def tick(timer):
    global tb_chk_cnt
    global test_cnt
    global mode_ptr
    global key_cnt
    global key2_cnt
    global key_press

Ambient temp

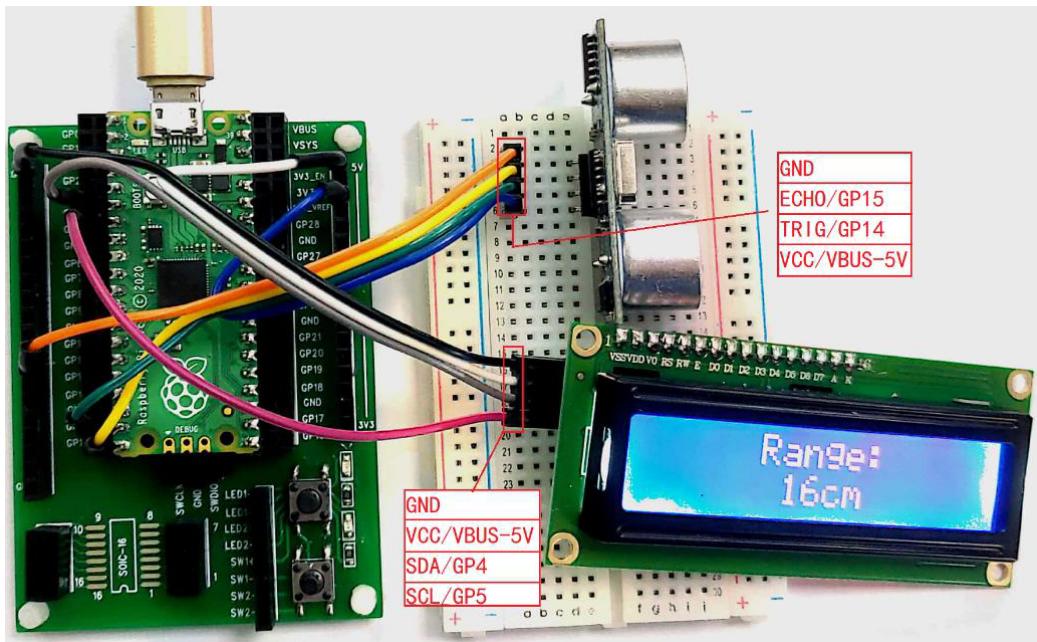
```

4.19 Capteur à ultrasons

Opération “main.py” Enregistrez d’abord le programme “lcd1602_i2c.py” sur raspberry pi Pico, la méthode de fonctionnement se réfère à 3.3/step3.

Note: Le nom du fichier doit être cohérent avec le nom du programme, les lettres majuscules et les symboles doivent être cohérents.

Le câblage est le suivant:



Le code du programme est le suivant:

Thonny - Raspberry Pi Pico :: /lcd1602_i2c.py @ 277 : 1

```

File Edit View Run Tools Help
main.py [ lcd1602_i2c.py ] x
1 from time import sleep_ms
2
3 # The PCF8574 has a jumper selectable address: 0x20 - 0x27
4 DEFAULT_I2C_ADDR = 0x27#0x27
5
6 # Defines shifts or masks for the various LCD line attached to the PCF8574
7
8 MASK_RS = 0x01
9 MASK_RW = 0x02
10 MASK_E = 0x04
11 SHIFT_BACKLIGHT = 3
12 SHIFT_DATA = 4
13
14
15 class LcdApi:
16
Shell <
>>> %Run -c $EDITOR_CONTENT
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

```

MicroPython (Raspberry Pi Pico)

Thonny - D:\树莓派\超声测距模块\pico_code\main.py @ 36 : 16

```

File Edit View Run Tools Help
main.py [ lcd1602_i2c.py ] x
1 from time import sleep_ms, sleep_us
2 from machine import I2C, Pin
3 from lcd1602_i2c import I2cLcd
4
5 TRIG_OUT = machine.Pin(14, machine.Pin.OUT)
6 ECHO_INT = Pin(15, Pin.IN, Pin.PULL_UP)
7
8 # The PCF8574 has a jumper selectable address: 0x20 - 0x27
9 DEFAULT_I2C_ADDR = 0x27
10
11
12 def read_Data(self):
13     global dat
14     if ECHO_INT.value() == 0:
15         dat = 0
16     else:
# include delay time
#include hardware device
#define LCD1602 Function device
#define HC-SR04 pin function
##define HC-SR04 pin function INPUT

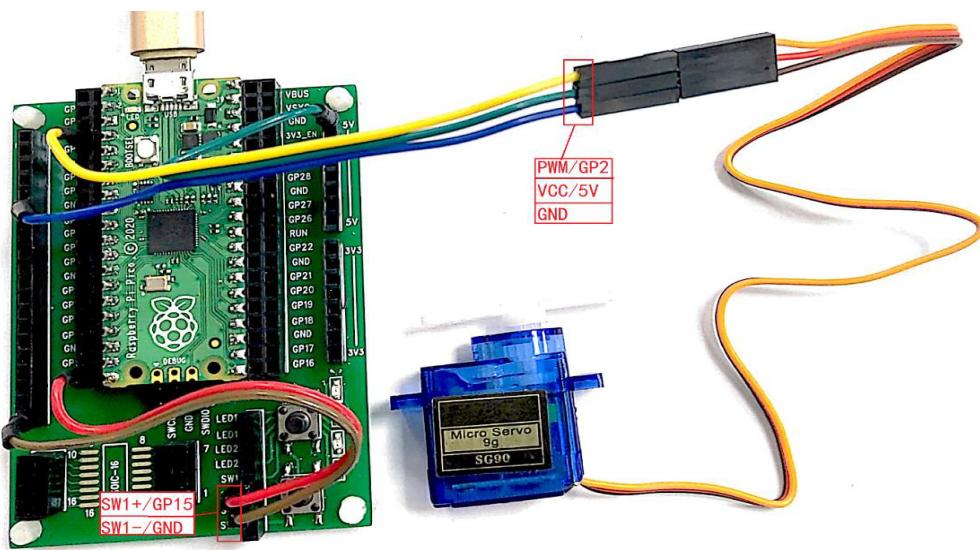
```

Shell <
>>> %Run -c \$EDITOR_CONTENT
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c \$EDITOR_CONTENT

MicroPython (Raspberry Pi Pico)

4.20 Moteur de direction

Le câblage est le suivant:



Le code du programme est le suivant:

```
Thonny - C:\Users\duncan380\Desktop\pico_0609\9g_moto\main.py @ 20:1
File Edit View Run Tools Help
main.py x
1 from machine import Pin, PWM, Timer
2 import time
3
4 pwm = PWM(Pin(2))
5 key1 = machine.Pin(15, machine.Pin.IN, machine.Pin.PULL_UP)
6
7
8 #moto run range:500--2500 us, precision: 0.09°/us,
9 #us= duty*20000/65535
10 #duty=us*65535/20000
11 pwm.freq(50)                                #50hz=2000us duty times=20000/65535
12 DUTY0 = 1638                                  #us= 500us,0°
13 DUTY45 = 3276                                  #us= 1000us,45°
14 DUTY90 = 4915                                  #us= 1500us,90°
15 DUTY100 = 5278                                 #us= 1611us,100°
16 DUTY135 = 6554                                 #us= 2000us,135°
17 DUTY150 = 7099                                 #us= 2166us,150°
18 DUTY180 = 8192                                 #us= 2500us,180°

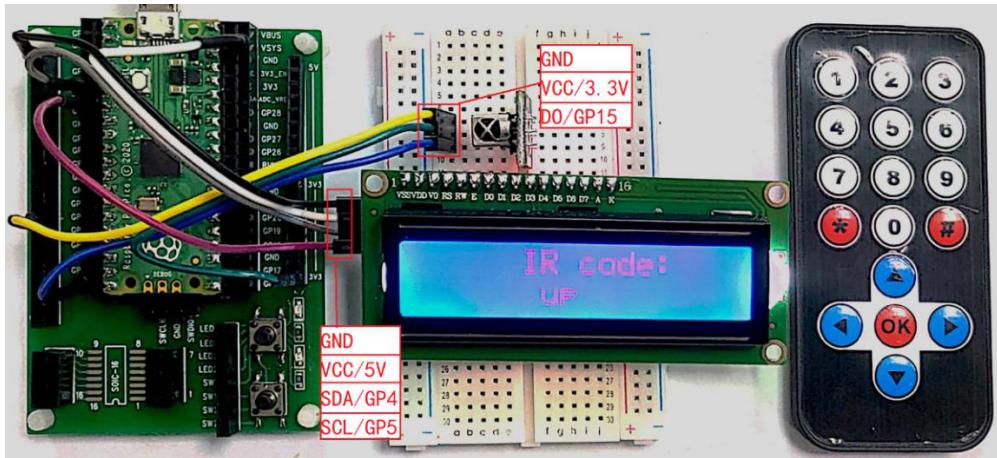
Shell x
0
45
90
100
135
150
180
0
MicroPython (Raspberry Pi Pico)
```

4.21 Télécommande infrarouge

Opération "main.py" Enregistrez d'abord le programme "lcd1602_i2c.py" sur raspberry pi Pico, la méthode de fonctionnement se réfère à 3.3/step3.

Note: Le nom du fichier doit être cohérent avec le nom du programme, les lettres majuscules et les symboles doivent être cohérents.

Le câblage est le suivant:



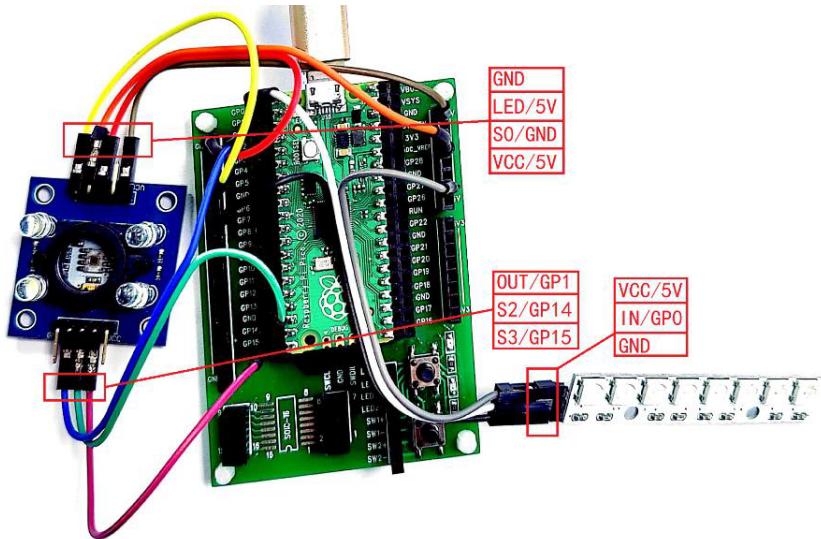
Le code du programme est le suivant:

```
Thonny - D:\树莓派\红外遥控接收解码\pico_code\main.py @ 154:33
File Edit View Run Tools Help
main.py [ lcd1602_i2c.py ]
1 from time import sleep_ms, sleep_us
2 from machine import I2C, Pin
3 from lcd1602_i2c import I2cLcd
4
5
6 IR_INT = Pin(15, Pin.IN, Pin.PULL_UP) #INPUT pin
7
8 # The PCF8574 has a jumper selectable address: 0x20 - 0x27
9 DEFAULT_I2C_ADDR = 0x27
10
11
12 def read_Data(self):
13     global dat
14     global code_dis_flag
15     if IR_INT.value() == 1:
16         print('IRf')
Shell
IRf
IRf
IRf
IRf
SLf
SHf
MicroPython (Raspberry Pi Pico)
```

4.22 Capteur de couleur

Opération "main.py" Enregistrez d'abord le programme "ws2812b.py" sur raspberry pi Pico, la méthode de fonctionnement se réfère à 3.3/step3.

Le câblage est le suivant:



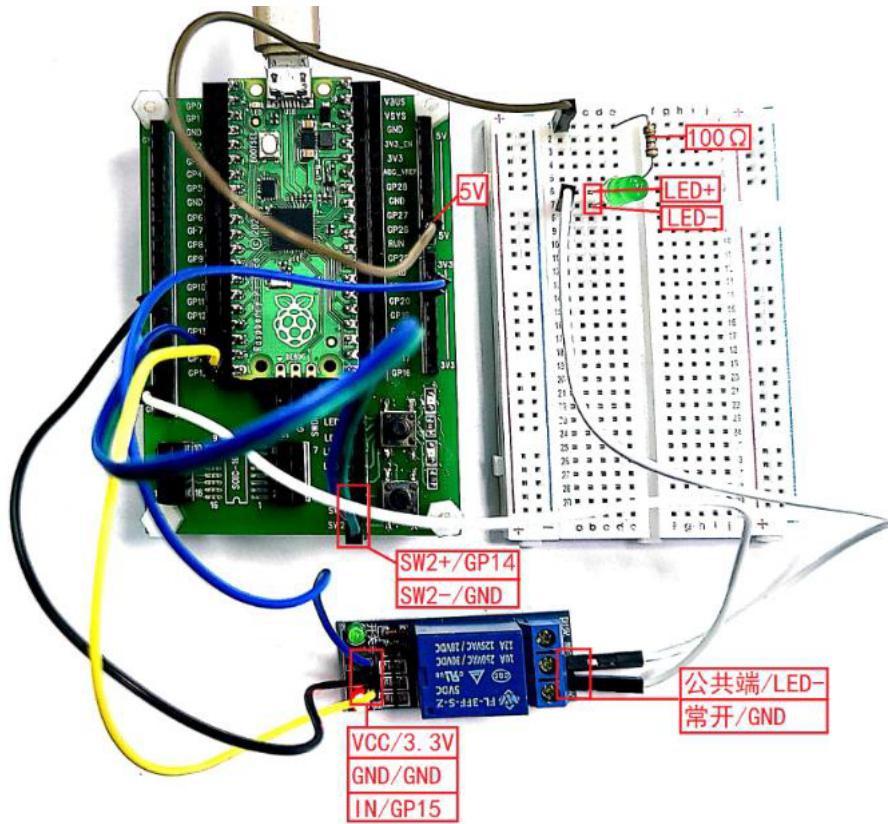
Le code du programme est le suivant:

```
Thonny - D:\树莓派\color_pico_code\ws2812b.py @ 92 : 1
File Edit View Run Tools Help
[main.py] ws2812b.py
1 import array, time
2 from machine import Pin
3 import rp2
4
5 @rp2.asm_pio(sideset_init=rp2 PIO.OUT_LOW, out_shiftdir=rp2 PIO.SHIFT_LEFT, autopull=True, pull_thresh:
6 def ws2812():
7     T1 = 2
8     T2 = 5
9     T3 = 3
10    wrap_target()
11    label("bitloop")
12    out(x, 1)      .side(0)      [T3 - 1]
13    jmp(not_x, "do_zero") .side(1)      [T1 - 1]
14    jmp("bitloop")      .side(1)      [T2 - 1]
15    label("do zero")
<   >
Shell x
>>>
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

```
Thonny - Raspberry Pi Pico ::/main.py @ 137 : 8
File Edit View Run Tools Help
[main.py] ws2812b.py
1 from machine import Pin,Timer
2 import time
3 from ws2812b import ws2812b
4
5 num_leds = 144
6 pixels = ws2812b(num_leds, 0, 0, delay=0)
7
8 #LED = VCC
9 p2 = Pin(1, Pin.IN, Pin.PULL_UP)           #OUT pin
10 #S0_out = GND                            #2% output frequency
11 #S1_out = VCC
12 S2_out = machine.Pin(14, machine.Pin.OUT)
13 S3_out = machine.Pin(15, machine.Pin.OUT)
14
15 dat = 0
16 flag = 0
Shell x
>>>
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

4.23 Relais

Le câblage est le suivant:



Le code du programme est le suivant:

```
File Edit View Run Tools Help
main.py
1 import machine
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 0
8
9 while True:
10     if aout.value() == 1:
11         led_external.value(1)
12         if flag:
13             print('led on')
14             flag = 0
15             utime.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off')
20             flag = 1
#main loop
# check AOUT pin == input high
# LED output high
#delay 0.2sec
# AOUT pin == input low
# LED output low
Shell ×
led on
led off
led on
MicroPython (Raspberry Pi Pico)
```