



Université de Gabes

Institut supérieur d'informatique et de Multimédia de Gabes



Programmation Python pour l'embarqué

Enseignant: Dr. Mohamed MCHIRI

[Contact: mohamed.mchiri@i1simg.tn](mailto:mohamed.mchiri@i1simg.tn)

Année universitaire : 2023/2024

Programmation Python pour l'embarqué

Objectifs du cours

- Découvrir et maîtriser les outils nécessaires et les aspects pratiques du langage python appliqués aux systèmes embarqués à microcontrôleurs, le Micro-python.
- Explorer les bibliothèques dédiées pour les systèmes embarqués et exploiter les plateformes Micro-python.
- Explorer la prise de contrôle de composants électroniques que l'on peut raccorder à une carte à microcontrôleur.

Programmation Python pour l'embarqué

Plan du cours

- ❑ Chapitre 1: Introduction: Qu'est-ce que Micro-python?
- ❑ Chapitre 2: Généralités sur Python 3
- ❑ **Chapitre 3: Plateformes Micro-python**
- ❑ Chapitre 4: Réalisations

Programmation Python pour l'embarqué

Plan du cours

□ Chapitre 3: Plateformes Micro-python

- L'IDE Thonny
- NodeMCU ESP8266
- ESP32
- Pi Pico

Chapitre 3

Plateformes Micropython

❑ Chapitre 3: Plateformes Micro-python

- L'IDE Thonny
- NodeMCU ESP8266
- ESP32
- Pi Pico

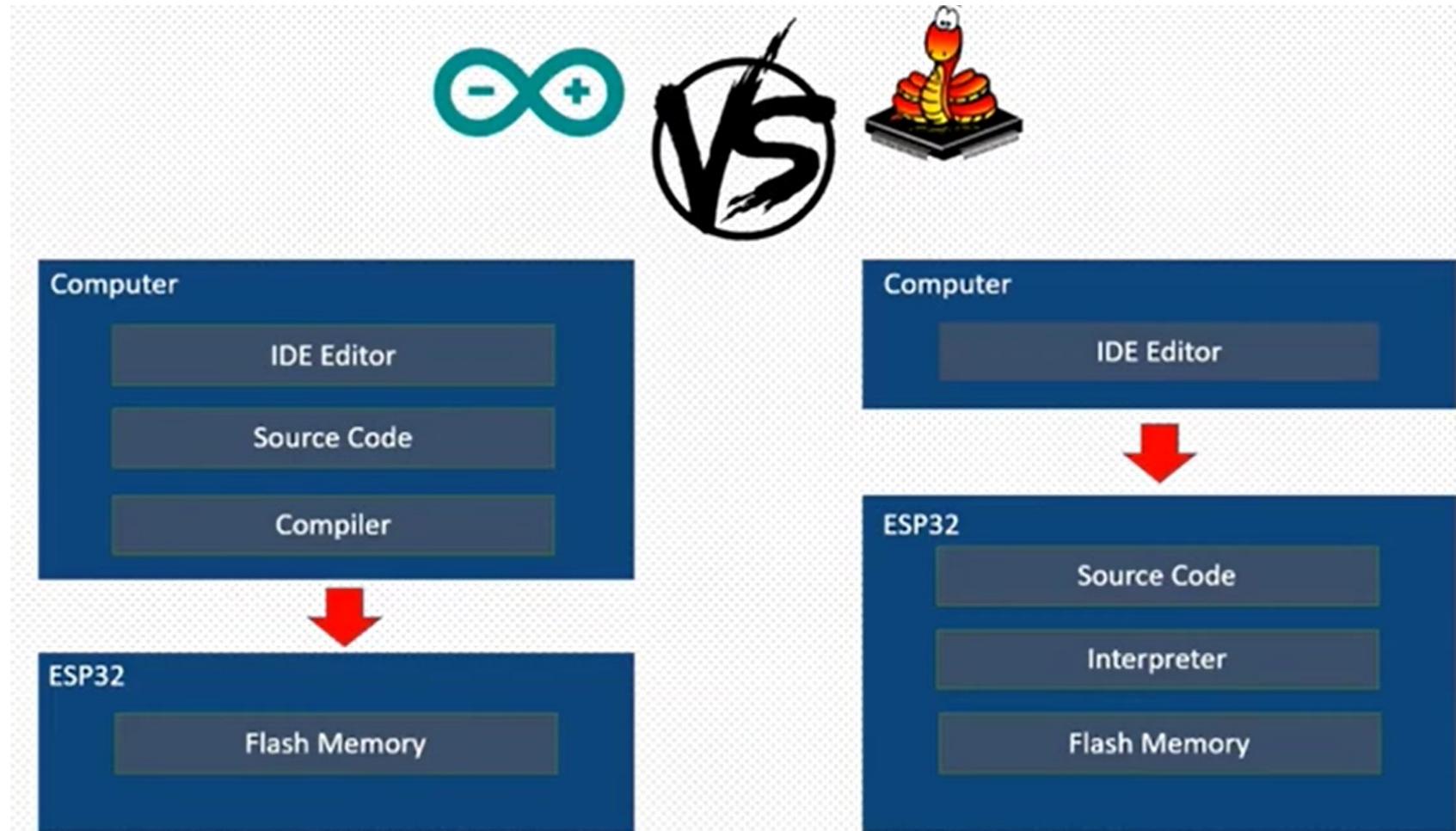
Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Rappel:

- **Micropython : interprété**
- **Arduino: compilé**

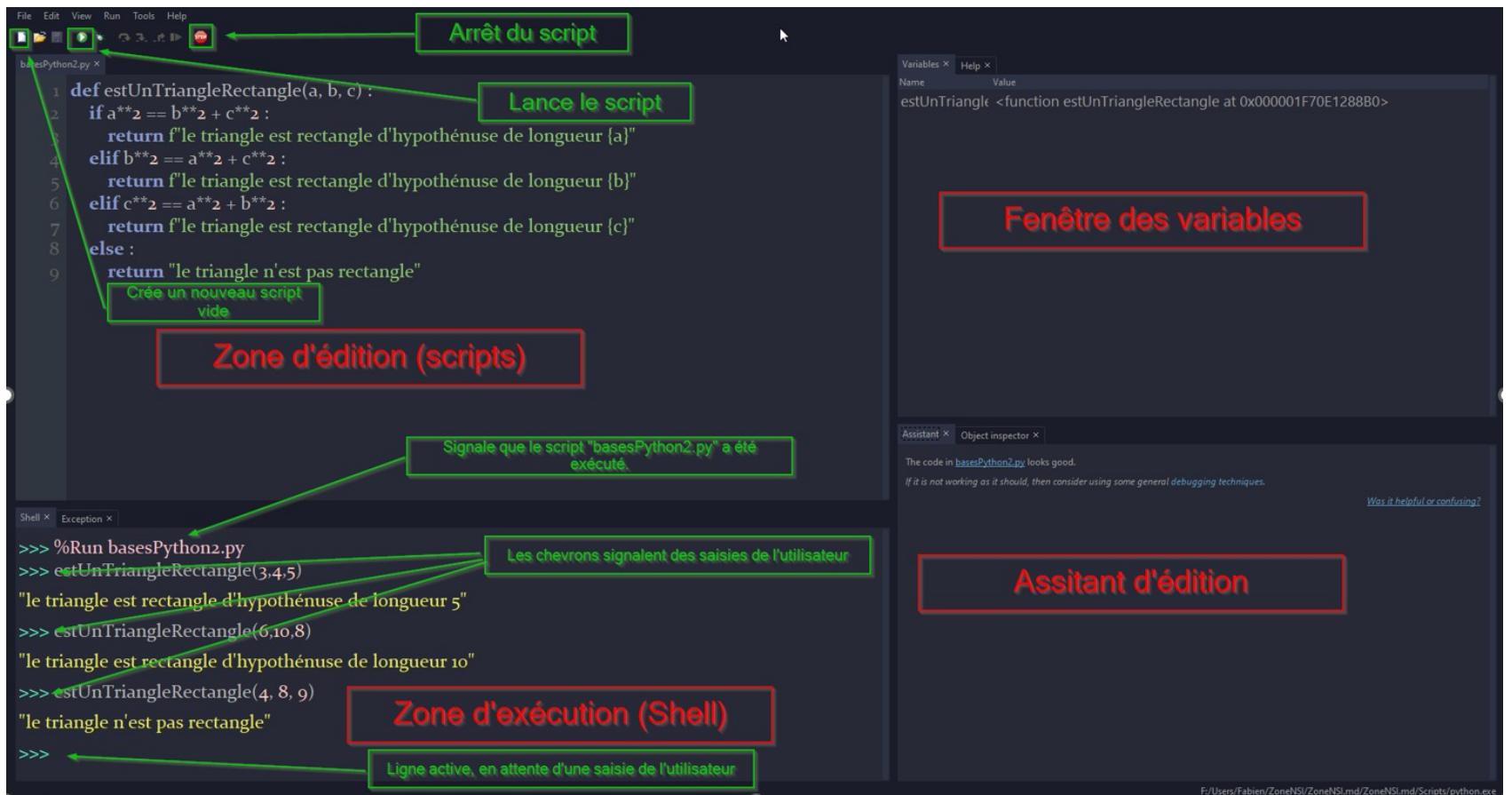


Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

- **Thonny**: est un IDE libre et gratuit, simple à installer, offrant une interface épurée pour commencer facilement et rapidement la programmation Python.



- **Zone d'exécution**: aussi appelée **Shell** ou **console**, est une zone *interactive* où une instruction Python est exécutée directement après avoir été tapée. Cette zone est souvent utilisée pour *tester une instruction ou série d'instructions*, mais est aussi l'endroit où sera exécuté un **script Python**. Attention, dans le **Shell**, il n'est pas possible de **revenir en arrière et de modifier ce qui a été tapé**. En cas d'erreur, il faudra recommencer !

- **Zone d'édition** ou **zone des scripts** est une zone dans laquelle on peut taper des séries d'instructions Python, et les sauvegarder sous la forme d'un fichier d'extension **.py**. Ces instructions ne seront pas exécutées tant que l'utilisateur n'aura pas demandé explicitement cette exécution.
Pour **exécuter un script**, il faudra appuyer, soit sur **la flèche verte** de la barre de menu, soit sur la touche **F5**, soit par l'intermédiaire du menu **Run>Run current script**. Le résultat de l'exécution du script sera affiché dans le Shell.

- Les zones situées à droite de l'éditeur (fenêtre des variables et assistant d'édition), contiennent des informations qui peuvent être utiles pour analyser un programme qui ne fonctionne pas.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

Téléchargement :

Sur Windows, macOS et Linux, vous pouvez aussi installer l'IDE Thonny ou mettre à jour une version existante.

Allez à l'adresse: <http://thonny.org/>

La version portable fonctionne sur clé USB.

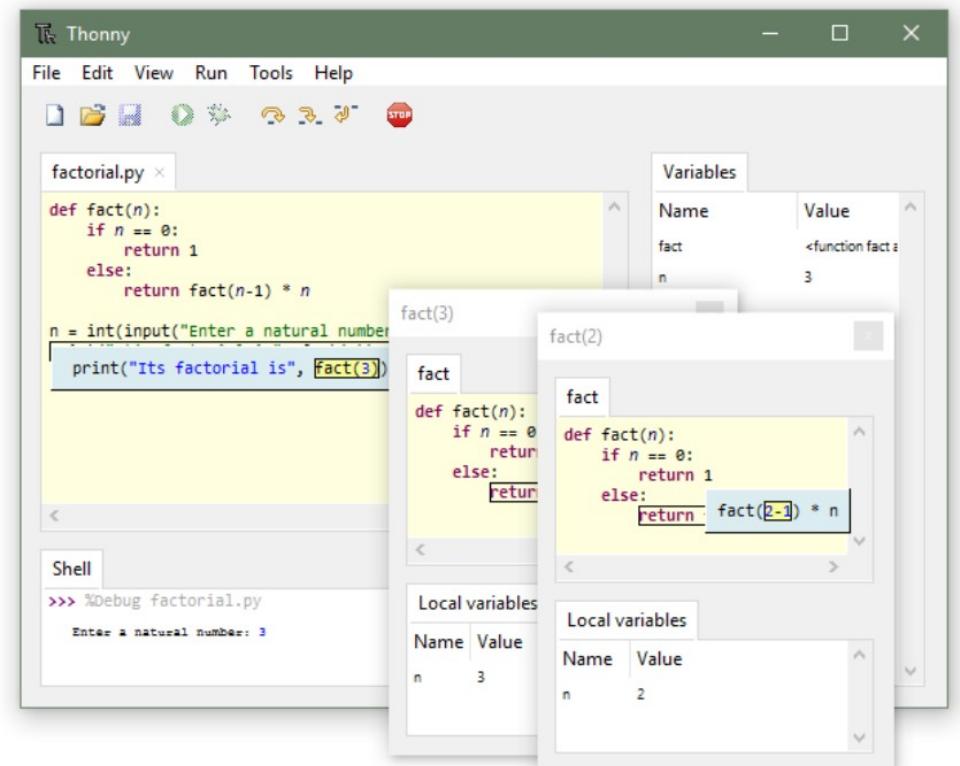
Téléchargez les fichiers correspondant à votre OS et exécutez-les pour installer **Thonny**.

Thonny

Python IDE for beginners



Download version [4.1.3](#) for
Windows • Mac • Linux



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

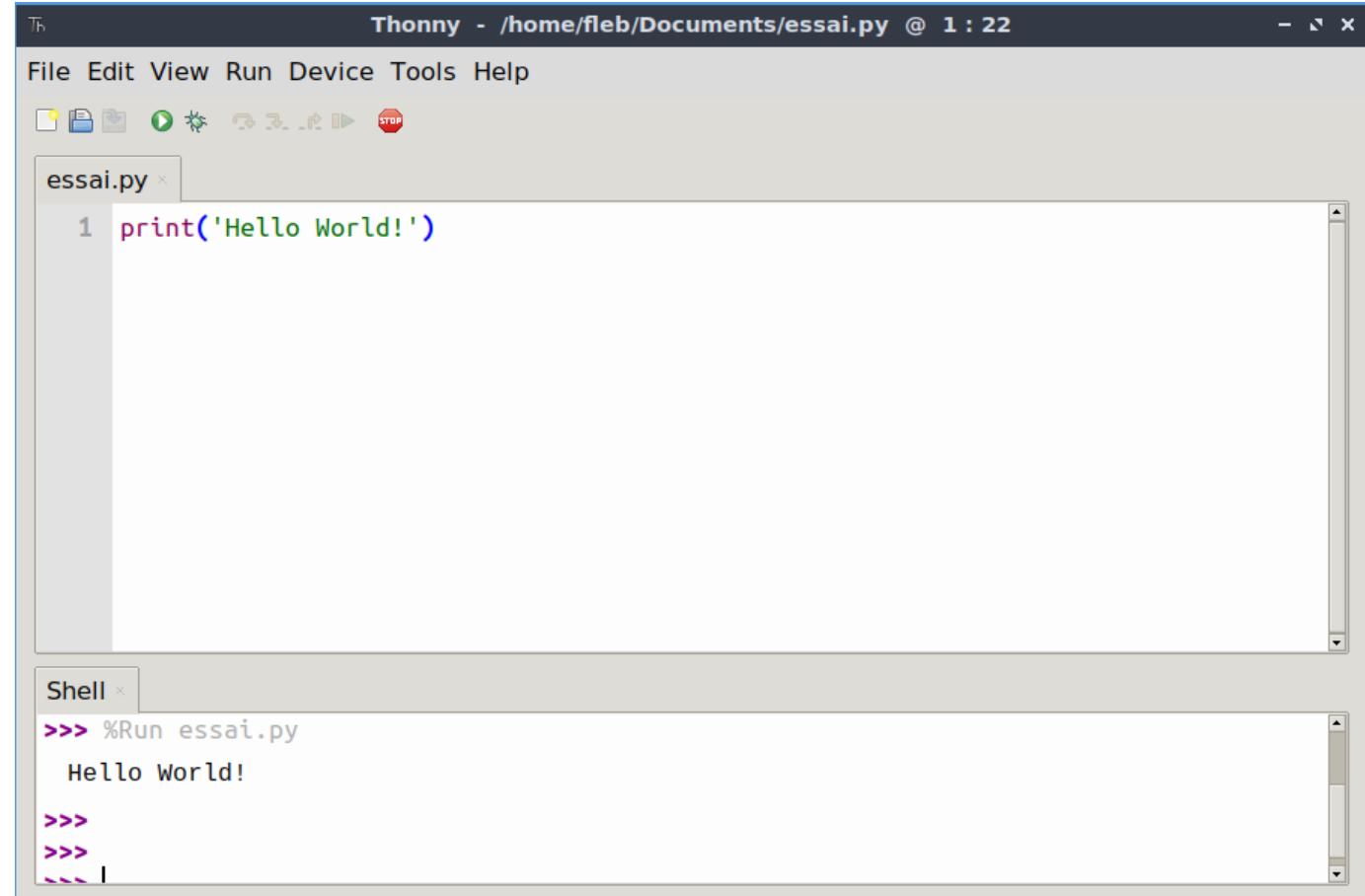
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Lancez l'application Thonny.

Vous pouvez utiliser Thonny pour écrire du code Python standard. Tapez la ligne suivante dans la fenêtre principale de l'éditeur:

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

puis cliquez sur le bouton Run (l'application vous demandera d'abord de renseigner l'emplacement et le nom du fichier à sauvegarder) :



The screenshot shows the Thonny IDE interface. At the top, the title bar reads "Thonny - /home/fleb/Documents/essai.py @ 1 : 22". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations and run/stop. A code editor window titled "essai.py" contains the single line of code: "1 print('Hello World!')". Below the code editor is a "Shell" window. In the shell, the command ">>> %Run essai.py" is entered, followed by the output "Hello World!". The shell window has scroll bars on the right side.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

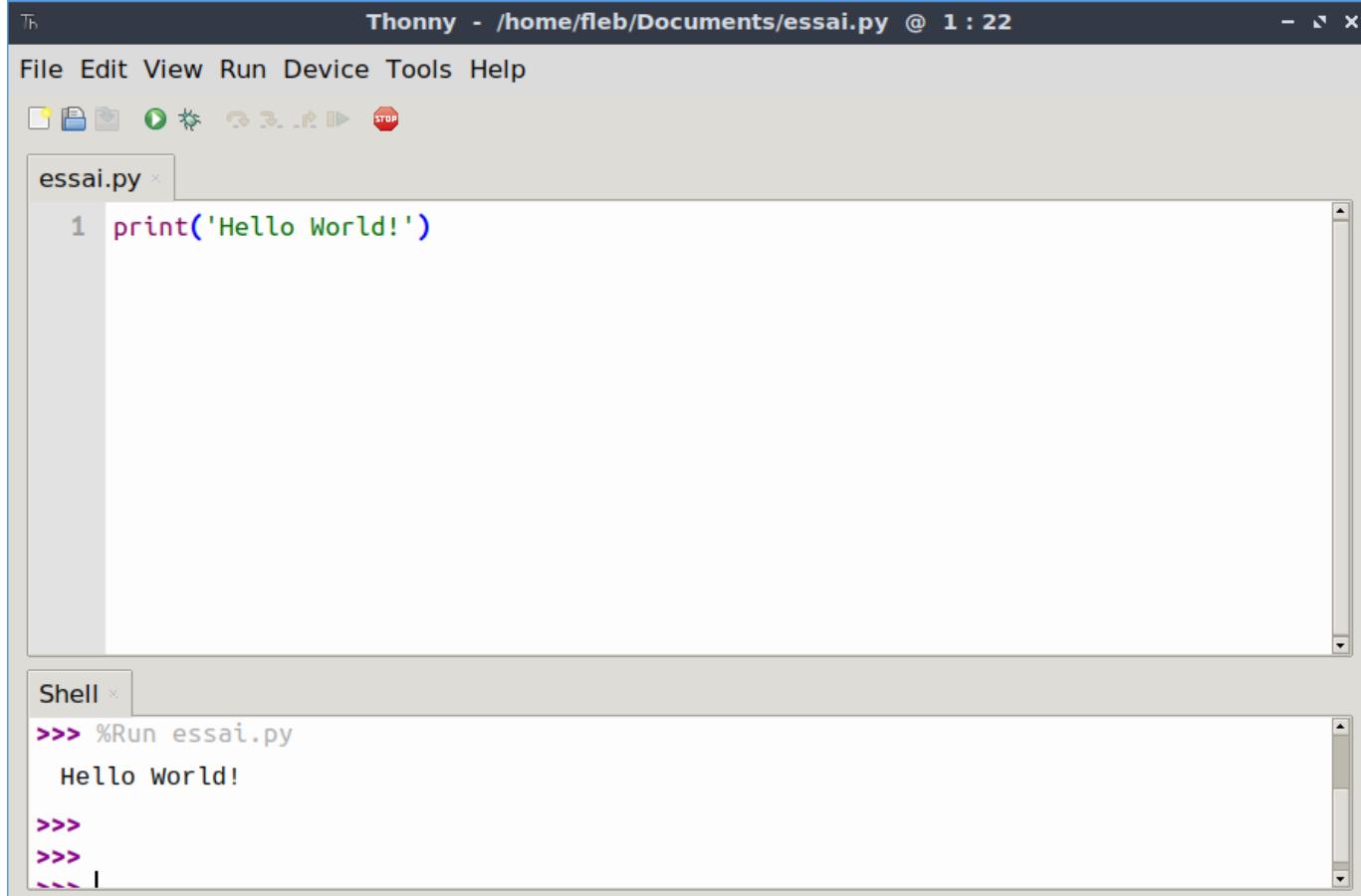
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Lancez l'application Thonny.

Vous pouvez utiliser Thonny pour écrire du code Python standard. Tapez la ligne suivante dans la fenêtre principale de l'éditeur:

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

puis cliquez sur le bouton Run (l'application vous demandera d'abord de renseigner l'emplacement et le nom du fichier à sauvegarder) :



```
Thonny - /home/fleb/Documents/essai.py @ 1 : 22
File Edit View Run Device Tools Help
Thonny
essai.py
1 print('Hello World!')
Shell
>>> %Run essai.py
Hello World!
>>>
>>>
>>>
```

Chapitre 3

Plateformes Micropython

❑ Chapitre 3: Plateformes Micro-python

- L'IDE Thonny
- **NodeMCU ESP8266**
- ESP32
- Pi Pico

Chapitre 3 : Plateformes MicroPython

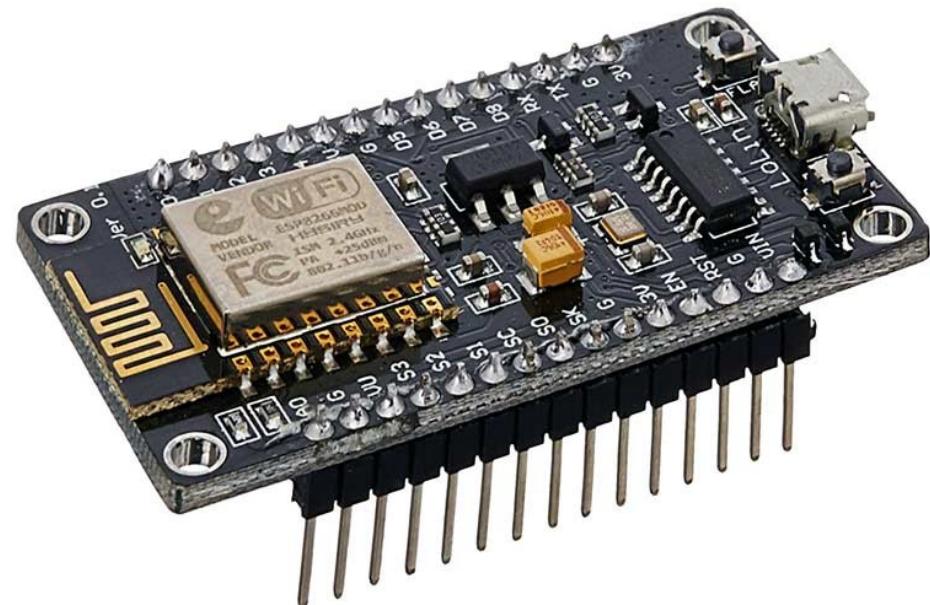
Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

- L'**ESP8266**, développé par le fabricant chinois **Espressif**, est un SoC Wi-Fi intégrant un microprocesseur Tensilica Xtensa 32 bits LX106, souvent utilisé dans les applications IoT.
- **NodeMCU** est une plate-forme open source IoT, matérielle et logicielle, basée sur un SoC Wi-Fi ESP8266 ESP-12. Le terme « NodeMCU » se réfère par défaut au firmware plutôt qu'aux kits de développement.
- L'ESP8266 est disponible dans une grande variété de versions (ESP01-ESP02...). L'**ESP-12E NodeMCU** est actuellement la version la plus pratique.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

- L'**ESP8266** peut être utilisé comme appareil autonome ou comme adaptateur UART vers Wi-Fi pour permettre à d'autres microcontrôleurs de se connecter à un réseau Wi-Fi. Par exemple, vous pouvez connecter un ESP8266 à un **Arduino** pour ajouter des fonctionnalités Wi-Fi à votre carte Arduino.
- Avec l'ESP8266, vous pouvez contrôler les entrées et les sorties comme vous le feriez avec un Arduino, mais avec des capacités **Wi-Fi**. Cela signifie que vous pouvez mettre vos projets **en ligne**, ce qui est idéal pour les applications de domotique et d'**Internet des objets**.
- L'ESP8266 consomme **très peu d'énergie** par rapport aux autres microcontrôleurs et peut même passer en mode veille profonde pour consommer moins d'énergie ;

- L'ESP8266 peut générer son propre réseau **Wi-Fi** (point d'accès) ou se connecter à d'autres réseaux Wi-Fi (station) pour accéder à Internet. Cela signifie que l'ESP8266 peut accéder aux services en ligne pour effectuer des requêtes **HTTP** ou enregistrer des données dans le **cloud**, par exemple. Il peut également agir comme un **serveur Web** afin que vous puissiez y accéder à l'aide d'un navigateur Web et pouvoir contrôler et surveiller vos cartes **à distance**.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

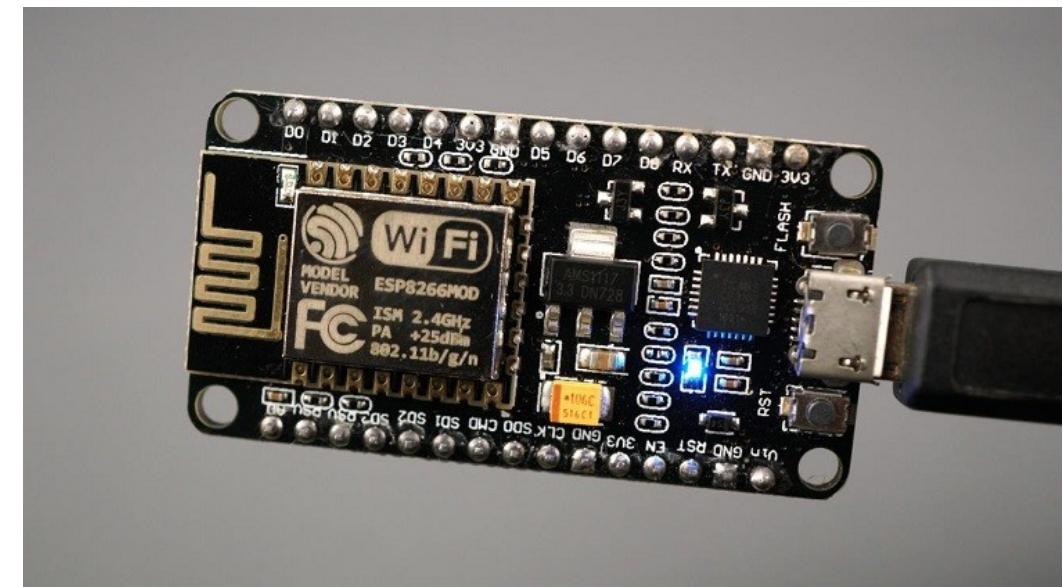
Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

L'**ESP 12-E NodeMCU** est l'une des cartes de développement **ESP8266** les plus utilisées. Il dispose de **4 Mo** de **mémoire flash**, d'un accès à **11 broches GPIO** et d'une broche de **ADC** avec une résolution de **10 bits**. De plus, la carte dispose d'un régulateur de tension intégré et vous pouvez alimenter le module à l'aide de la prise mini **USB** ou de la broche **Vin**.

L'**ESP 12-E NodeMCU** n'a pas besoin d'un programmateur FTDI ou de circuits supplémentaires, car il dispose d'un convertisseur USB-série intégré. La plupart des cartes sont livrées avec les puces **CP2101** ou **CH340**.



L'**ESP 12-E NodeMCU** est livré avec une **antenne** intégrée pour le signal **Wi-Fi** et des boutons **RST** et **FLASH** pour réinitialiser la carte et la mettre en mode clignotant. Il y a une **LED bleue** connectée en interne au **GPIO 2**, ce qui est très pratique pour le débogage.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

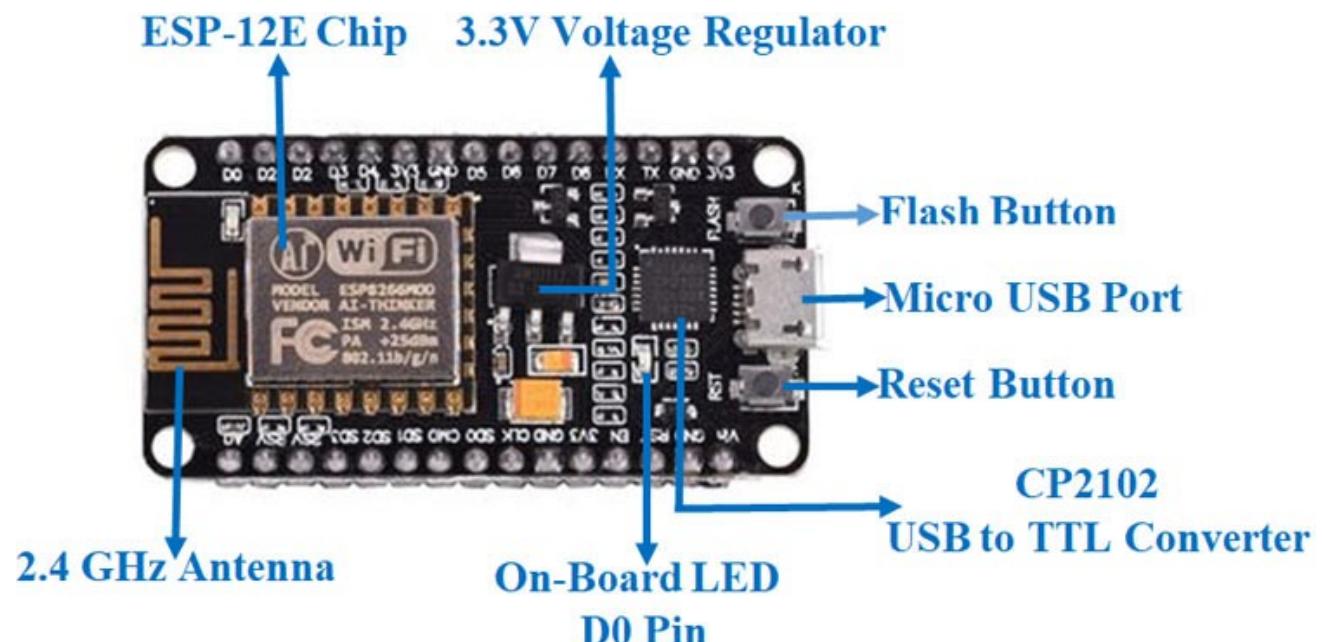
Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

■ Caractéristiques de l'ESP8266:

- **Microcontroller:** Tensilica 32-bit RISC CPU Xtensa LX106
- **Operating Voltage:** 3.3V
- **Input Voltage:** 7-12V
- **Digital I/O Pins (DIO):** 16
- **Analog Input Pins (ADC):** 1
- **UARTs:** 1
- **SPIs:** 1
- **I2Cs:** 1
- **Flash Memory:** 4 MB
- **SRAM:** 64 KB
- **Clock Speed:** 80 MHz
- **USB-TTL based on CP2102** is included onboard
- **PCB Antenna**



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Configuration PinOut de l'ESP8266:

Pin Category	Name	Description
Power	Micro-USB, 3.3V, GND, Vin	Micro-USB: NodeMCU can be powered through the USB port 3.3V: Regulated 3.3V can be supplied to this pin to power the board GND: Ground pins Vin: External Power Supply
Control Pins	EN, RST	The pin and the button resets the microcontroller

Analog Pin	A0	Used to measure analog voltage in the range of 0-3.3V
GPIO Pins	GPIO1 to GPIO16	NodeMCU has 16 general purpose input-output pins on its board
SPI Pins	SD1, CMD, SD0, CLK	NodeMCU has four pins available for SPI communication.
UART Pins	TXD0, RXD0, TXD2, RXD2	NodeMCU has two UART interfaces, UART0 (RXD0 & TXD0) and UART1 (RXD1 & TXD1). UART1 is used to upload the firmware/program.
I2C Pins		NodeMCU has I2C functionality support but due to the internal functionality of these pins, you have to find which pin is I2C.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

■ Brochage de l'ESP8266:

Broches d'alimentation : 4 pins: 1 Vin et 3 pins 3,3 V. Le pin Vin peut être utilisé pour alimenter directement le ESP8266 et ses périphériques, si vous disposez d'une alimentation 5V régulée. La broche 3,3 V est la sortie du régulateur de tension embarqué ; vous pouvez en tirer jusqu'à 600 mA.

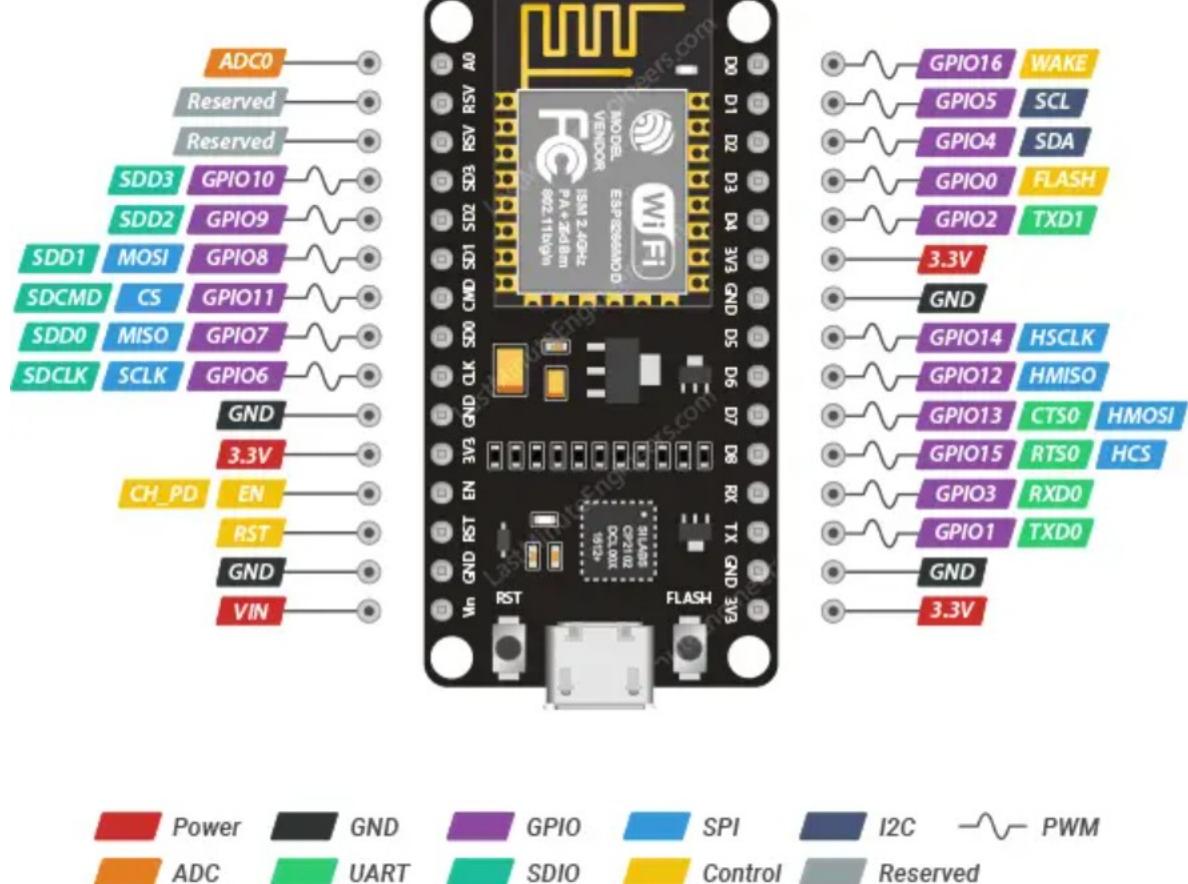
GND: Ground Pin

Broches GPIO: 17 broches GPIO auxquelles il est possible d'attribuer différentes fonctions en programmant les registres appropriés.

Canal ADC: Le ESP8266 comprend 1 ADC de précision 10 bits. L'ADC peut être utilisé pour effectuer deux mesures : tester la tension d'alimentation de la broche VDD 3.3 et tester la tension d'entrée de la broche TOUT.

Broches SPI: ESP8266 dispose de deux SPI (SPI et HSPI) en mode esclave et maître. Ces SPI prennent également en charge les fonctionnalités SPI à usage général.

Broches I2C: GPIO4 (SDA) et GPIO5 (SCL) sont utilisés comme broches I2C pour faciliter l'utilisation du code et des bibliothèques existants.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

■ Brochage de l'ESP8266:

Broches UART: Le ESP8266 dispose de deux interfaces UART, UART0 et UART2, qui prennent en charge la communication asynchrone (RS232 et RS485) jusqu'à 4,5 Mbit/s. UART0 (broches TXD0, RXD0, RST0 et CTS0) est utilisé pour la communication, tandis que UART1 (broche TXD1) ne dispose que d'un signal de transmission de données et est généralement utilisé pour l'impression de journaux.

~ Broches PWM : La sortie PWM (Pulse Width Modulation) peut être implémentée par programmation sur toutes les broches GPIO de GPIO0 à GPIO15. Sur le ESP8266, le signal PWM a une résolution de 10 bits, et la gamme de fréquences PWM est réglable entre 1000 µs et 10000 µs, c'est-à-dire entre 100 Hz et 1 kHz.

Broches SDIO: Le ESP8266 dispose d'une interface SDIO (Secure Digital Input/Output Interface) esclave pour connecter les cartes SD. SDIO v1.1 (4 bits, 25 MHz) et SDIO v2.0 (4 bits, 50 MHz) sont pris en charge.

Broches de contrôle sont utilisés pour contrôler le ESP8266. Ces broches sont:

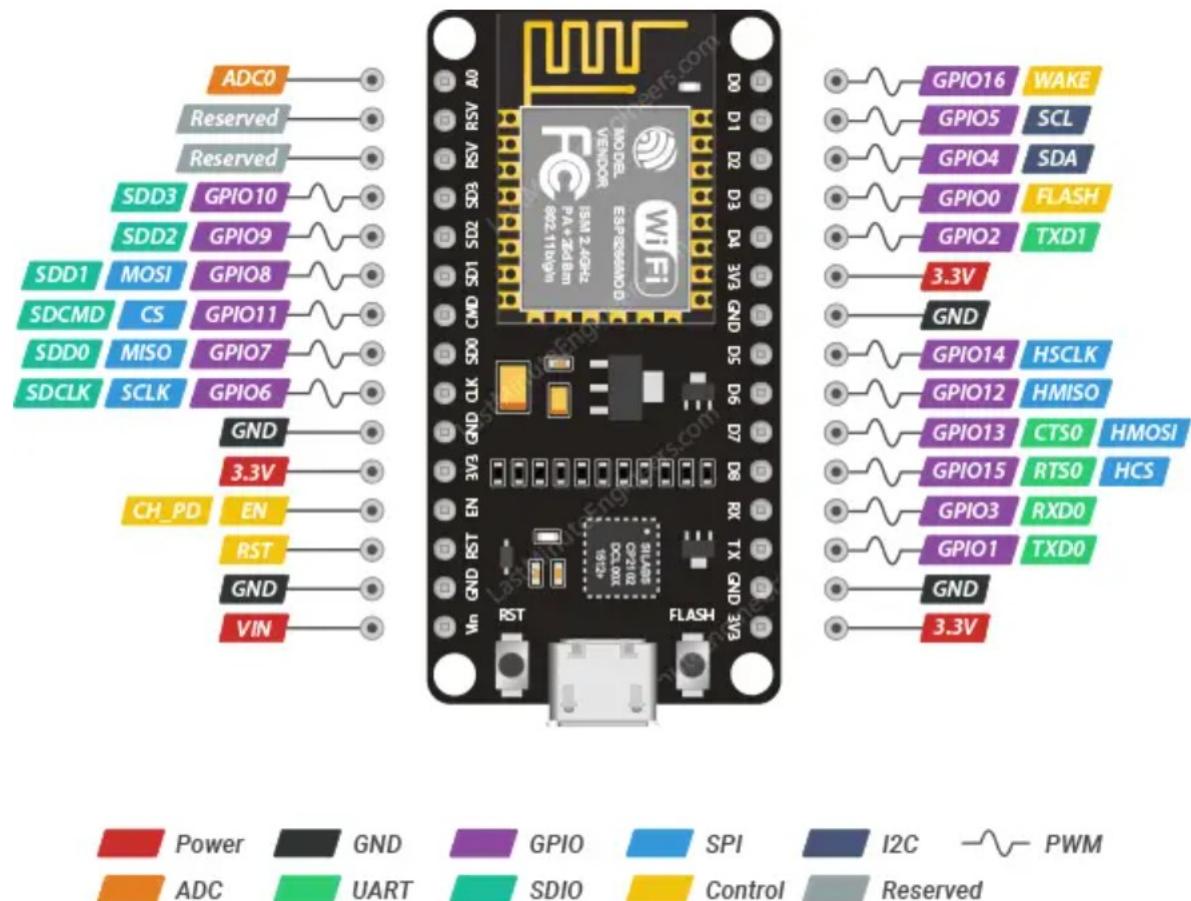
la broche EN, la broche RST, la broche FLASH et la broche WAKE.

EN: pour activer le ESP8266

RST: réinitialiser le ESP8266

FLASH: pour le boot

WAKE: réveiller le ESP8266 du sommeil profond



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

■ Brochage de l'ESP8266:

Label	GPIO	Input	Output	Notes
D0	GPIO16	no interrupt	no PWM or I2C support	HIGH at boot used to wake up from deep sleep
D1	GPIO5	OK	OK	often used as SCL (I2C)
D2	GPIO4	OK	OK	often used as SDA (I2C)
D3	GPIO0	pulled up	OK	connected to FLASH button, boot fails if pulled LOW
D4	GPIO2	pulled up	OK	HIGH at boot connected to on-board LED, boot fails if pulled LOW

D5	GPIO14	OK	OK	SPI (SCLK)
D6	GPIO12	OK	OK	SPI (MISO)
D7	GPIO13	OK	OK	SPI (MOSI)
D8	GPIO15	pulled to GND	OK	SPI (CS) Boot fails if pulled HIGH
RX	GPIO3	OK	RX pin	HIGH at boot
TX	GPIO1	TX pin	OK	HIGH at boot debug output at boot, boot fails if pulled LOW
A0	ADC0	Analog Input	X	

GPIO4 and GPIO5 are the **most safe** to use GPIOs if you want to operate **relays**.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Installer MicroPython sur la carte ESP8266:

Installation des pilotes USB :

Lors du branchement du câble USB sur le NodeMCU, il faudra utiliser l'explorateur de fichiers de Windows indiquant le numéro de port utilisé, ex: COM4.

- >  Périphériques système
- ▼  Ports (COM et LPT)
 -  Port de communication (COM1)
 -  Silicon Labs CP210x USB to UART Bridge (COM4)
- >  Processeurs

Si le port COM n'est pas identifiable, il faudrait envisager d'installer le driver USB. La plupart des cartes ESP8266 utilisent les pilotes **CP2101** ou **CH340**. Installez les pilotes correspondants.

Download and Install VCP Drivers

Downloads for Windows, Macintosh, Linux and Android below.

*Note: The Linux 3.x.x and 4.x.x version of the driver is maintained in the current Linux 3.x.x and 4.x.x tree at www.kernel.org.

Legacy OS Software Versions

Driver Package download links and support information

Serial Enumeration Driver

What is the serial enumeration driver and why would I need it?

Software Downloads

Software (10)

Software · 10

CP210x Universal Windows Driver	v10.1.10 1/13/2021
CP210x VCP Mac OSX Driver	v6.0.1 3/31/2021
CP210x Windows Drivers	v6.7.6 9/3/2020
CP210x Windows Drivers with Serial Enumerator	v6.7.6 9/3/2020
CP210x_5x_AppNote_Archive	9/3/2020
Show 5 more Software	

□ Installer MicroPython sur la carte ESP8266:

Installation de Firmware:

- MicroPython est un interpréteur Python optimisé pour les microcontrôleurs comme les cartes ESP8266. On peut écrire des scripts Python qui seront directement exécutés sur une carte ESP8266. Pour ce faire, il suffit de flasher la carte ESP32 avec MicroPython dessus et d'utiliser un logiciel IDE (par exemple *Thonny IDE*) pour coder des scripts Python et les envoyer à l'ESP8266.
- Le firmware de Micropython est disponible sur le site officiel:
www.micropython.org
- Pensez bien à prendre la version stable la plus récente et de prendre le fichier avec l'extension **.bin**

Firmware

Releases

v1.21.0 (2023-10-05) .bin / [.elf] / [.map] / [Release notes] (latest)
v1.20.0 (2023-04-26) .bin / [.elf] / [.map] / [Release notes]
v1.19.1 (2022-06-18) .bin / [.elf] / [.map] / [Release notes]
v1.18 (2022-01-17) .bin / [.elf] / [.map] / [Release notes]
v1.17 (2021-09-02) .bin / [.elf] / [.map] / [Release notes]
v1.16 (2021-06-18) .bin / [.elf] / [.map] / [Release notes]
v1.15 (2021-04-18) .bin / [.elf] / [.map] / [Release notes]
v1.14 (2021-02-02) .bin / [.elf] / [.map] / [Release notes]
v1.13 (2020-09-11) .bin / [.elf] / [.map] / [Release notes]
v1.12 (2019-12-20) .bin / [.elf] / [.map] / [Release notes]
v1.11 (2019-05-29) .bin / [.elf] / [.map] / [Release notes]
v1.10 (2019-01-25) .bin / [.elf] / [.map] / [Release notes]
v1.9.4 (2018-05-11) .bin / [.elf] / [.map] / [Release notes]
v1.9.3 (2017-11-01) .bin / [.elf] / [.map] / [Release notes]
v1.9.2 (2017-08-23) .bin / [.elf] / [.map] / [Release notes]
v1.9.1 (2017-06-12) .bin / [.elf] / [.map] / [Release notes]
v1.9 (2017-05-26) .bin / [.elf] / [.map] / [Release notes]
v1.8.7 (2017-01-08) .bin / [.elf] / [.map] / [Release notes]

Deux méthodes sont proposées pour installer Micropython sur votre carte ESP8266 :

- Utiliser un IDE (exemple Thonny IDE)
- L'outil esptool

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

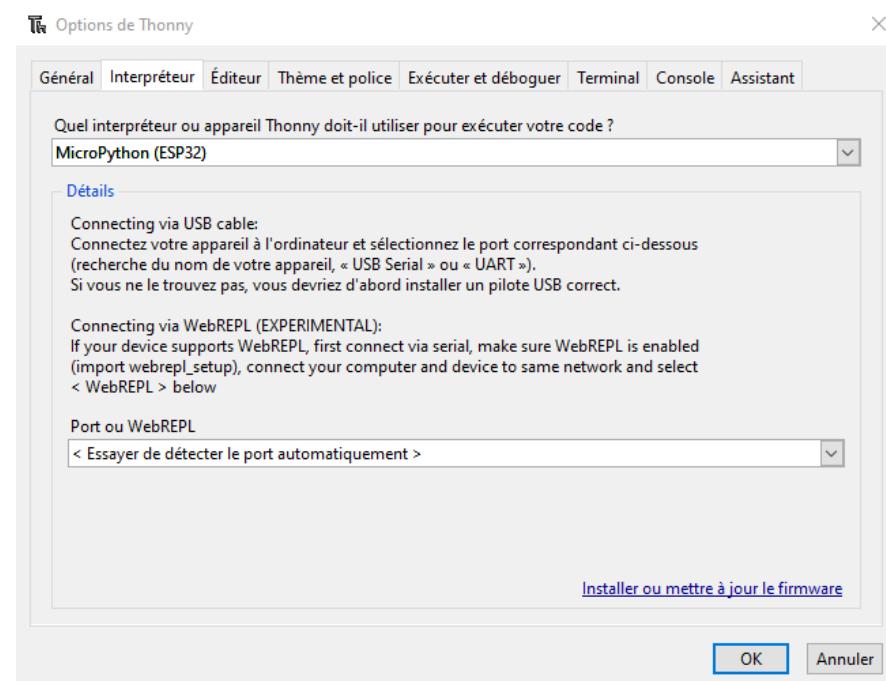
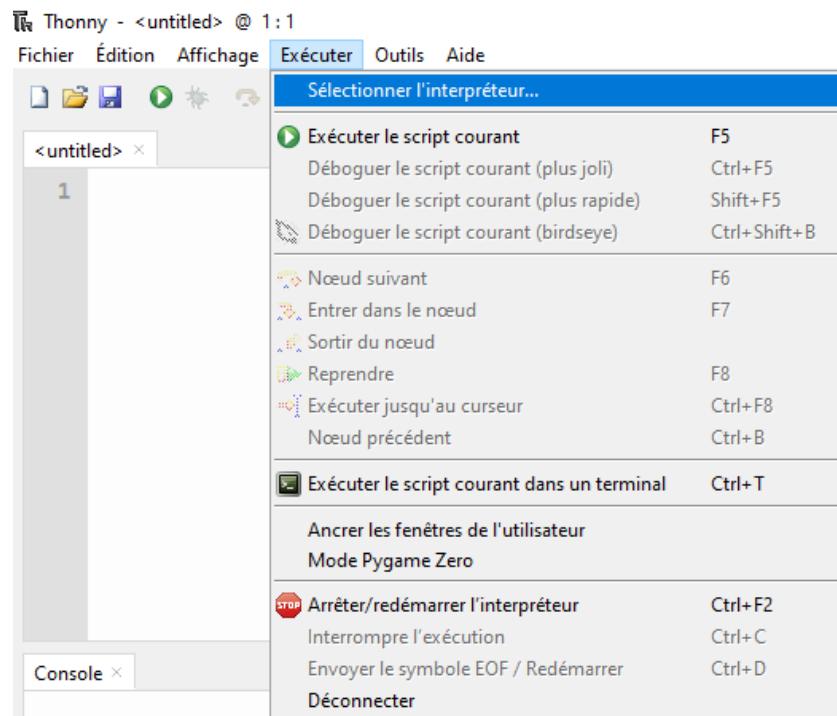
Section 3 : ESP32

Section 4 : PI Pico

☐ Installer MicroPython sur la carte ESP8266:

Flasher la carte avec Thonny IDE:

Pour installer MicroPython sur l'ESP8266, il suffit de se rendre choisir l'interpréteur « MicroPython (ESP8266) » puis cliquer sur « *Installer ou mettre à jour le firmware* ».



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

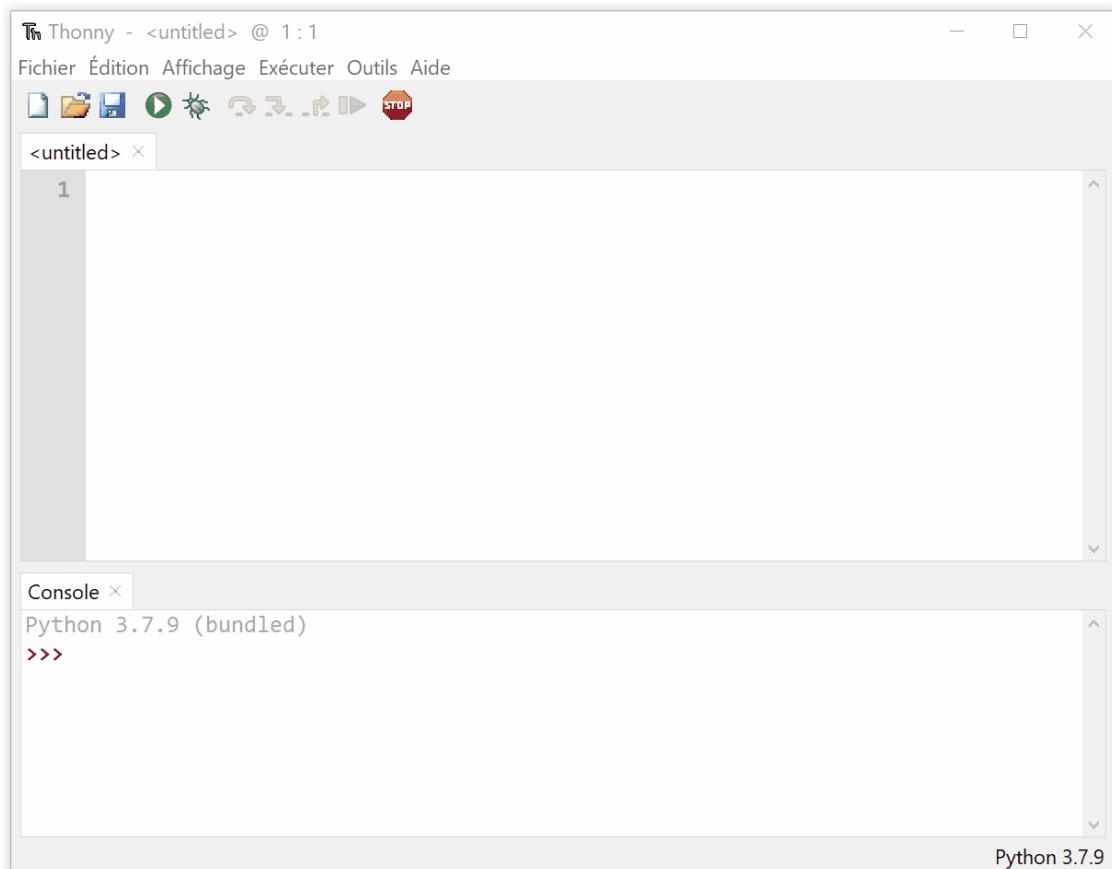
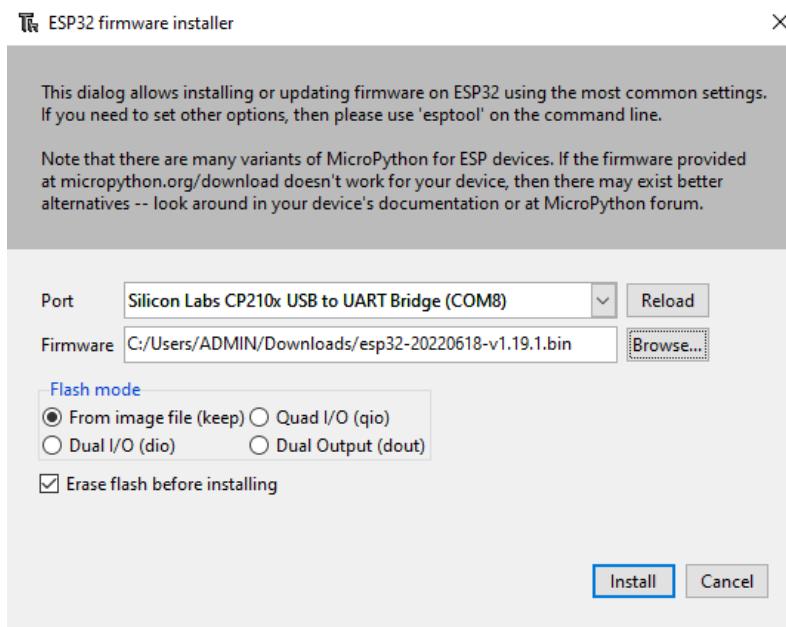
Section 3 : ESP32

Section 4 : PI Pico

☐ Installer MicroPython sur la carte ESP8266:

Flasher la carte avec Thonny IDE:

Sélectionnez votre carte et le fichier contenant le firmware précédemment téléchargé. Puis appuyez sur « Install »



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

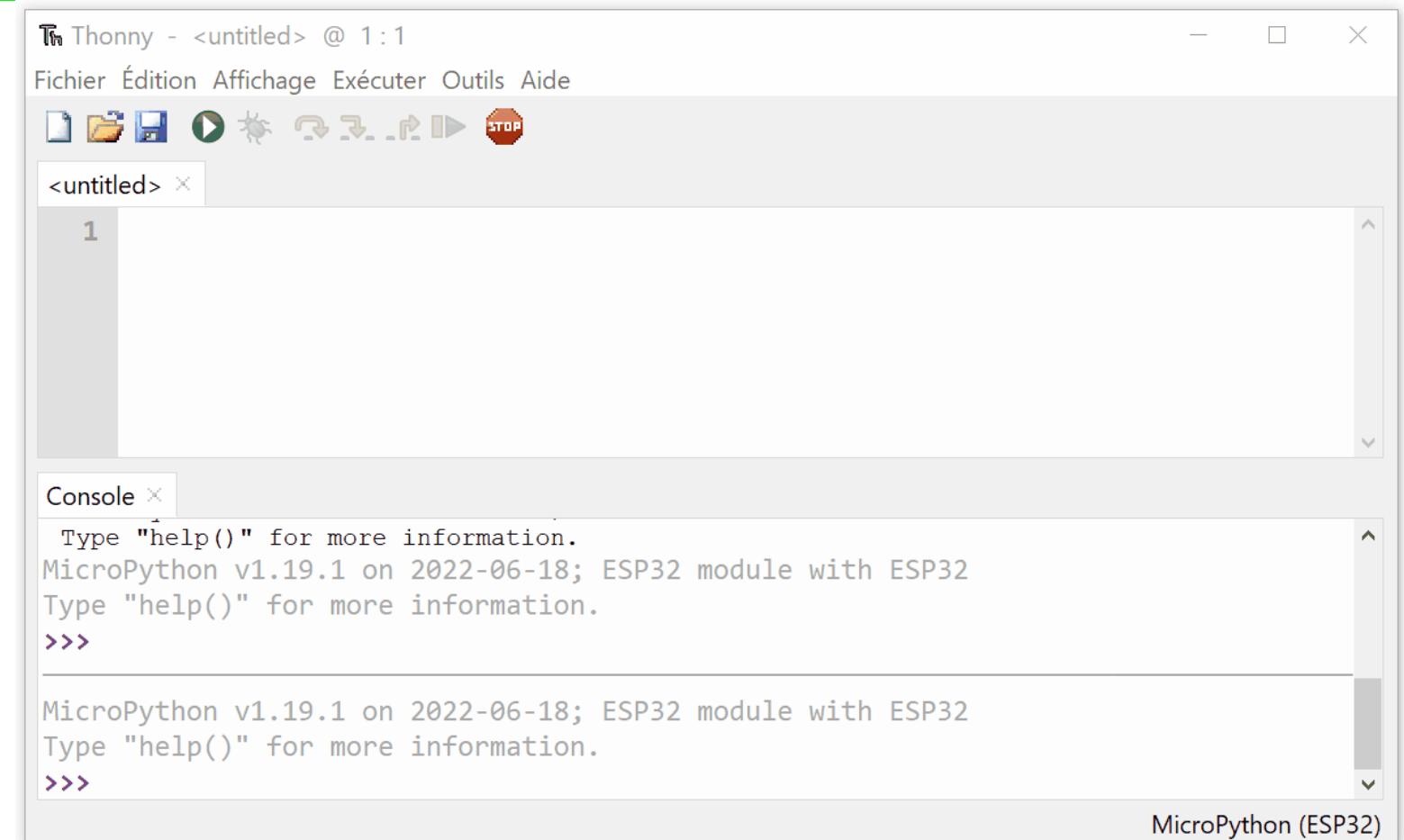
Section 3 : ESP32

Section 4 : PI Pico

☐ Installer MicroPython sur la carte ESP8266:

Flasher la carte avec Thonny IDE:

Si rien ne se passe, vous pouvez essayer d'appuyer sur le bouton STOP qui relance l'interpréteur.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Installer MicroPython sur la carte ESP8266:

Flasher la carte avec l'outil esptool:

Pour les utilisateurs avancés qui ne voudraient pas utiliser Thonny IDE, on peut utiliser directement l'outil de communication avec l'ESP8266 : **esptool.py**

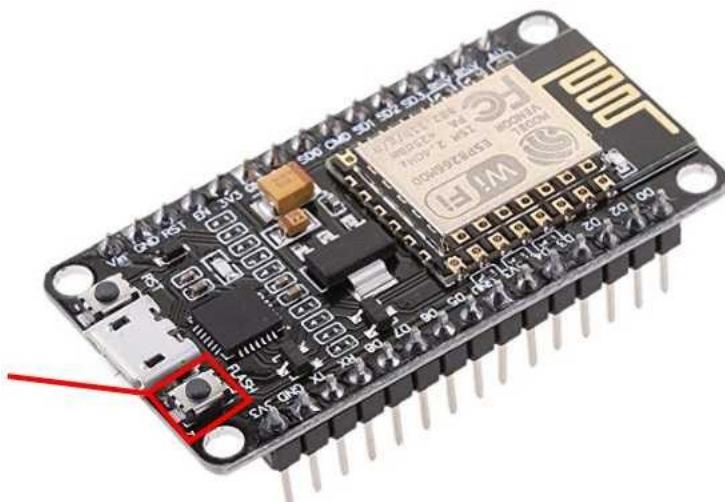
Vous pouvez l'installer via un terminal avec le gestionnaire de paquets *pip* :

pip install esptool

Avant de flasher le firmware MicroPython, vous devez effacer la mémoire flash ESP8266. Ainsi, avec votre ESP8266 connecté à votre ordinateur, maintenez enfoncé le « **BOTTE/FLASH** » bouton de votre carte ESP8266.

Tout en maintenant la touche « **BOTTE/FLASH** », exécutez la commande suivante pour effacer la mémoire flash de l'ESP8266 :

esptool.py --chip esp8266 erase_flash



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

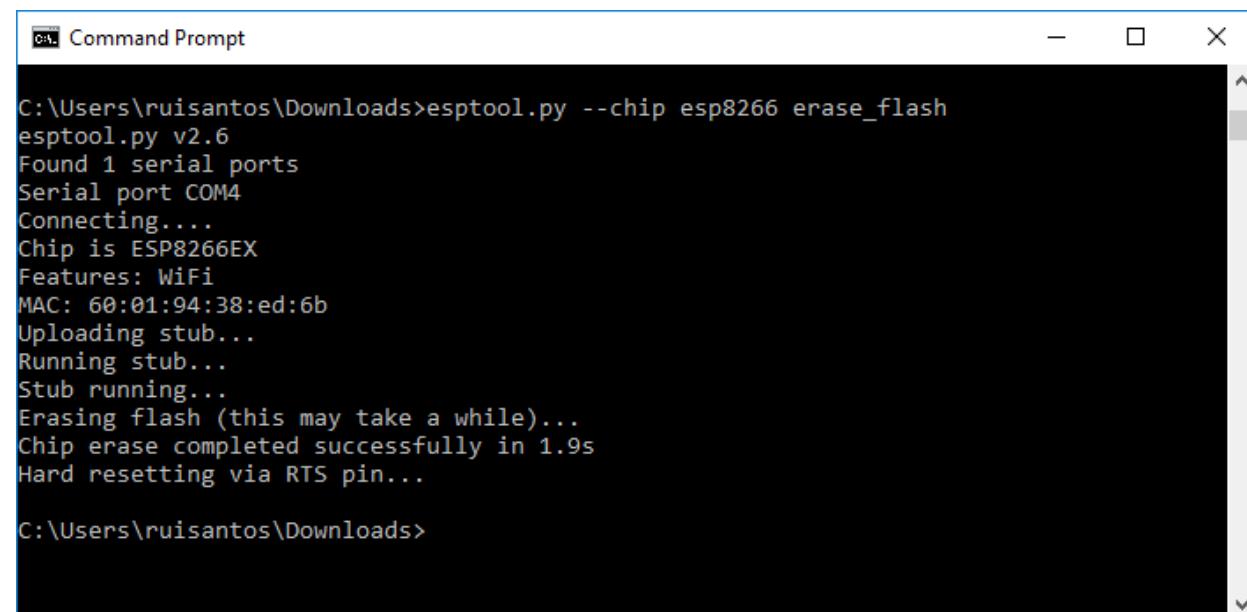
Section 3 : ESP32

Section 4 : PI Pico

□ Installer MicroPython sur la carte ESP8266:

Flasher la carte avec l'outil esptool:

Quand le « **Effacement** » le processus commence, vous pouvez relâcher le « **BOTTE/FLASH** » bouton. Après quelques secondes, la mémoire flash de l'ESP8266 sera effacée.



```
ca: Command Prompt
C:\Users\ruisantos\Downloads>esptool.py --chip esp8266 erase_flash
esptool.py v2.6
Found 1 serial ports
Serial port COM4
Connecting....
Chip is ESP8266EX
Features: WiFi
MAC: 60:01:94:38:ed:6b
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 1.9s
Hard resetting via RTS pin...

C:\Users\ruisantos\Downloads>
```

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Installer MicroPython sur la carte ESP8266:

Flasher la carte avec l'outil esptool:

Avec votre mémoire flash ESP8266 effacée, vous pouvez enfin flasher le firmware MicroPython. Vous avez besoin de votre nom de port série (COM4 dans notre cas) et de l'emplacement du fichier .bin.

Remplacez la commande suivante par vos coordonnées :

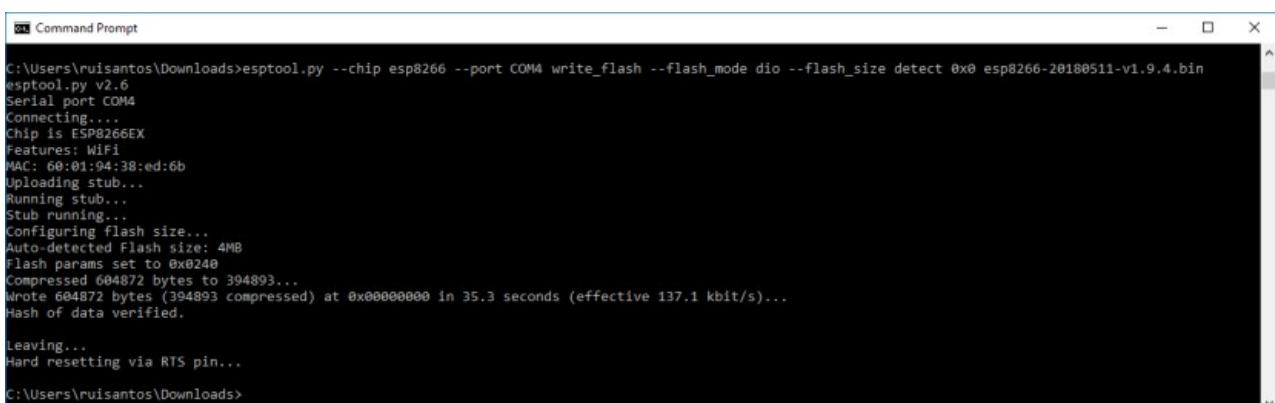
```
esptool.py --chip esp8266 --port <serial_port> write_flash --  
flash_mode dio --flash_size detect 0x0 <esp8266-X.bin>
```

Dans notre cas, la commande finale ressemble à ceci :

```
esptool.py --chip esp8266 --port COM4 write_flash --  
flash_mode dio --flash_size detect 0x0 esp8266-  
20180511-v1.9.4.bin
```

Maintenez la touche « BOTTE/FLASH », avant d'exécuter la commande flash.

Après quelques secondes, voici ce que vous devriez voir :



```
C:\Users\ruisantos\Downloads>esptool.py --chip esp8266 --port COM4 write_flash --flash_mode dio --flash_size detect 0x0 esp8266-20180511-v1.9.4.bin  
esptool.py v2.6  
Serial port COM4  
Connecting....  
Chip is ESP8266EX  
Features: Wifi  
MAC: 60:01:94:38:ed:6b  
Uploading stub...  
Running stub...  
Stub running...  
Configuring flash size...  
Auto-detected Flash size: 4MB  
Flash params set to 0x0240  
Compressed 604872 bytes to 394893...  
Wrote 604872 bytes (394893 compressed) at 0x00000000 in 35.3 seconds (effective 137.1 kbit/s)...  
Hash of data verified.  
  
Leaving...  
Hard resetting via RTS pin...  
C:\Users\ruisantos\Downloads>
```

Votre ESP8266 a été flashé avec succès avec le firmware MicroPython !

■ Programmer ESP8266 avec Micropython:

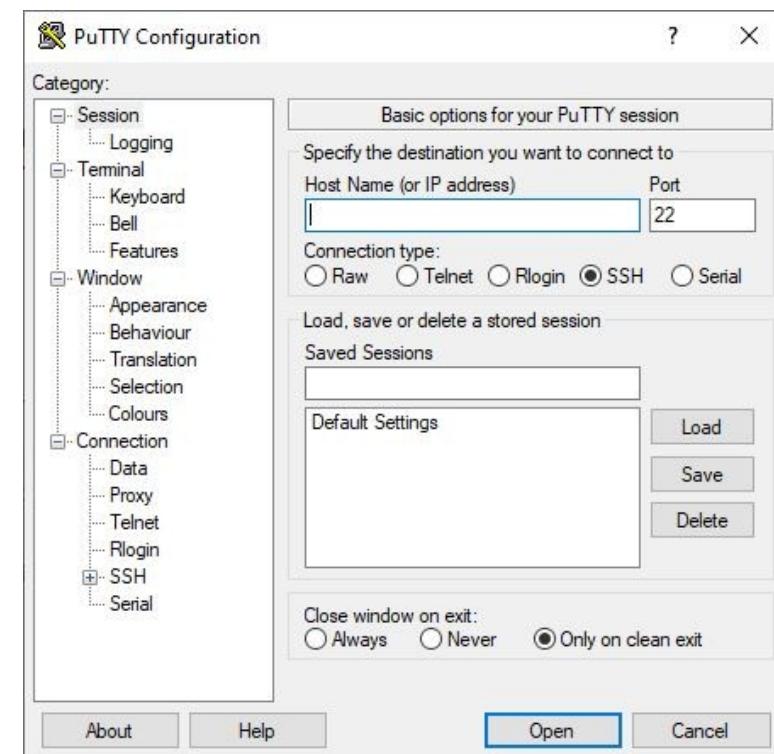
On peut programmer la carte ESP8266 par deux méthodes:

- En utilisant le **REPL** (Read Evaluate Print Loop)
- En utilisant un **IDE** (exemple Thonny IDE)

REPL signifie *Read Evaluate Print Loop*. Il s'agit du nom donné à l'invite interactive du MicroPython. Utiliser le REPL est le moyen le plus simple de tester du code simple et d'exécuter des commandes.

Pour pouvoir communiquer et programmer l'ESP8266 avec MicroPython, il nous faut installer un terminal: **PuTTY** ou **TeraTerm**.

Installer Putty puis sélectionner le port série correspondant au microcontrôleur



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

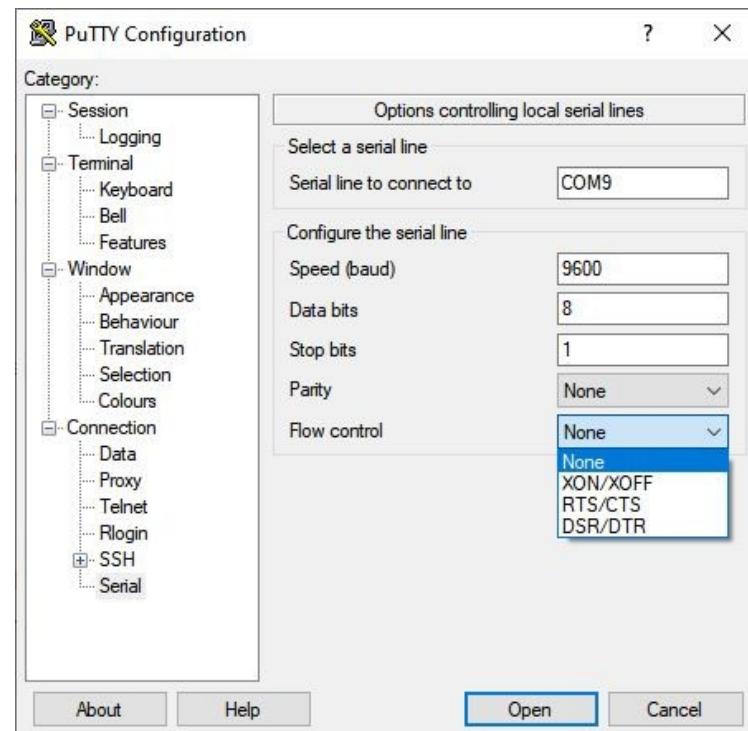
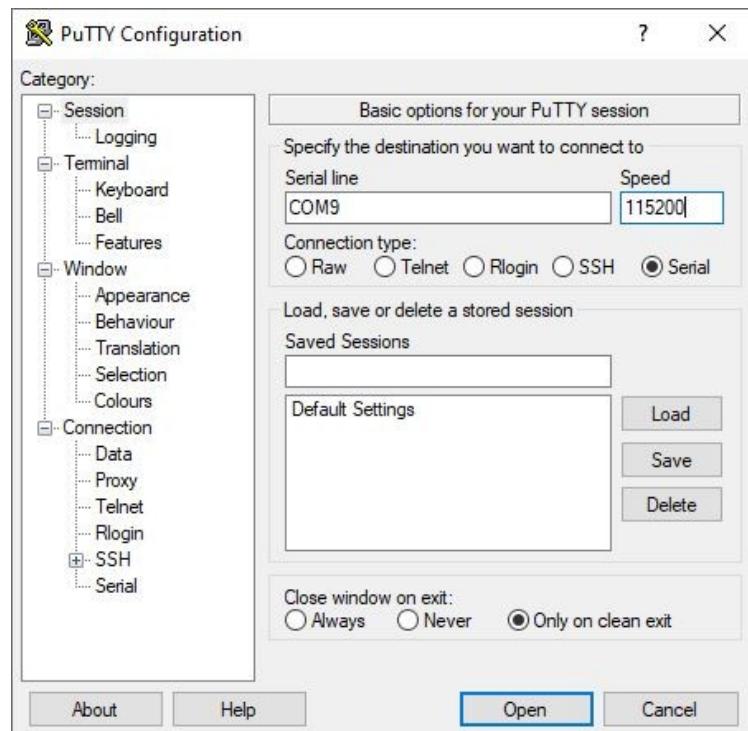
Section 3 : ESP32

Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython:

Programmer avec le REPL:

Configurer, ensuite, la communication série. Pour pouvoir communiquer avec l'ESP8266, il faut sélectionner None dans Flow control sous Serial



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

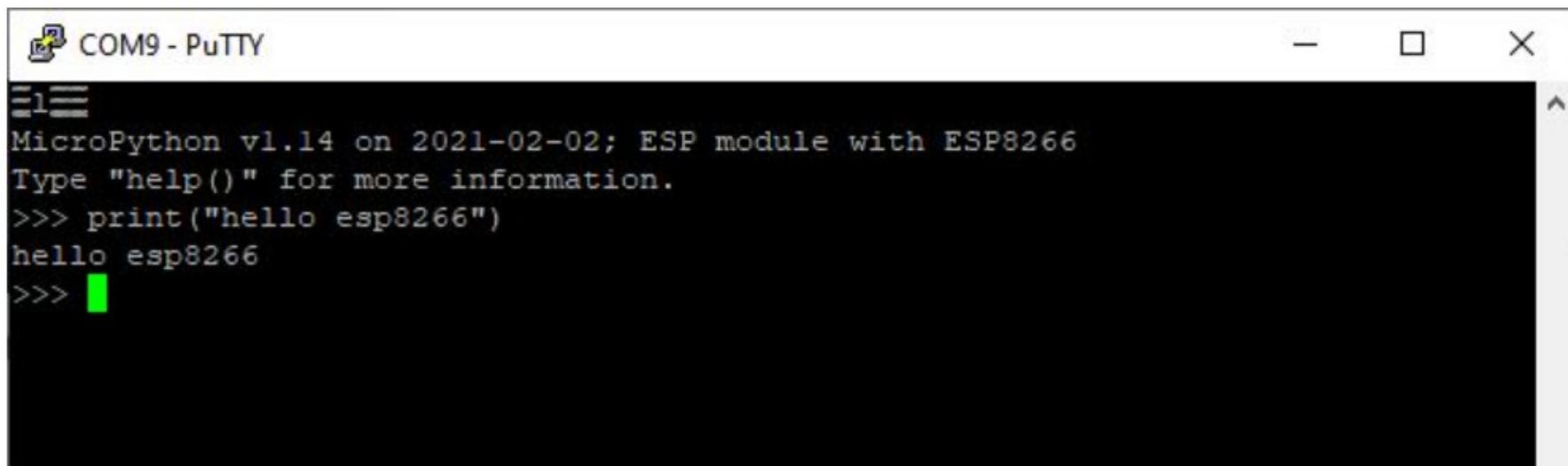
Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython:

Programmer avec le REPL:



The screenshot shows a terminal window titled "COM9 - PuTTY". The window contains the following text:

```
MicroPython v1.14 on 2021-02-02; ESP module with ESP8266
Type "help()" for more information.
>>> print("hello esp8266")
hello esp8266
>>> 
```

Exécutez les commandes suivantes en mode REPL:

```
>>> import sys
>>> print(sys.platform+ " " + sys.version)
esp8266 3.4.0
```

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

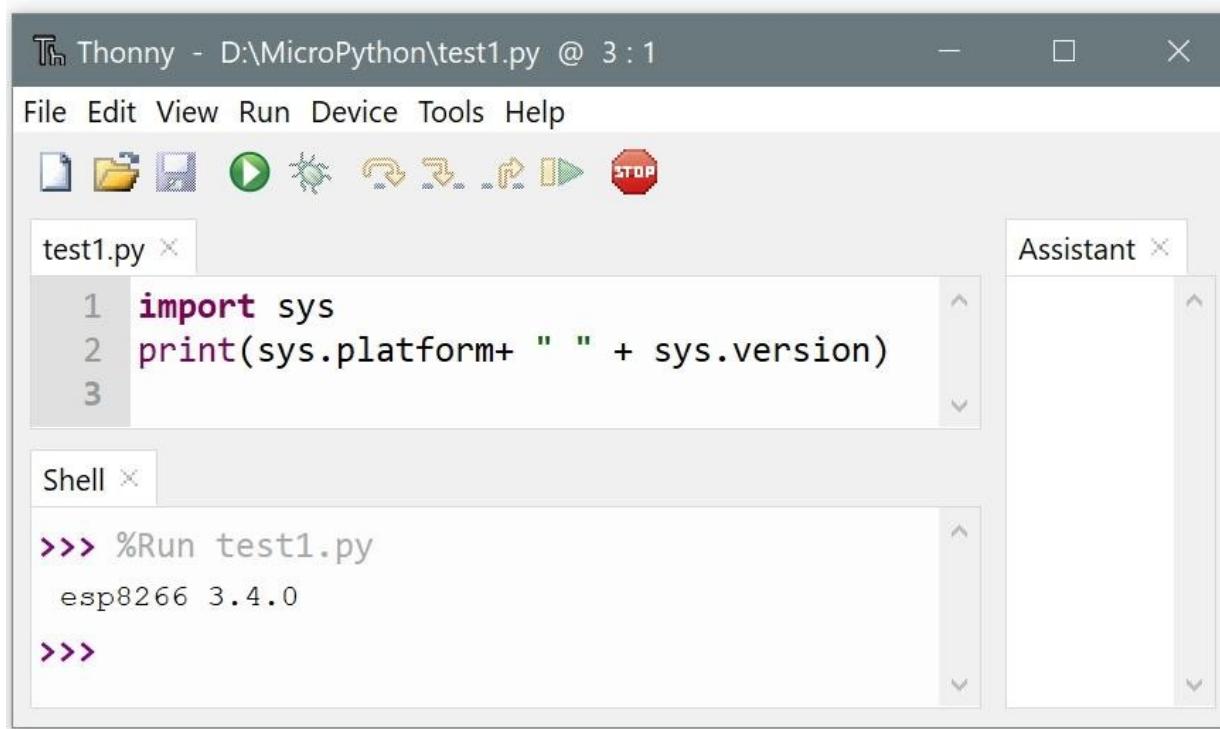
Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython:

Programmer avec Thonny IDE:



The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - D:\MicroPython\test1.py @ 3 : 1". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. Below the menu is a toolbar with icons for file operations and device connection. A code editor window titled "test1.py" contains the following Python code:

```
1 import sys
2 print(sys.platform+ " " + sys.version)
3
```

Below the code editor is a "Shell" window showing the output of the run command:

```
>>> %Run test1.py
esp8266 3.4.0
>>>
```

An "Assistant" window is also visible on the right side of the interface.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython:

Programmer avec Thonny IDE:

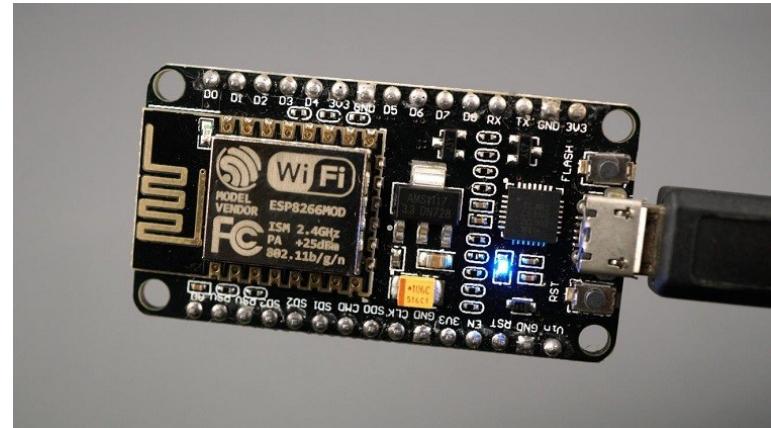
Exemple: LED clignotant sur la carte ESP8266 utilisant Thonny IDE:

Faire clignoter la LED incorporée dans la carte ESP8266 avec une période de 0.5 seconde.(sachant que le LED est branché sur le pin 2)

1. Commencer par créer un nouveau fichier: Fichier-> Nouveau ou Taper "Ctrl+N"

2.Saisir le code suivant :

code Micropython:



```
from machine import Pin  
import time  
  
led=Pin(2, Pin.OUT) #Définir le pin2 comme pin de sortie  
  
while True:  
    led.value(1) #allumer le LED  
    time.sleep(0.5)  
    led.value(0) #éteindre le LED  
    time.sleep(0.5)
```

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

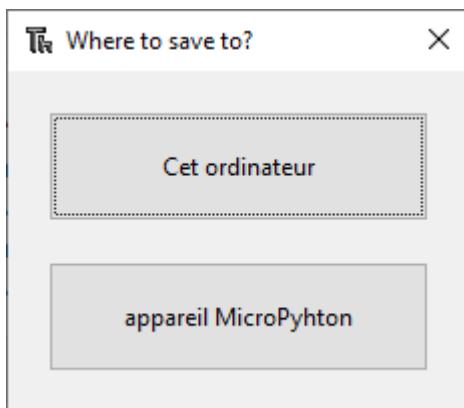
Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython:

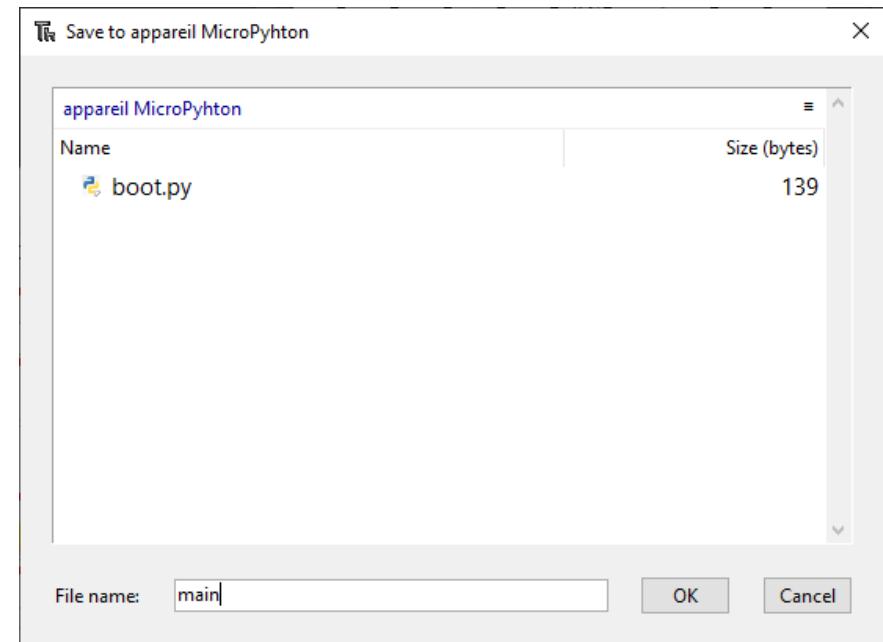
Programmer avec Thonny IDE:

Exemple: LED clignotant sur la carte ESP8266 utilisant Thonny IDE:

Enregistrer le fichier: fichier -> enregistrer sous -> Save ou cliquer sur le bouton "Save"



Choisir "Cet ordinateur" ou "appareil MicroPython" pour enregistrer directement dans la carte ESP8266



Choisir le nom **main** puis cliquer sur le bouton "Ok"

Remarque :

- Dans le dossier "appareil MicroPython", il y a par défaut "**boot.py**" qui se charge au démarrage de la carte ESP8266 et permet de définir les options de configuration.
- Il est recommandé d'écrire le code principal à exécuter dans un fichier "**main.py**" pour être exécuté immédiatement après "boot.py"

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

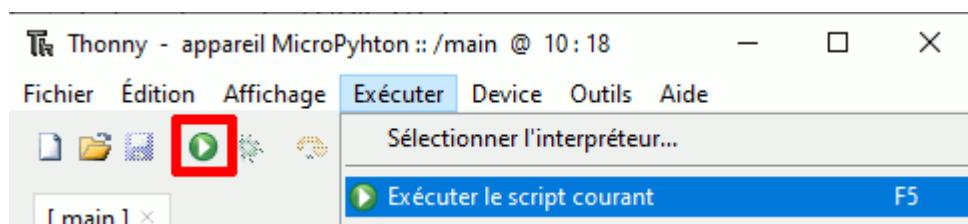
Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython:

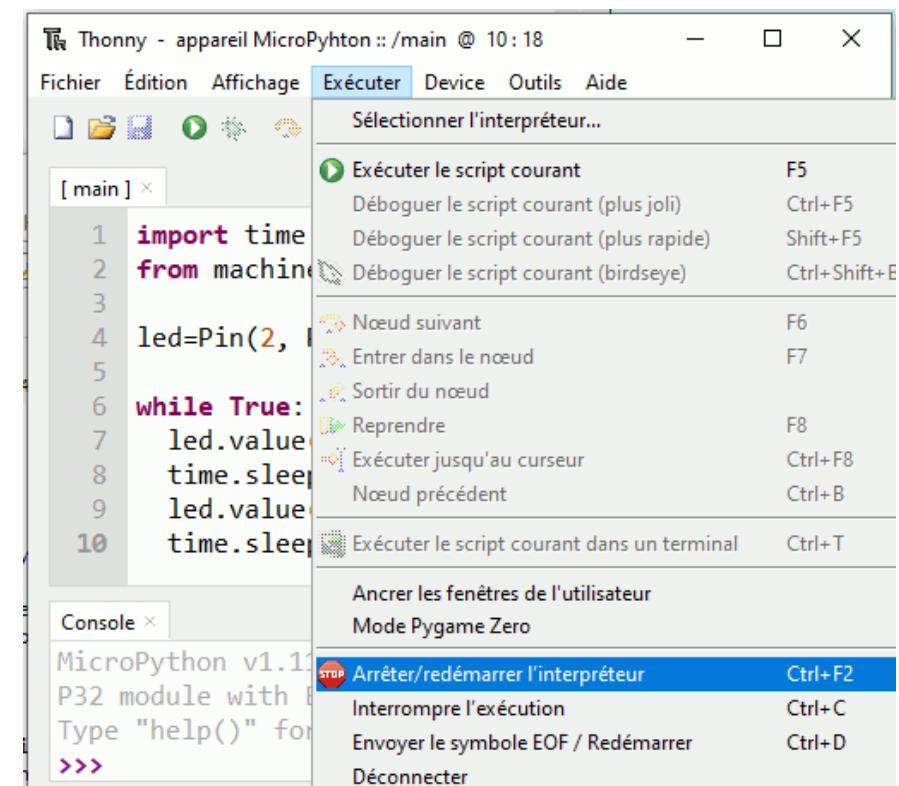
Programmer avec Thonny IDE:

Exemple: LED clignotant sur la carte ESP8266 utilisant Thonny IDE:

Pour lancer le code il suffit de cliquer sur le bouton " Exécuter le script courant" (F5)



Il faut arrêter l'exécution (Ctrl+F2) , avant de pouvoir enregistrer à nouveau sur la carte ESP8266.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython:

Erreurs..

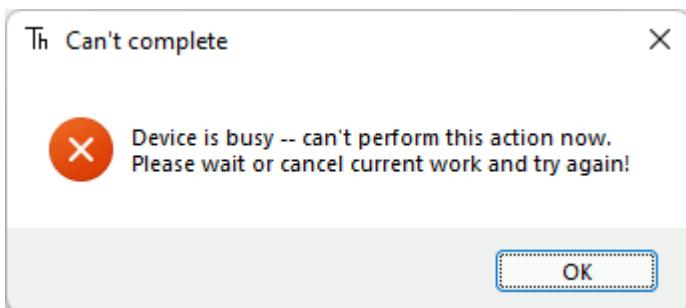
Exemple: LED clignotant sur la carte ESP8266 utilisant Thonny IDE:

Remarque 1:

Si cette erreur s'est produite lors de l'ouverture de "appareil MicoPython"

(Fichier -> ouvrir ->appareil MicoPython) :

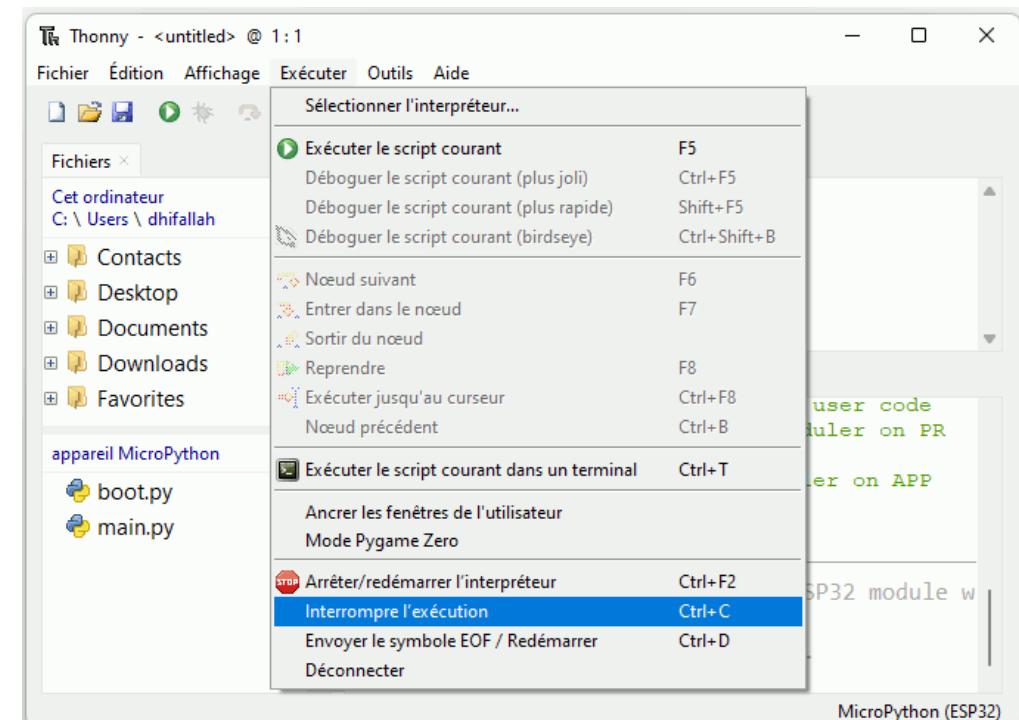
" Device is busy -- can't perform this action now. Please wait or cancel current work and try again!"



Procédez comme suivant :

-Menu "Exécuter" -> "**Arrêter/redémarrer l'interpréteur**"

Ctrl+F2 ensuite "**Interrompre l'exécution**" **Ctrl+C**



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython:

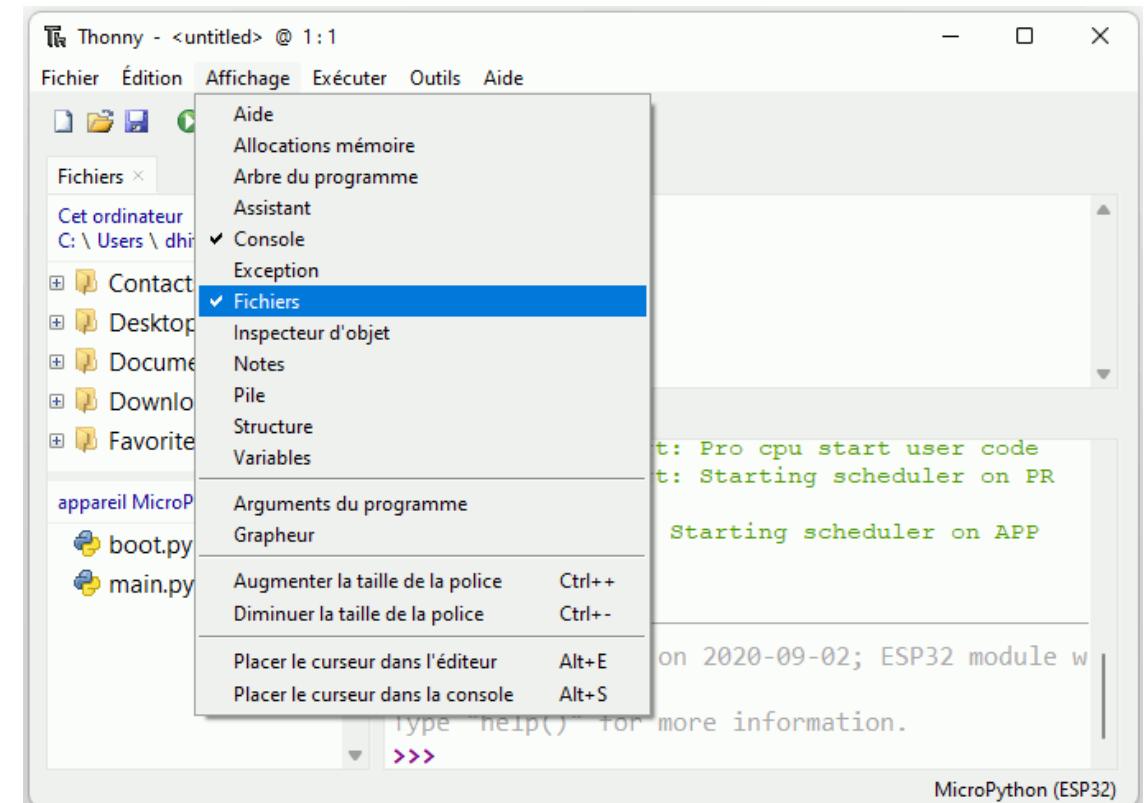
Affichage fichiers..

Exemple: LED clignotant sur la carte ESP8266 utilisant Thonny IDE:

Remarque 2:

On peut afficher les fichiers sur l'ordinateur ainsi que les fichiers sur la carte ESP8266 avec le menu :

"Affichage" -> "Fichiers" :



□ Programmer ESP8266 avec Micropython:

Applications à réaliser..

- LED Clignotante
- GPIO
- Push button
- Analog inputs (ADC)
- PWM
- Stepper motor
- Servo motor
- Interrupts Timers
- Deep sleep
- Capteur à ultrasons
- Capteurs Température / humidité / pression
- Les relais
- RFID
- Web server
- MQTT
- WIFI
- Etc..

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

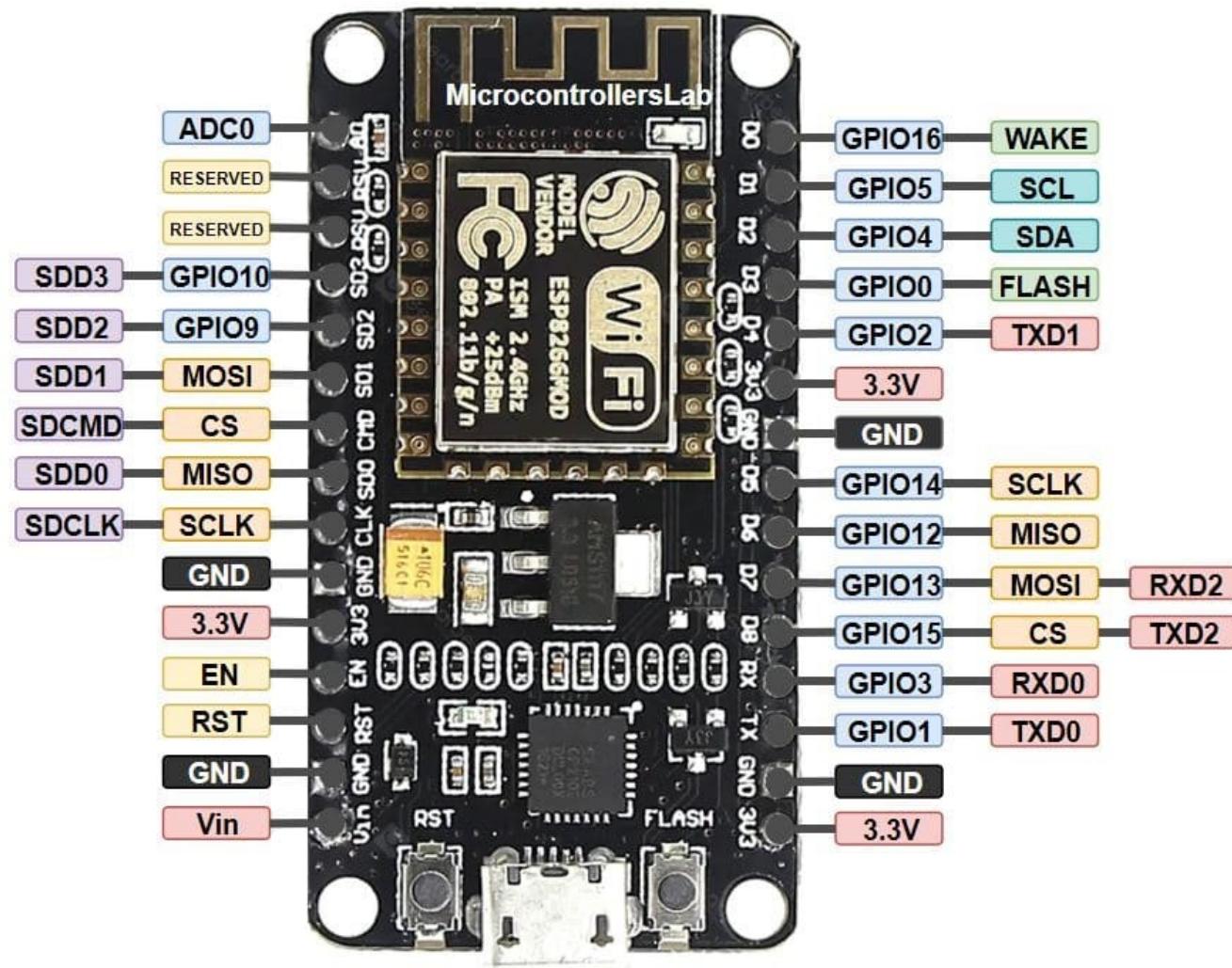
Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython: Les **GPIOs**

Qu'est-ce qu'un GPIO ?

GPIO est l'abréviation de General Purpose Input/Output.

Il s'agit de broches de signal **numérique** situées sur le module qui peuvent être utilisées comme entrée, sortie et même les deux peuvent être utilisées ensemble.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

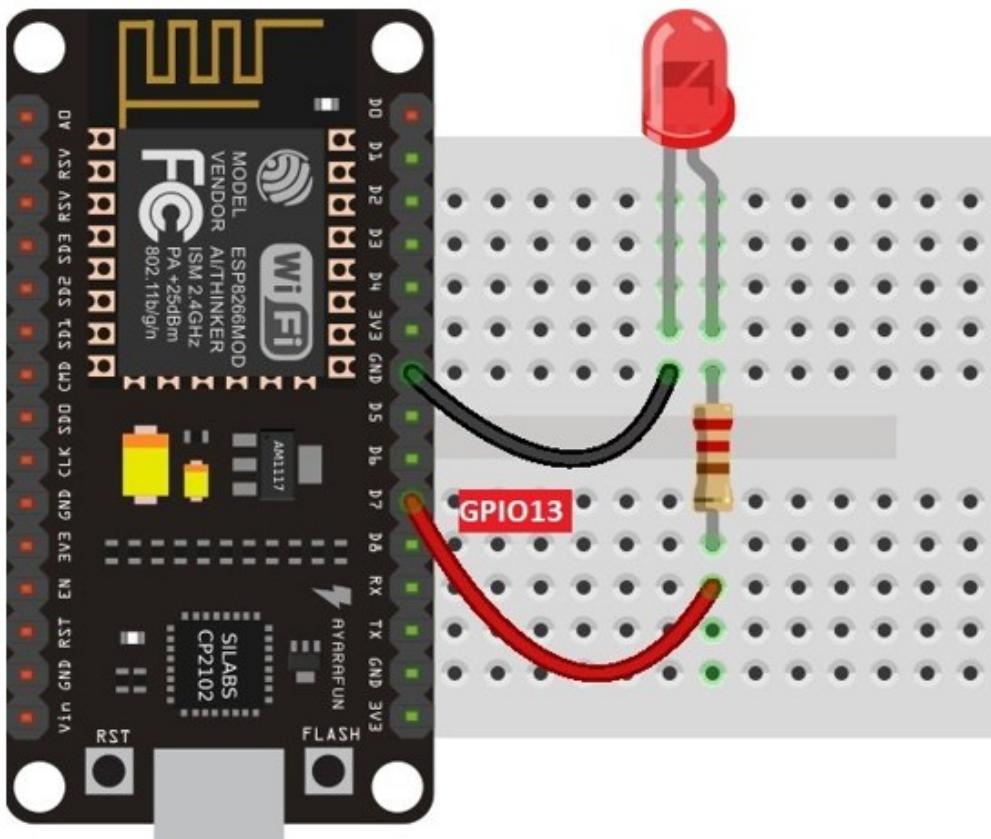
□ Programmer ESP8266 avec Micropython: Les **GPIOs**

Exercice 1: Entrée / sortie numérique

Faire clignoter une LED connectée sur le GPIO 5 en appuyant sur un bouton branché sur le GPIO 4.
Afficher l'état du bouton.

Nous avons besoin des équipements suivants :

- 1.Plaque a essai
- 2.Une LED
- 3.Une résistance de 220 ohms
- 4.Fils de connexion
- 5.Carte ESP 8266



```
import time
from machine import Pin
led=Pin(13,Pin.OUT)

while True:
    led.value(1)
    time.sleep(0.5)
    led.value(0)
    time.sleep(0.5)
```

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython: Les **GPIOs**

Exercice 1: Entrée / sortie numérique

Comment fonctionne le code:

En MicroPython, un module **machine** contient des classes de différents périphériques de ESP8266 tels que GPIO, ADC, PWM, I2C, SPI, etc. Ici, nous utilisons les broches **GPIO** de l'ESP8266. Par conséquent, nous allons importer la classe **GPIO** à partir d'un module **machine**. Nous importons ici la classe **Pin** à partir du module machine.

Nous créons une instance de l'objet Pin « led ». On peut donner n'importe quel nom à un objet tel que led13...

La boucle **while True:** s'agit d'une boucle infinie qui s'exécute à l'infini

Pin contient une fonction connue sous le nom de **value()** qui est utilisée pour définir l'état des broches GPIO. Passer 1 à cette fonction mettra la broche GPIO à un état élevé actif (**led.value(1)**).

De même **time.sleep(0.5)** //Delay of 0.5 seconds mettra la broche GPIO sur un état bas actif (**led.value(0)**),

```
import time
from machine import Pin
led=Pin(13,Pin.OUT)

while True:
    led.value(1)
    time.sleep(0.5)
    led.value(0)
    time.sleep(0.5)
```

```
import time
from machine import Pin
led=Pin(13,Pin.OUT)

while True:
    led.on()          # Turn LED on
    time.sleep(0.5)
    led.off()         # Turn LED off
    time.sleep(0.5)
```

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

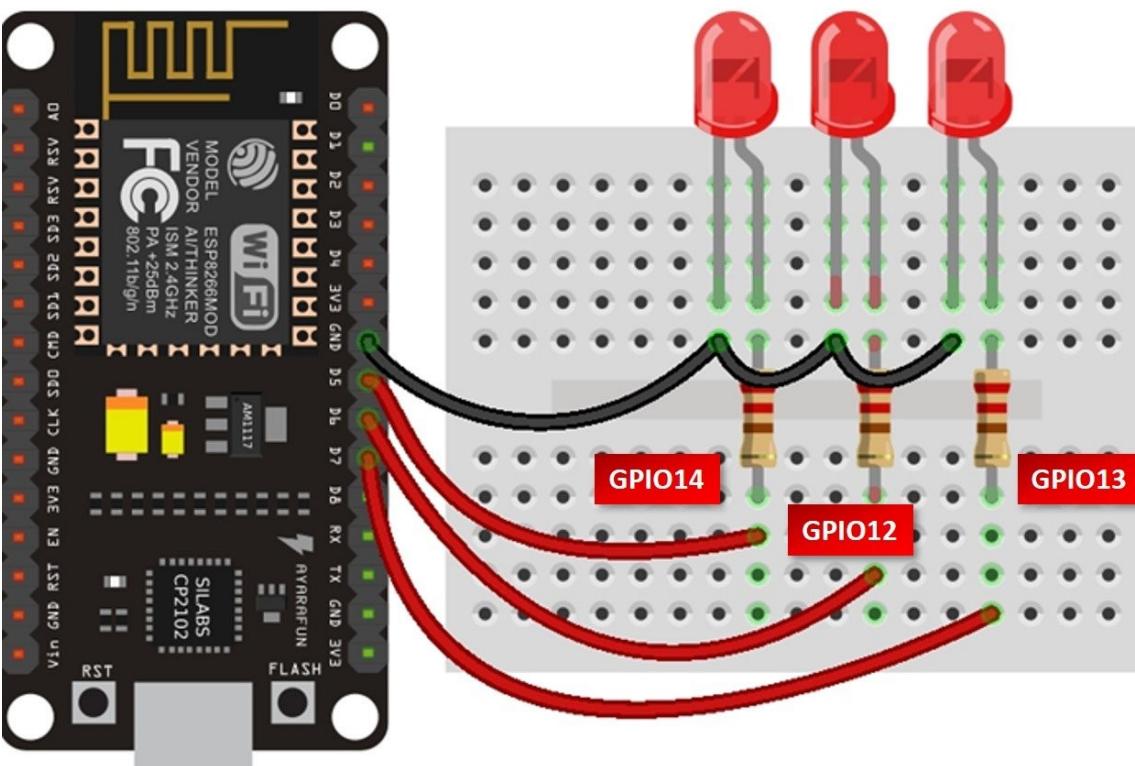
□ Programmer ESP8266 avec Micropython: Les GPIOs

Exercice 2: Entrée / sortie numérique

Faire clignoter trois
LEDs successivement.

Matériel:

1. Plaque à essai
2. Trois LED
3. Trois résistances de 220 ohms
4. Fils de connexion
5. Carte ESP 8266



```
from machine import Pin  
from time import sleep
```

```
led_13 = Pin(13, Pin.OUT)  
led_14 = Pin(14, Pin.OUT)  
led_12 = Pin(12, Pin.OUT)
```

```
while True:  
    led_12.value(0)  
    led_13.value(1)  
    led_14.value(0)  
    sleep(0.5)  
    led_12.value(1)  
    led_13.value(0)  
    led_14.value(0)  
    sleep(0.5)  
    led_12.value(0)  
    led_13.value(0)  
    led_14.value(1)  
    sleep(0.5)
```

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython: Les **GPIOs**

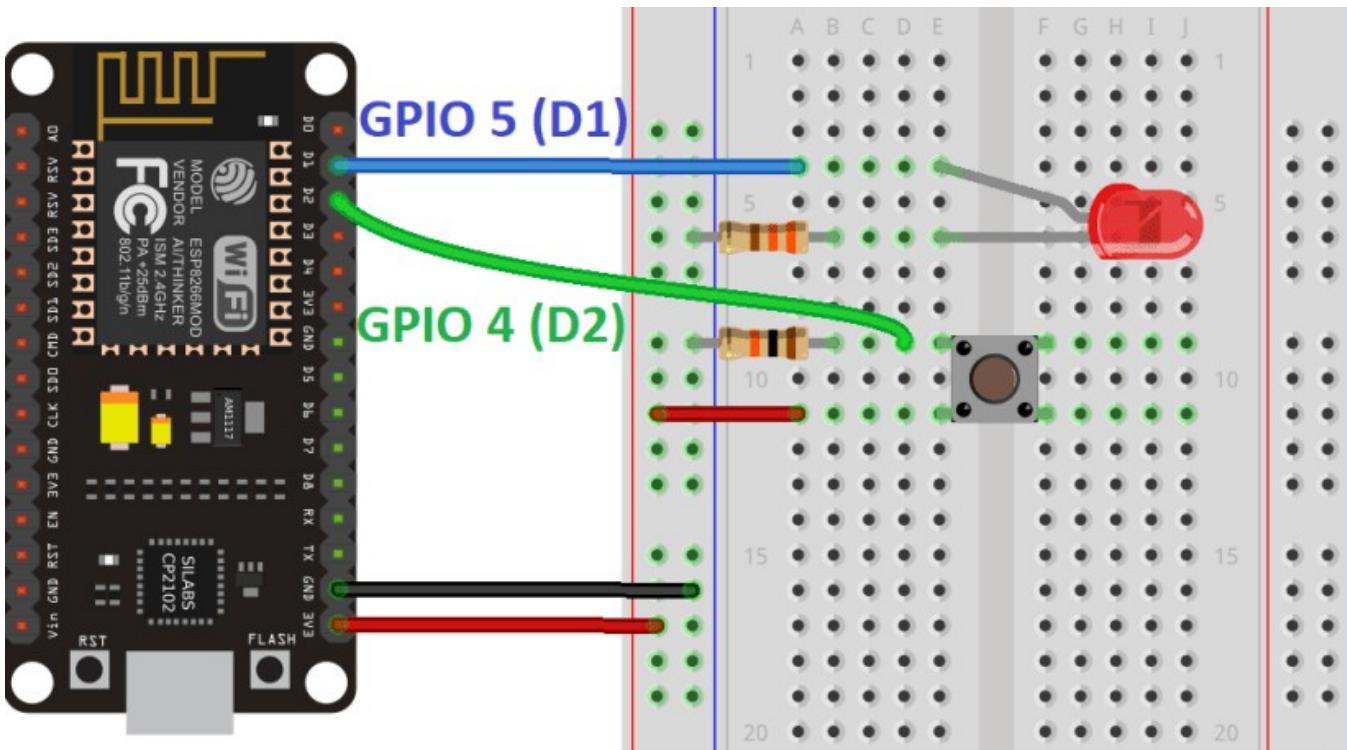
Exercice 3: Push button

Faire clignoter une LED connectée sur le GPIO 5 en appuyant sur un bouton branché sur le GPIO 4.

Afficher l'état du bouton.

Matériel:

- 1.Plaque a essai
- 2.LED
- 3.Un bouton poussoir
- 4.Deux résistances de 330 et 10K ohms
- 5.Fils de connexion
- 6.Carte ESP 8266



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

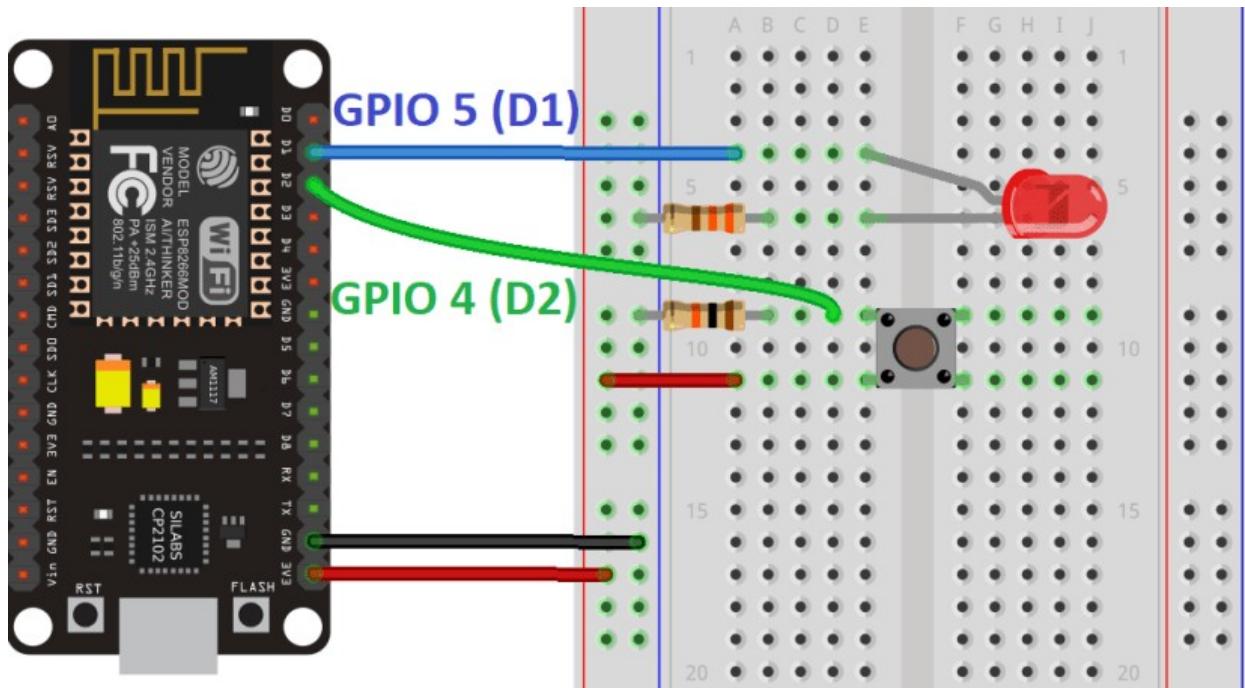
Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython: Les GPIOs

Exercice 3: Push button

```
from machine import Pin
from time import sleep
led = Pin(5, Pin.OUT)          # 5 number pin is Output
push_button = Pin(4, Pin.IN)    # 4 number pin is input
while True:
    logic_state = push_button.value()
    if logic_state == True:      # if pressed the push_button
        led.value(1)            # led will turn ON
    else:                      # if push_button not pressed
        led.value(0)            # led will turn OFF
```



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

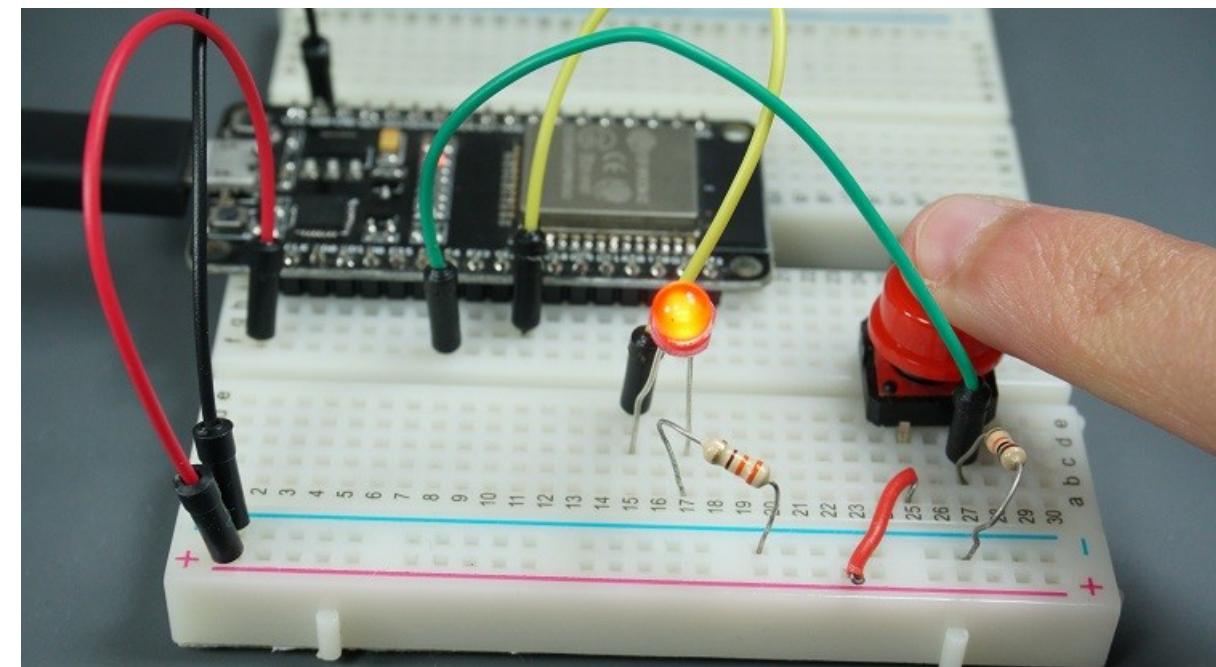
Section 3 : ESP32

Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython: Les **GPIOs**

Exercice 3: Push button

```
from machine import Pin
from time import sleep
led = Pin(5, Pin.OUT)          # 5 number pin is Output
push_button = Pin(4, Pin.IN)    # 4 number pin is input
while True:
    led.value(push_button.value())
    sleep(0.5)
```

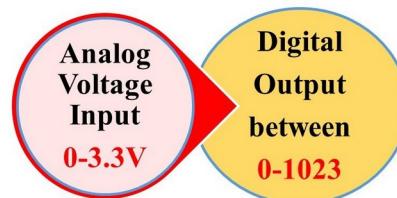
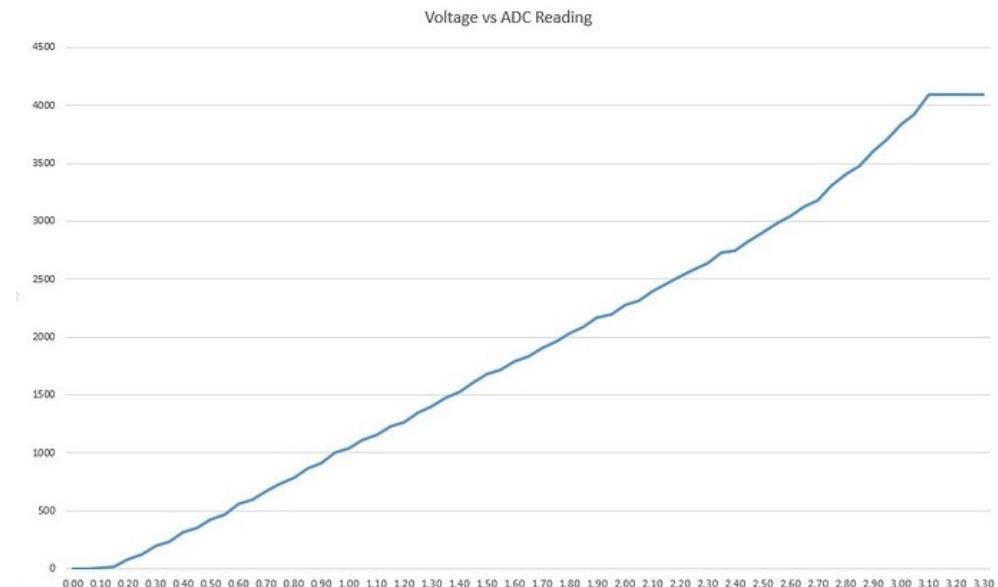


❑ Programmer ESP8266 avec Micropython: Analog Inputs ADC

Exercice 4: Mesure d'une entrée analogique

Pour l' Esp8266, les valeurs analogiques sont lues à travers des valeurs de tension variables entre 0 et 3,3 V. La tension que nous obtenons se voit alors attribuer une valeur comprise entre 0-1023. Cela signifie que la valeur maximale 1023 correspond à un 3,3 V et qu'une valeur de 0 correspond à 0 V. Toutes les valeurs correspondantes intermédiaires seront également attribuées en conséquence.

Cependant, il y a un léger inconvénient. Le module ADC n'est pas extrêmement sensible aux changements de valeurs. Il est de nature non linéaire. Ainsi, des valeurs très proches correspondront à des tensions similaires. Par exemple, 3.3V et 3.2V correspondent tous deux à la valeur de 1023. Le graphique suivant montre la relation non linéaire entre la valeur de la tension et la lecture de l'ADC.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

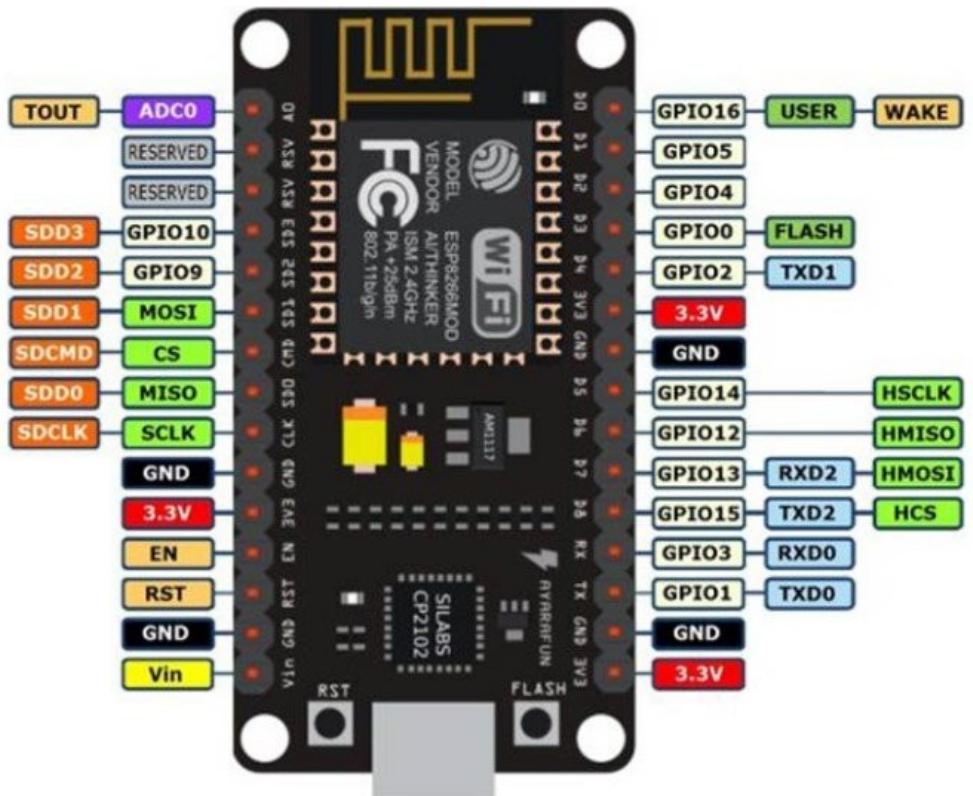
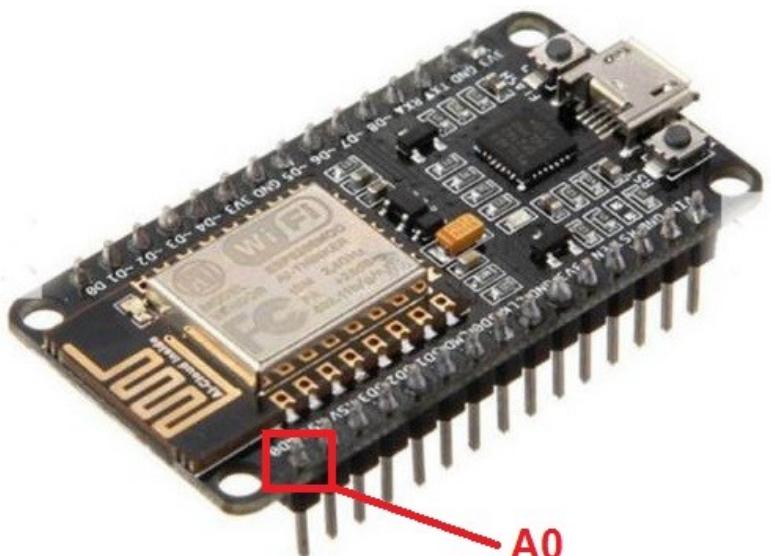
Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython: Analog Inputs ADC

Exercice 4: Mesure d'une entrée analogique

ESP8266 dispose d'une seule broche analogique, à savoir ADC0, qui est généralement imprimée A0 sur la carte.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

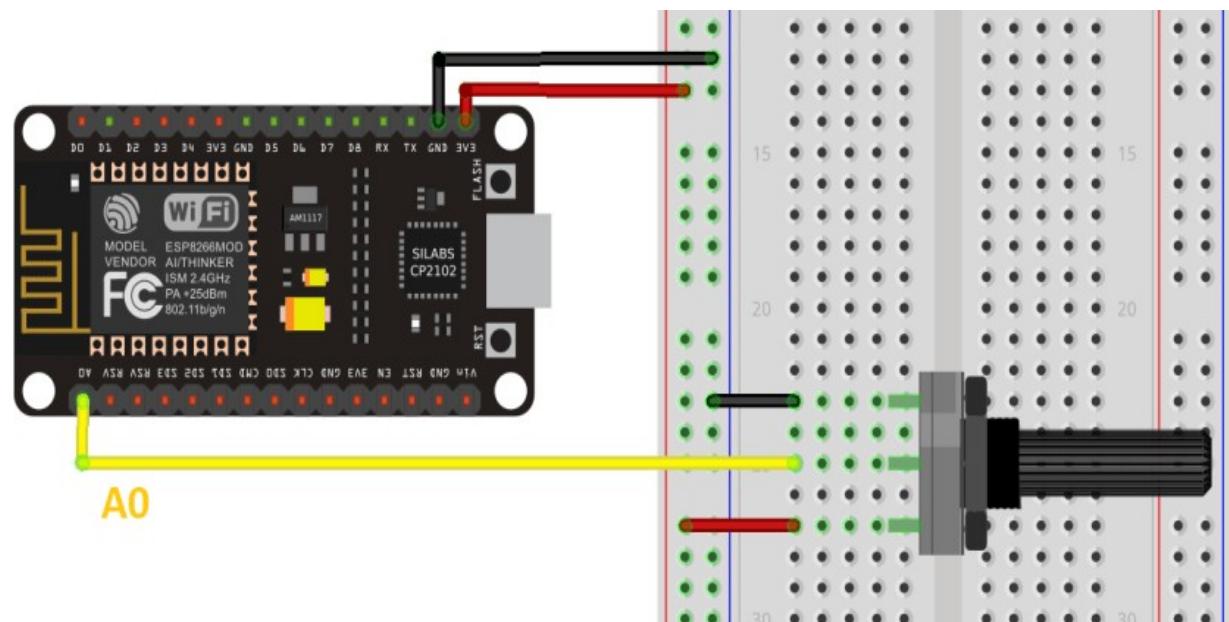
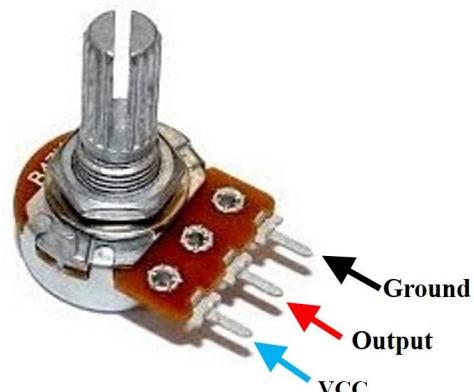
Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython: Analog Inputs ADC

Exercice 4: Mesure d'une entrée analogique

Nous allons maintenant apprendre à lire les entrées analogiques de l'ESP8266 à l'aide d'une résistance variable, c'est-à-dire d'un potentiomètre. Le schéma suivant montre les connexions du potentiomètre. La borne centrale donne la sortie et les deux bornes sur les côtés sont fournies respectivement pour la connexion à la terre (Ground) et à la tension de alimentation (VCC).

Connections of Potentiometer



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

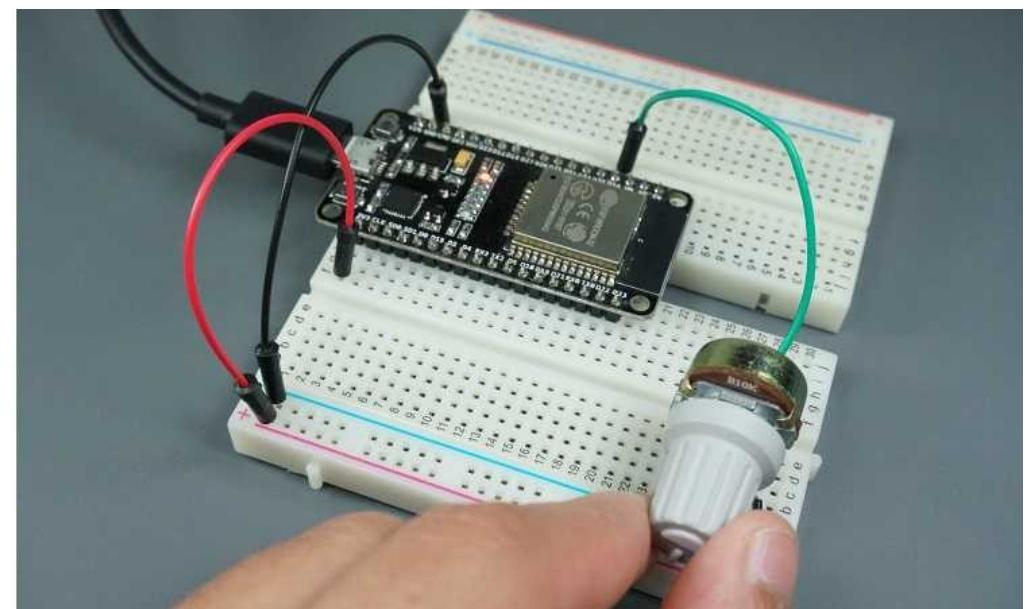
Section 3 : ESP32

Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython: Analog Inputs ADC

Exercice 4: Mesure d'une entrée analogique

```
from machine import Pin, ADC
from time import sleep
potentiometer = ADC(0)
while True:
    potentiometer_value = potentiometer.read()      #reading analog pin
    print(potentiometer_value)                      #printing the ADC value
    sleep(0.25)
```

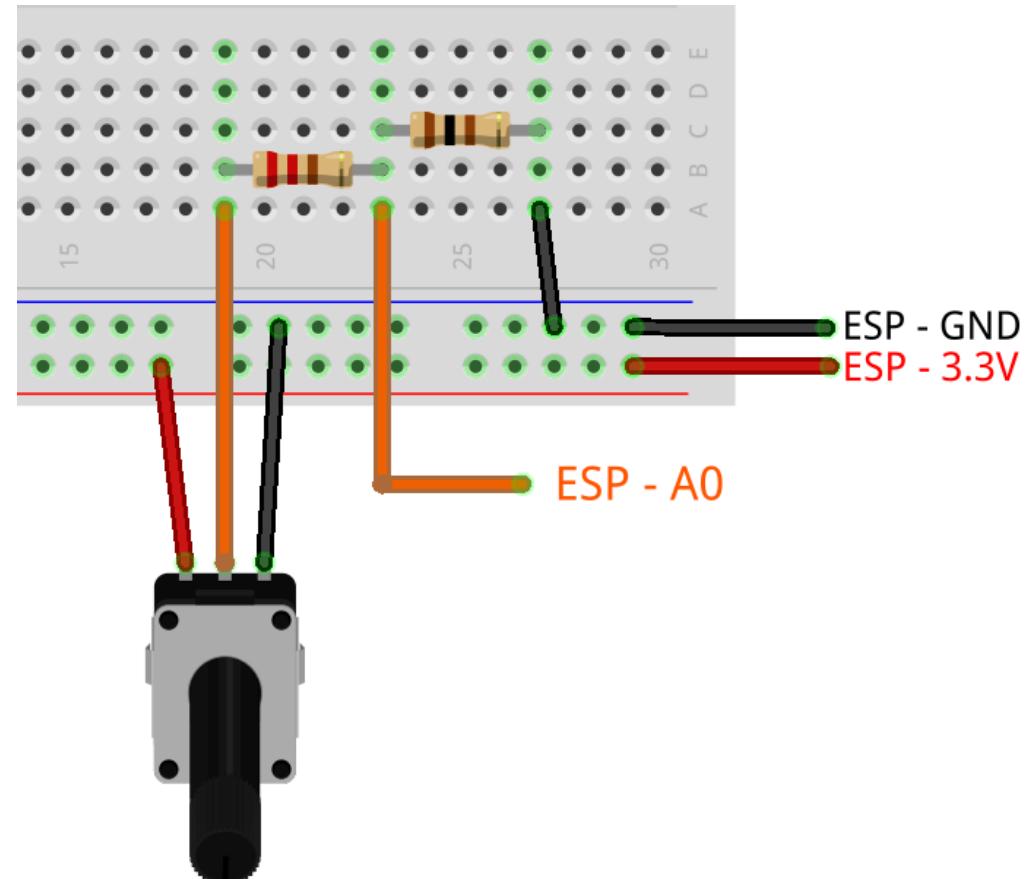


Programmer ESP8266 avec Micropython: Analog Inputs ADC

Exercice 4: Mesure d'une entrée analogique

Si vous utilisez une puce ESP8266 avec une plage de tension d'entrée de 0 V à 1 V, (Exemple ESP-01), vous devez vous assurer que la tension d'entrée sur la broche A0 ne dépasse pas 1 V. Vous avez donc besoin d'un circuit diviseur de tension, comme indiqué ci-dessous.

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$



❑ Programmer ESP8266 avec Micropython: PWM

Exercice 5: PWM

Nous allons apprendre à accéder aux modules **PWM** dans ESP8266 à l'aide du firmware MicroPython. Nous verrons comment produire les signaux PWM avec les broches GPIO des cartes de développement ESP et par conséquent produire différents niveaux de luminosité de LED.

Le **PWM**, acronyme de Pulse Width modulation, signifie modulation de largeur d'impulsion. Il s'agit d'un type de signal qui est obtenu à partir de notre ESP8266. Le signal de sortie est une forme d'onde carrée. C'est une technique qui permet de générer une tension comprise entre 0 et 3,3V en utilisant **uniquement des sorties numériques**. En effet, cette astuce est basée sur la proportion temporelle d'un signal logique à son état haut (3.3V) et à son état bas (0V) : le **PWM** consiste à faire varier la **largeur d'une impulsion** électrique.

En pratique, le PWM est utilisé pour : Contrôle de la vitesse d'un moteur - Contrôle de la luminosité des LED - Générer des signaux carrés (avec rapport cyclique $\alpha=0,5$) - Générer des notes de musique, etc..

□ Programmer ESP8266 avec Micropython: PWM

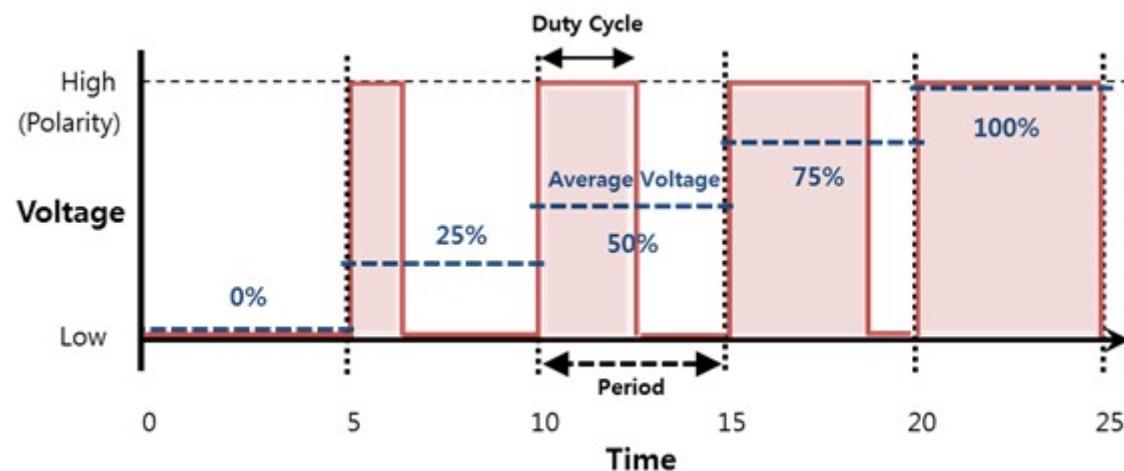
Exercice 5: PWM

Le “ON-Time” est la durée jusqu’à laquelle le signal reste élevé (high) et le “OFF-Time” est la durée jusqu’à laquelle il reste faible (low). Afin de mieux comprendre les concepts de PWM dans notre carte ESP, nous devons connaître les deux termes suivants qui sont étroitement associés au signal numérique :

1. le rapport cyclique: Duty cycle α

Le rapport cyclique est le pourcentage de temps pendant lequel le signal PWM est «ON», ce qui signifie qu'il reste élevé. Par exemple, si le signal est toujours OFF, il s'agit d'un facteur de 0 % et s'il est toujours ON, il s'agit d'un facteur de 100 %. La formule du rapport cyclique est illustrée dans l'expression suivante :

$$\text{Rapport cyclique} = \text{ON-Time of signal} / \text{Période de temps du signal}$$



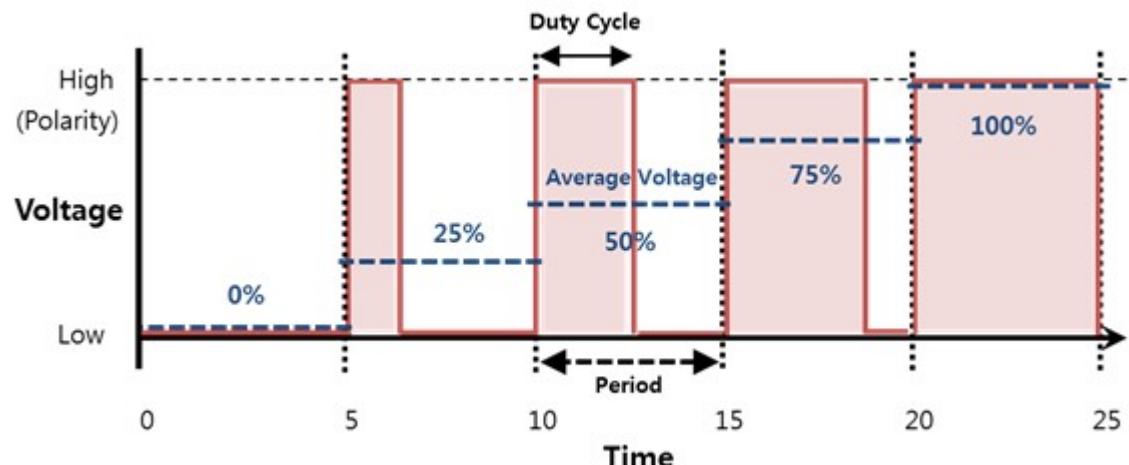
□ Programmer ESP8266 avec Micropython: PWM

Exercice 5: PWM

Le “ON-Time” est la durée jusqu’à laquelle le signal reste élevé (high) et le “OFF-Time” est la durée jusqu’à laquelle il reste faible (low). Afin de mieux comprendre les concepts de PWM dans notre carte ESP nous devons connaître les deux termes suivants qui sont étroitement associés au signal numérique :

2. La fréquence

La fréquence d'un signal PWM est le nombre de cycles par seconde et définit la vitesse à laquelle un signal PWM effectue un cycle (période). Cela signifie que la période d'un signal PWM est la somme des temps d'activation et d'arrêt d'un signal PWM. Par exemple, si la période de temps d'un signal est de 20 ms, sa fréquence sera de 50 Hz. Cette formule est utilisée pour calculer la fréquence :



$$\text{Frequency} = 1/\text{Time Period}$$

$$\text{Time Period} = \text{ON time} + \text{OFF time}$$

$$V_{out} = V_{in} * \alpha$$

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

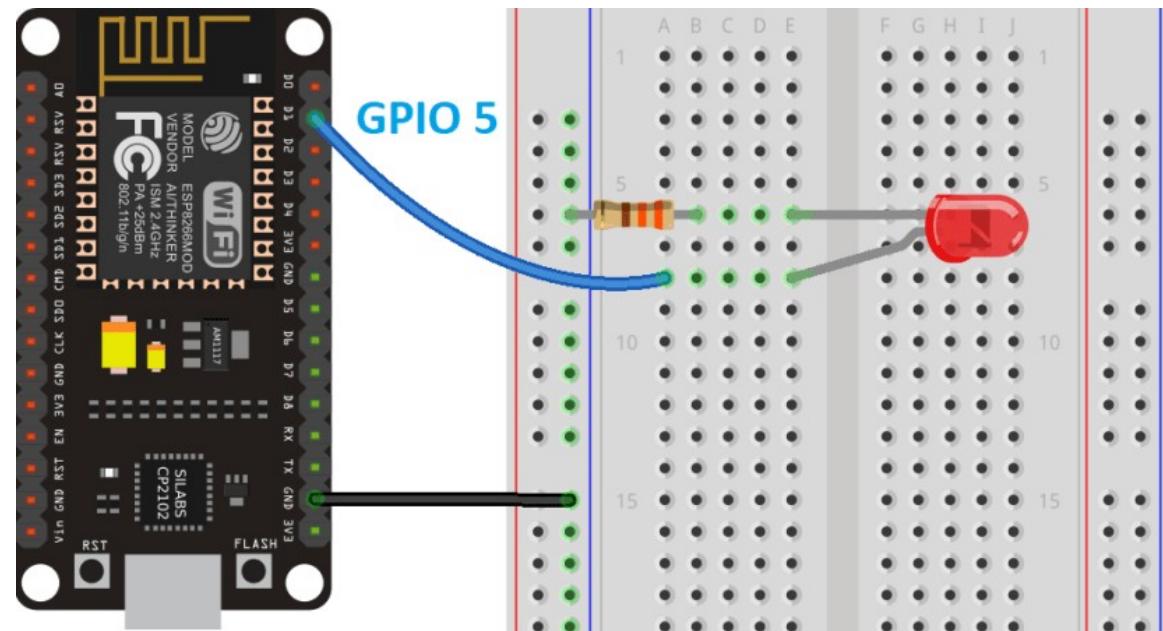
Section 3 : ESP32

Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython: PWM

Exercice 5: PWM

```
from machine import Pin, PWM  
from time import sleep  
  
freq= 5000  
  
led = PWM(Pin(5), freq)  
  
while True:  
    for duty_cycle in range(0, 1024):  
        led.duty(duty_cycle)  
        sleep(0.005)
```



❑ Programmer ESP8266 avec Micropython: PWM

Exercice 5: PWM

Explication du code: Afin de créer une broche **PWM** pour lire les GPIO d'entrée/sortie, nous allons également importer la classe **PWM** depuis le module **machine** de MicroPython. Nous allons importer le module **machine** qui contient des classes pour interagir avec les **GPIO**. Le module **time** est également importé pour être utilisé dans les retards.

Ensuite, nous créons un objet PWM appelé **led** pour passer des paramètres. Le premier paramètre indique où la broche est connectée dans notre cas **GPIO5** et le paramètre suivant comme la fréquence **5000Hz**.

Comme nous n'avons pas passé le rapport cyclique comme troisième paramètre, nous allons l'incorporer dans notre boucle **while**. Il est défini sur 0% par défaut. Nous utiliserons la méthode **duty()** sur l'objet PWM pour passer notre rapport cyclique en argument.

À l'intérieur de la boucle infinie, nous allons générer une boucle **'for'** qui augmente le rapport cyclique de **1** après un intervalle de **5 ms**.

Au cours de chaque itération de la boucle for, la variable **duty_cycle** est incrémentée de 1. La modification du rapport cyclique est la façon dont vous produisez différents niveaux de luminosité.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

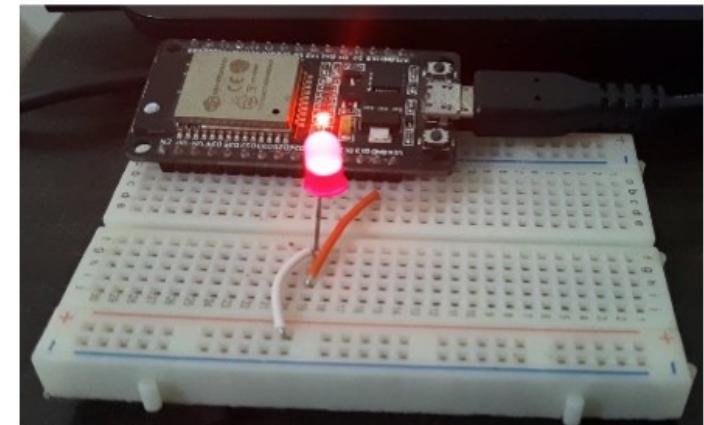
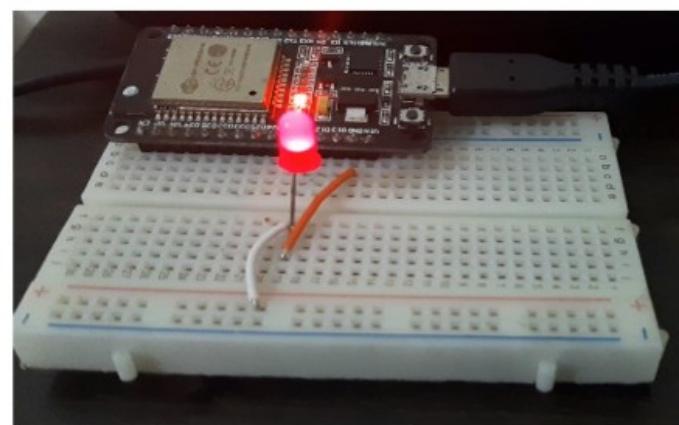
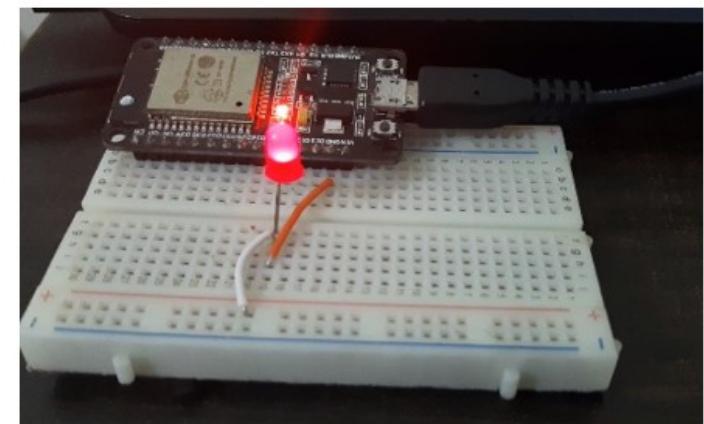
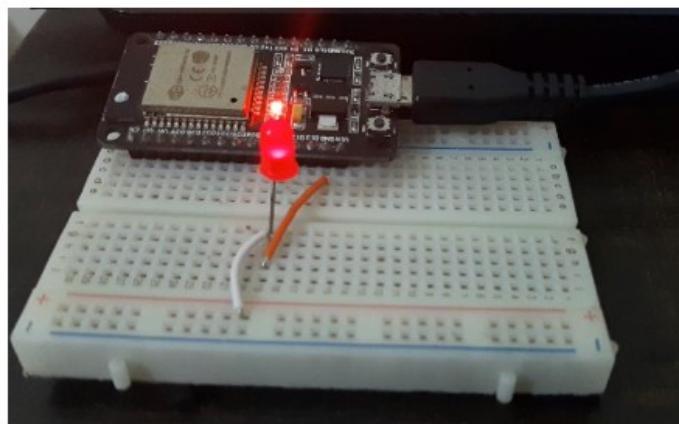
Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython: PWM

Exercice 5: PWM

Remarque: assigner une valeur avec duty_u16
(sur une résolution de 16 bits)

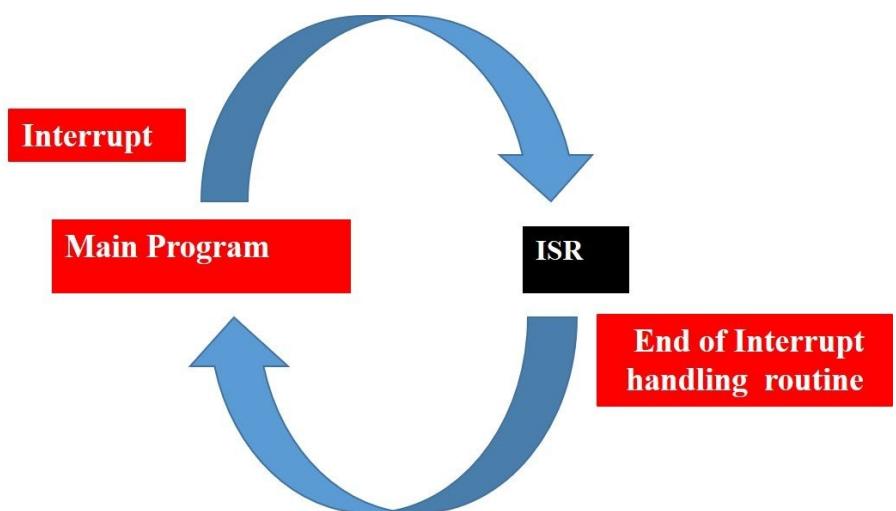
```
from machine import Pin, PWM
from time import sleep
pwm = PWM(5, freq=50, duty_u16=8192)
# create a PWM object on a pin 5
# and set freq and duty
pwm.duty_u16(32768) # set duty to 50%
```



□ Programmer ESP8266 avec Micropython:

Exercice 6: Les interruptions

En électronique, une **interruption** est un signal envoyé à un processeur pour indiquer qu'une tâche importante doit être exécutée **immédiatement**, **interrompant** ainsi l'exécution du programme en cours. Lorsqu'une interruption se produit, le processeur **arrête l'exécution** du programme **principal** pour exécuter une tâche, puis **revient** au programme principal.



Lorsqu'un changement est détecté, un événement est déclenché (une fonction ISR, interrupt Service routine, est appelée).

Les interruptions peuvent être générées de différentes manières, par exemple à la suite d'un **événement** externe, et elles permettent d'exécuter certaines tâches de manière asynchrone, c'est-à-dire indépendamment du programme principal.

Dans la pratique, les interruptions sont généralement utilisées :

- Pour **exécuter des portions** de code critique lorsqu'un événement externe se produit. Par exemple, lorsqu'un bouton est enfoncé, une fonction Python s'exécute automatiquement.
- Pour **exécuter des fonctions périodiquement**. Par exemple, pour faire clignoter une LED toutes les 5 secondes.

□ Programmer ESP8266 avec Micropython:

Exercice 6: Les interruptions

La détection d'un événement est basée sur la forme du signal qui arrive à la broche. Le processeur travaille alors temporairement sur une tâche différente (ISR), puis revient au programme principal une fois la routine de manipulation terminée.

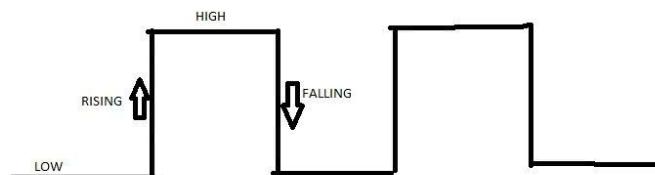


Figure:Digital Signal

Voici les différents types de détection possible d'une interruption :

- **Pin.IRQ_LOW_LEVEL** déclenche l'interruption dès que le signal est à 0V
- **Pin.IRQ_HIGH_LEVEL** déclenche l'interruption dès que le signal est à 3,3 V
- **Pin.IRQ_RISING** : Déclenche l'interruption dès que le signal passe (De 0 à 3.3V)LOW to HIGH
- **Pin.IRQ_FALLING** déclenche l'interruption dès que le signal passe (De 3,3 V à 0)HIGH to LOW

IRQ est l'abréviation de Interrupt Request pour demander une demande d'interruption.

■ Programmer ESP8266 avec Micropython:

Exercice 6: Les interruptions

Pour configurer une interruption dans MicroPython, vous devez suivre les étapes suivantes :

1. **Définissez une fonction de gestion des interruptions:** cette fonction doit être aussi simple que possible, afin que le processeur revienne rapidement à l'exécution du programme principal. La meilleure approche consiste à signaler au code principal que l'interruption s'est produite en utilisant une variable **globale**, pin par exemple. La fonction de gestion des interruptions doit accepter un paramètre de type Pin. Ce paramètre est renvoyé à la fonction de rappel et fait référence à l'objet GPIO à l'origine de l'interruption.

```
def handle_interrupt(pin):
```

2. Configurez le GPIO qui agira comme une broche d'interruption en entrée.

Par exemple:

```
input=Pin(2,Pin.IN)
```

3. Attachez une interruption à cette broche en appelant la commande irq() :

```
pir.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt)
```

Le irq() accepte les arguments suivants :

1. **trigger:** Définit le mode de déclenchement. Il existe 3 conditions différentes :

- **Pin.IRQ_FALLING:** pour déclencher l'interruption chaque fois que la broche passe de HIGH à LOW ;

- **Pin.IRQ_RISING:** pour déclencher l'interruption chaque fois que la broche passe de LOW à HIGH.

2. **handler** : il s'agit d'une fonction qui sera appelée lorsqu'une interruption est détectée, dans ce cas la fonction handle_interrupt().

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

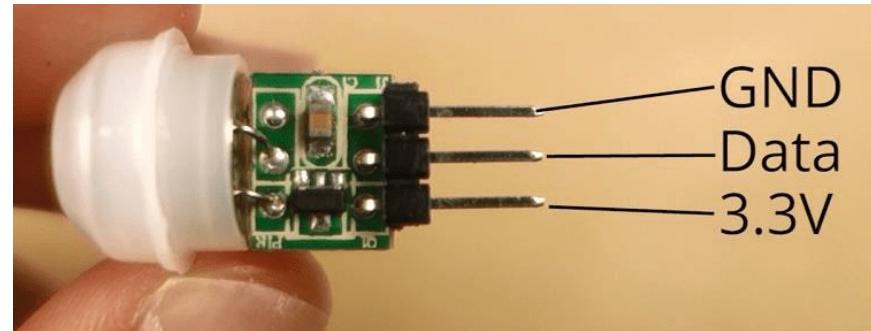
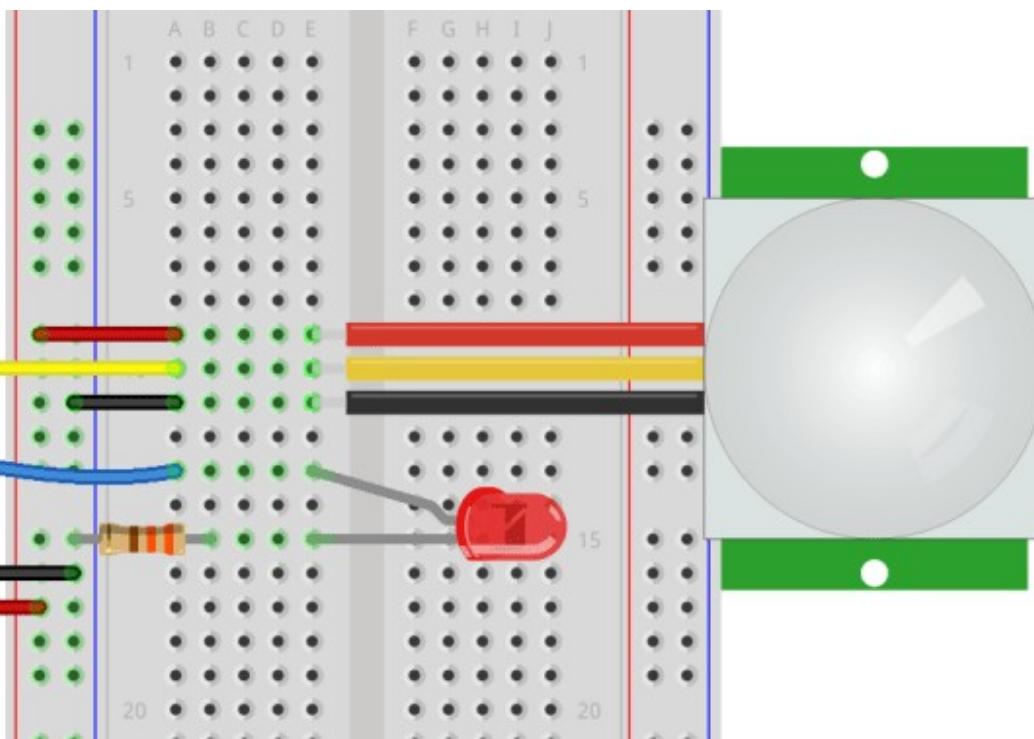
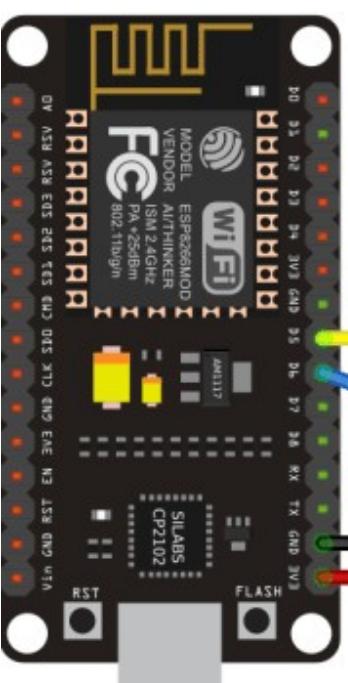
Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython:

Exercice 6: Les interruptions

Exemple d'utilisation d'une interruption avec un capteur de mouvement PIR:



ESP8266 interrupt pins: you can use all GPIOs, except GPIO 16.

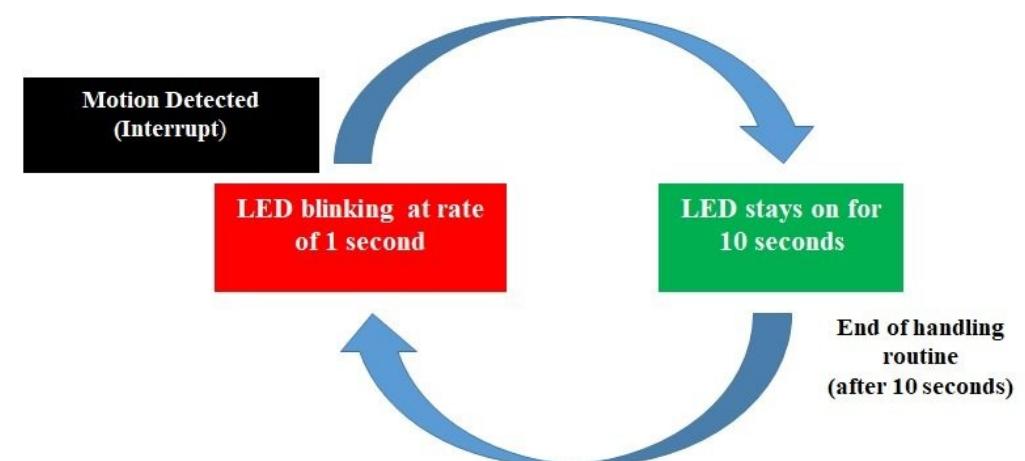
❑ Programmer ESP8266 avec Micropython:

Exercice 6: Les interruptions

Exemple d'utilisation d'une interruption avec un capteur de mouvement PIR:

Le capteur PIR agit comme une source pour l'interruption externe. Cela signifie que nous connectons la sortie du capteur PIR avec la broche **GPIO** de l'ESP8266. De plus, nous attachons l'interruption déclenchée par le **front montant** à cette broche GPIO. Cela signifie que cette broche GPIO déclenchera l'interruption chaque fois qu'elle détectera un front montant sur son entrée.

Initialement, le voyant d'état clignotera après un délai de 1 seconde. Lorsqu'un capteur PIR détecte un mouvement, une interruption externe est provoquée, la LED reste allumée pendant **10** secondes, puis revient à son niveau normal de clignotement (marche/arrêt avec un délai de **1** seconde).



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

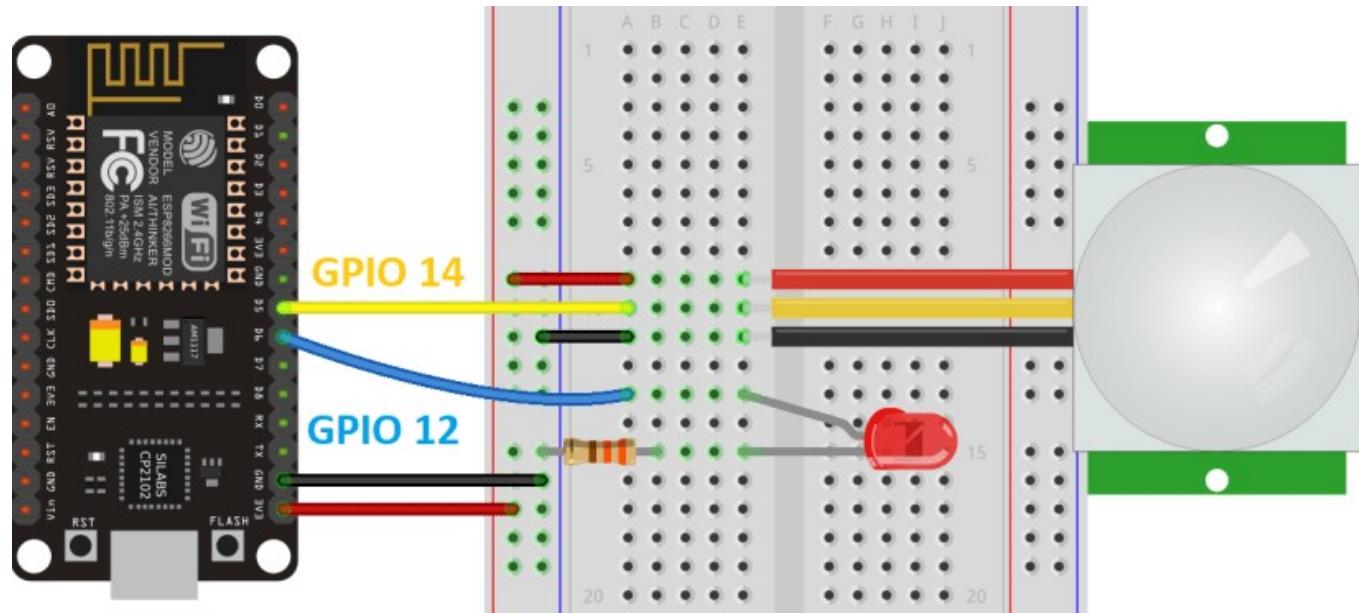
□ Programmer ESP8266 avec Micropython:

Exercice 6: Les interruptions

Exemple d'utilisation d'une interruption avec un capteur de mouvement PIR:

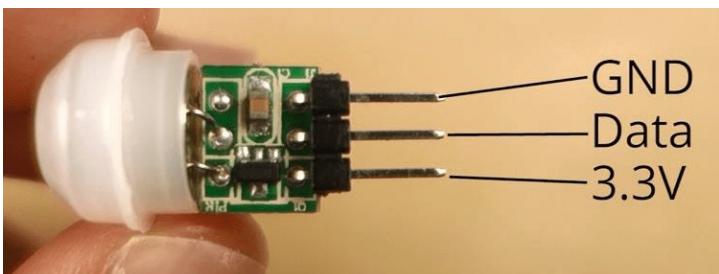
Pièces requises

Carte ESP8266 – LED - Resistance 330 ohms - Mini capteur de mouvement PIR (AM312 ou HC-SR501) - Plaque à essai et fils de connexion



Important : le PIR AM312 fonctionne à 3,3 V.

Cependant, si vous utilisez un autre capteur de mouvement PIR comme le HC-SR501, il fonctionne à 5 V. Vous pouvez soit le modifier pour qu'il fonctionne à 3,3 V, soit simplement l'alimenter à l'aide de la broche Vin.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython:

Exercice 6: Les interruptions

Exemple d'utilisation d'une interruption avec un capteur de mouvement PIR:

```
from machine import Pin          #importing classes
from time import sleep           #Import sleep from time class
Motion_Detected = False          #Global variable to hold the state of motion sensor
def handle_interrupt(pin):        #defining interrupt handling function
    global Motion_Detected
    Motion_Detected = True
led=Pin(12,Pin.OUT)               #setting GPIO12 led as output
PIR_Interrupt=Pin(14,Pin.IN)       # setting GPIO14 PIR_Interrupt as input
#Attach external interrupt to GPIO14 and rising edge as an external event source
PIR_Interrupt.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt)
```

while True:

if Motion_Detected:

print('Motion is detected!')

led.value(1)

sleep(10)

led.value(0)

print('Motion is stopped!')

Motion_Detected = False

else:

led.value(1) #led is on

sleep(1) #delay of 1 second

led.value(0) #led is off

sleep(1) #delay of 1 second

□ Programmer ESP8266 avec Micropython:

Exercice 7: Timers

Auparavant, nous avons appris à faire clignoter une LED en utilisant des retards à l'infini, mais cette fois, nous exécuterons cette fonction en utilisant des **minuteries** à la place. Les minuteries sont plus efficaces que l'utilisation de la fonction veille (sleep) car elles ne bloquent pas les fonctions contrairement à cette dernière. Une fonction de blocage empêche le programme d'effectuer une tâche jusqu'à ce que la précédente soit terminée.

Par exemple, la fonction **sleep()**, lorsque nous appelons sleep(2), notre programme s'arrête à cette ligne pendant 2 secondes, et agit donc comme une fonction bloquante. Ce n'est pas le cas pour les **timers**. Pour effectuer plusieurs tâches à la fois, nous préférons utiliser **Timer** et devons éviter d'utiliser des **delays** car ils ralentissent le processus.

Les minuteries disponibles dans les cartes ESP peuvent également être utilisées pour effectuer une certaine tâche périodiquement après un certain temps. Par exemple, nous pouvons utiliser une interruption de minuterie pour basculer la LED toutes les secondes.

□ Programmer ESP8266 avec Micropython:

Exercice 7: Timers

ESP8266 dispose de deux minuteries : **Timer0** et **Timer1**.

Nous devons utiliser **Timer1** car le Timer0 est utilisé par le **WiFi**. Mais un point important à noter ici est que, la classe MicroPython Timer du module machine ne prend pas en charge les Timers de l'ESP8266. Mais il fournit une API pour les minuteries virtuelles (bases RTOS). Nous pouvons utiliser le module machine et la classe Timer avec un ID de minuterie de -1 pour utiliser des minuteurs virtuels.

```
from machine import Timer
tim = Timer(-1)
tim.init(period=5000, mode=Timer.ONE_SHOT, callback=lambda t:print(1))
```

La fonction **Timer()** a trois arguments, à savoir :

1.Period : c'est la période du signal d'interruption en millisecondes. Il s'agit du temps total jusqu'à ce que le rappel soit appelé.
2.Mode : Nous pouvons choisir entre deux types de mode : 'Timer.PERIODIC' ou 'Timer.ONE_SHOT'. C'est-à-dire si nous voulons configurer notre minuterie comme périodique ou en une seule fois.
3.Callback : le callback est exécuté chaque fois qu'un timer est déclenché. Lorsque nous utiliserons le timer en mode périodique, le callback sera appelé après chaque période que nous spécifierons.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

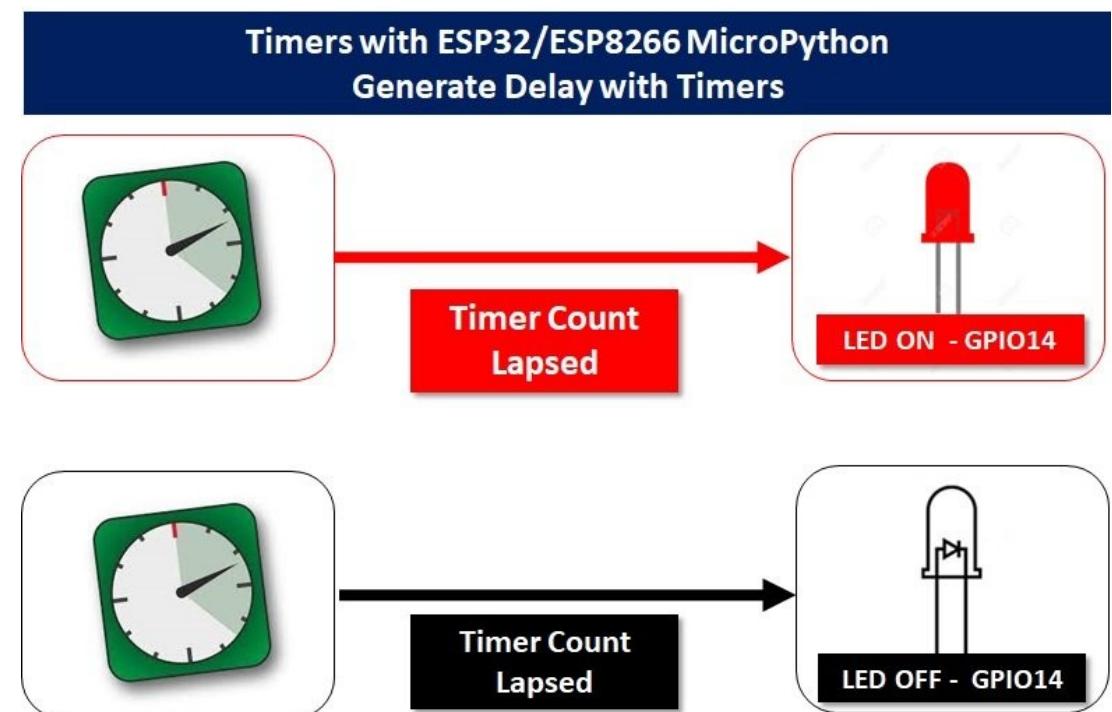
Section 3 : ESP32

Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython:

Exercice 7: Timers

Auparavant, nous devions utiliser une boucle infinie pour retarder la LED ou vérifier si un bouton était enfoncé. Grâce à un **timer**, le même résultat sera obtenu tout en libérant de la place pour l'exécution d'autres processus. Nous allons maintenant voir une démonstration de la façon de faire clignoter une LED à l'aide d'un **timer interrupt**.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython:

Exercice 7: Timers

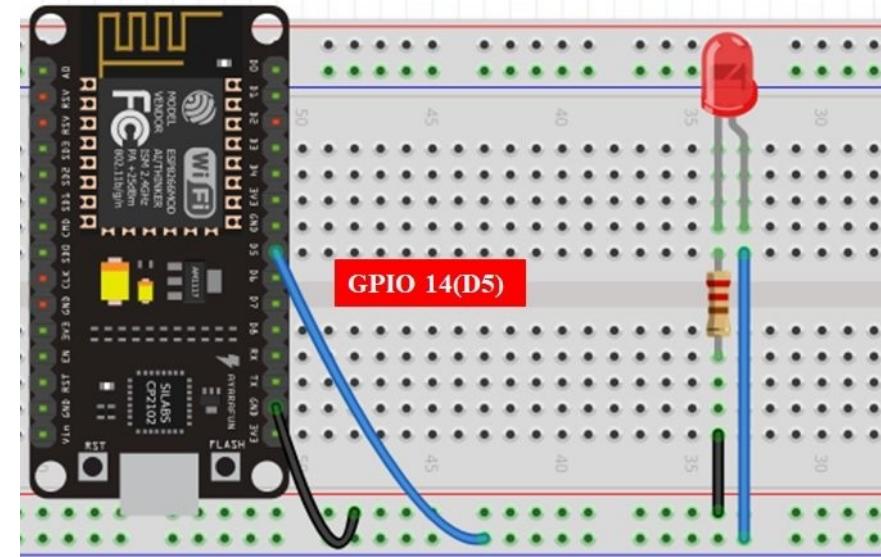
La dernière ligne du programme initialise le timer avec 3 paramètres :

- la période : 1000 ms
- le mode est réglé sur périodique
- Le troisième paramètre est défini sur une fonction qui s'exécutera après chaque période. Dans ce cas, nous le définissons sur une fonction lambda qui bascule la led: led.value(not led.value()) (*function which is toggling the led*)

```
from machine import Pin, Timer  
led= Pin(14, Pin.OUT)  
led.value(0)  
tim=Timer(-1)  
tim.init(period=1000, mode=Timer.PERIODIC, callback=lambda t:led.value(not led.value())) #initializing the timer
```

#importing pin, and timer class
GPIO14 as led output
#LED is off
#create an instance of Timer method (virtual timer)

ESP8266



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

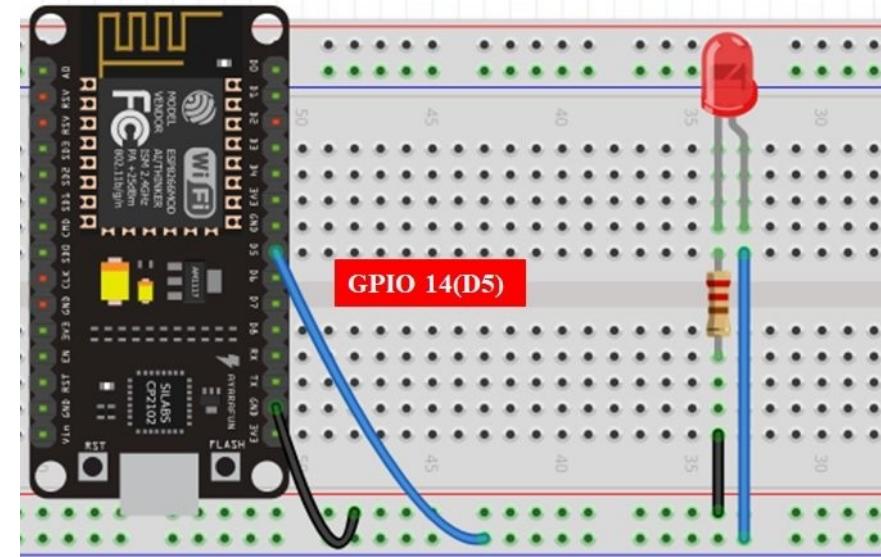
Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython:

Exercice 7: Timers

```
from machine import Pin, Timer  
  
led= Pin(14, Pin.OUT)  
  
led.value(0)  
  
tim=Timer(-1)  
  
def loop(timer):  
    if led.value():  
        led.off()  
    else:  
        tim.init(period=1000, mode=Timer.PERIODIC, callback=loop)
```

ESP8266



❑ Programmer ESP8266 avec Micropython:

Exercice 7: I2C LCD

Dans cet exemple, nous allons apprendre à interfaçer l'écran **LCD I2C** avec l'**ESP8266** et à afficher du texte/des chiffres simples et des caractères personnalisés sur l'écran **LCD I2C**. Cet écran LCD I2C est un appareil **16×2**, ce qui signifie qu'il peut afficher 16 colonnes sur deux rangées de caractères. Les caractères sont alphanumériques, mais vous pouvez créer des caractères personnalisés pour les graphiques de base, les graphiques à barres, etc... L'écran LCD a le type habituel de contrôleur **hd44780**, et il a également un circuit **I2C** connecté avec lui qui facilite la connexion aux cartes **ESP**. L'écran LCD 16X2 sans circuit I2C a seize broches.

Si nous voulons connecter cette carte directement à la carte ESP, nous devons utiliser au moins huit broches de notre carte, ce qui sera un gaspillage. La meilleure solution est donc d'utiliser un écran LCD I2C au lieu d'un écran LCD 16×2 typique.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

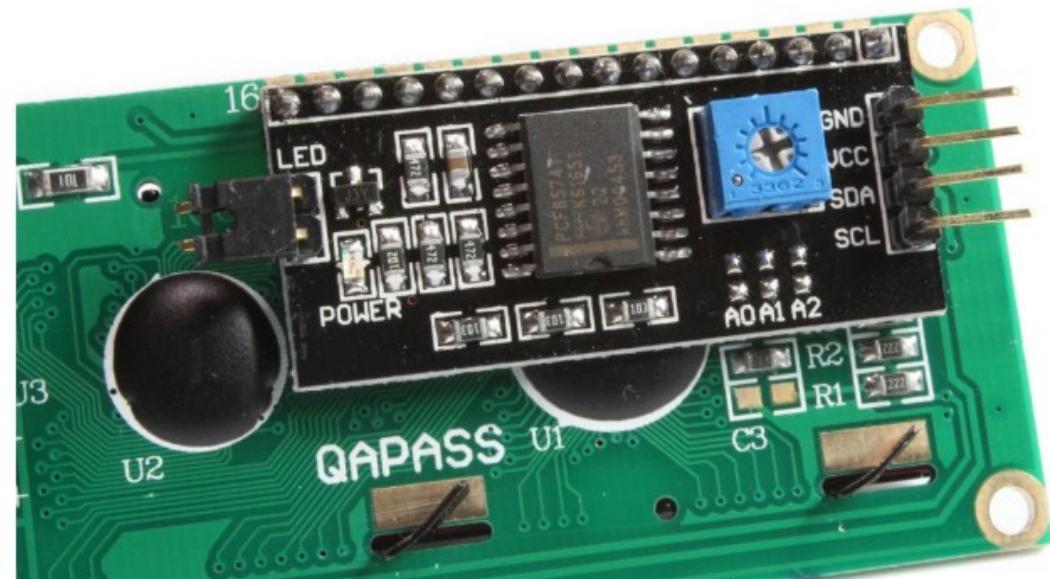
□ Programmer ESP8266 avec Micropython:

Exercice 7: I2C LCD

Cet écran possède 4 broches:

- Broche GND
- Broche Vcc
- SDA
- SCL

À l'arrière de cet écran à cristaux liquides, vous pouvez également voir une résistance variable. Cette résistance variable permet de modifier la luminosité de l'écran LCD. Ce potentiomètre est très pratique lorsque vous utilisez ce module d'affichage dans différentes conditions d'éclairage.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

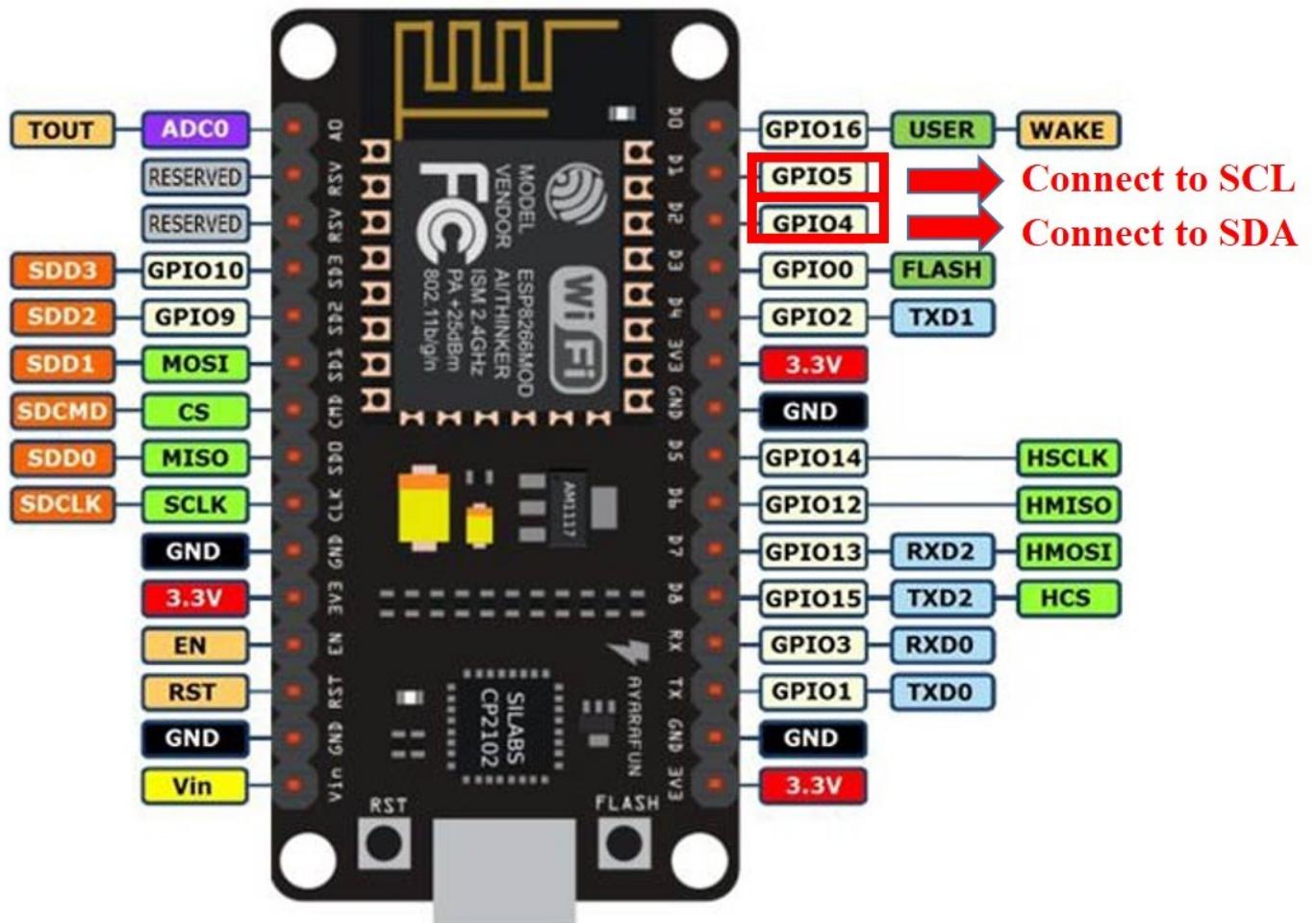
Section 4 : PI Pico

Programmer ESP8266 avec Micropython:

Exercice 7: I2C LCD



ESP8266	I2C LCD
Vin	Vin
GPIO4 (D2)	SDA
GPIO5 (D1)	SCL
GROUND	GND



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

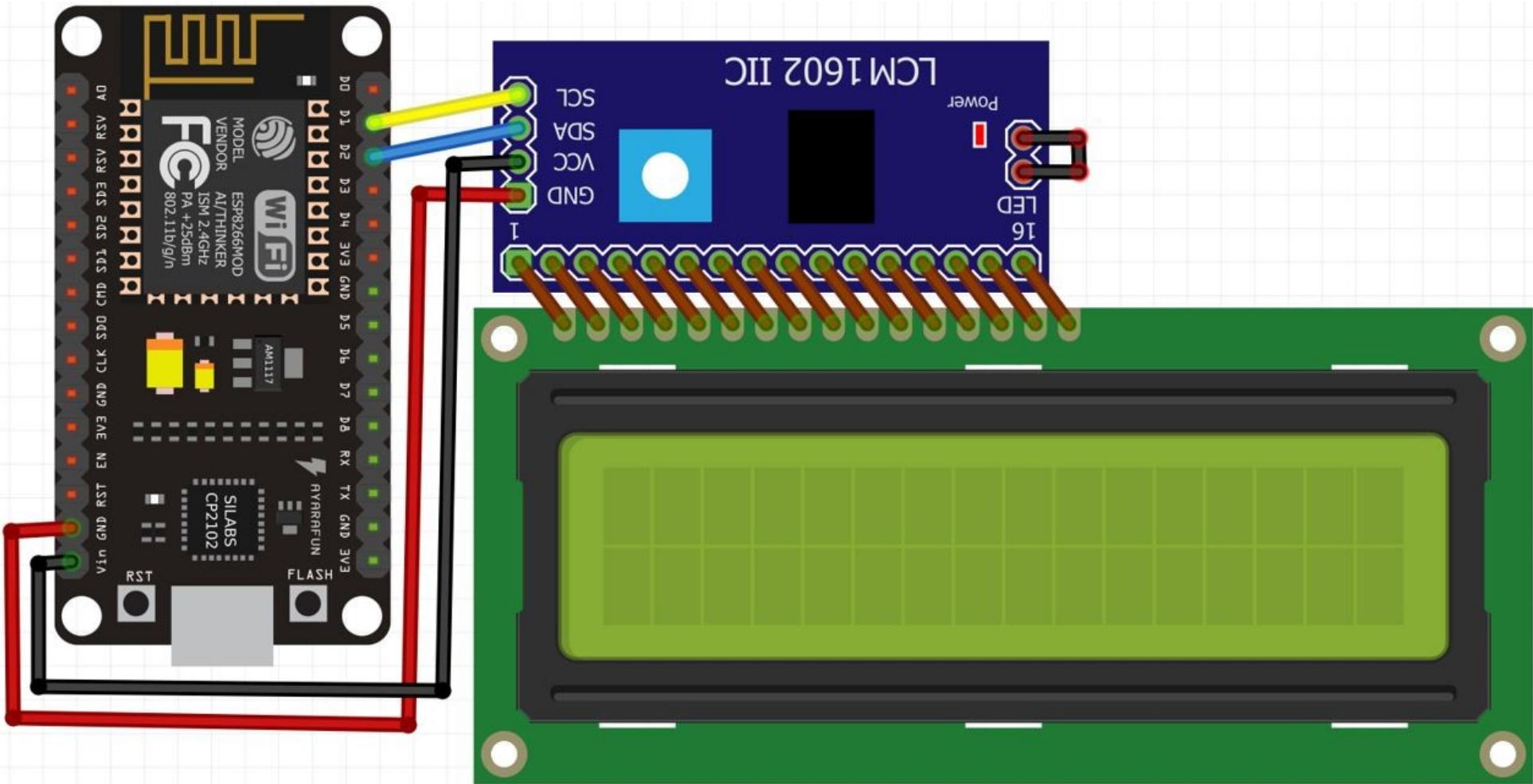
Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

Programmer ESP8266 avec Micropython:

Exercice 7: I2C LCD



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

□ Programmer ESP8266 avec Micropython:

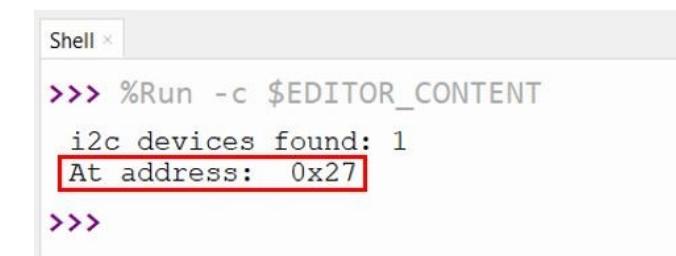
Exercice 7: I2C LCD

Obtention de l'adresse LCD I2C

Lorsque vous connectez votre écran I2C à l'ESP32/ESP8266, vous devez vérifier son adresse. Parce que chaque périphérique I2C est associé à une adresse. Pour de nombreux appareils d'écran LCD I2C, l'adresse par défaut est **0x27** où 0x indique le format hexadécimal des nombres. Mais l'adresse peut être différente dans certains cas. Cette adresse dépend de la position des pads A0, A1 et A2 sur le contrôleur I2C de cet appareil.

Maintenant, copiez ce code et téléchargez-le sur votre carte avec l'écran LCD I2C déjà connecté à celui-ci. Ce code recherchera tous les périphériques I2C connectés avec ESP32 et spécifiera le nombre de périphériques avec l'adresse dans la console du shell.

```
import machine
sdaPIN=machine.Pin(4)
sclPIN=machine.Pin(5)
i2c=machine.I2C(sda=sdaPIN,scl=sclPIN, freq=10000)
devices = i2c.scan()
if len(devices) == 0:
    print('No i2c device !')
else:
    print('i2c devices found:',len(devices))
for device in devices:
    print('At address: ',hex(device))
```



❑ Programmer ESP8266 avec Micropython:

Exercice 7: I2C LCD

Bibliothèques lcd_api.py et i2c_lcd.py

Pour notre exemple, nous aurons besoin de deux bibliothèques : **lcd_api.py** et **i2c_lcd.py** .

On peut trouver ces deux bibliothèques sur internet. Copiez et enregistrez-les dans votre appareil MicroPython avec les noms de fichiers respectifs. Ouvrez un nouveau fichier dans Thonny. Copiez les bibliothèques. Enregistrez-les dans ESP8266 avec les noms **lcd_api.py** et **i2c_lcd.py** dans le dossier lib.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

❑ Programmer ESP8266 avec Micropython:

Exercice 7: I2C LCD

Exemple d'utilisation d'une interruption avec un capteur de mouvement PIR:

```
import machine
from machine import Pin, SoftI2C
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
from time import sleep
I2C_ADDR = 0x27
totalRows = 2
totalColumns = 16
i2c = SoftI2C(scl=Pin(5), sda=Pin(4), freq=10000)
lcd = I2cLcd(i2c, I2C_ADDR, totalRows, totalColumns)
```

while True:

```
    lcd.putstr("I2C MPSEIOT")
    sleep(2)
    lcd.clear()
    lcd.putstr("Lets Count 0-10!")
    sleep(2)
    lcd.clear()
    for i in range(11):
        lcd.putstr(str(i))
        sleep(1)
        lcd.clear()
```

Chapitre 3

Plateformes Micropython

❑ Chapitre 3: Plateformes Micro-python

- L'IDE Thonny
- NodeMCU ESP8266
- **ESP32**
- Pi Pico

Chapitre 3 : Plateformes MicroPython

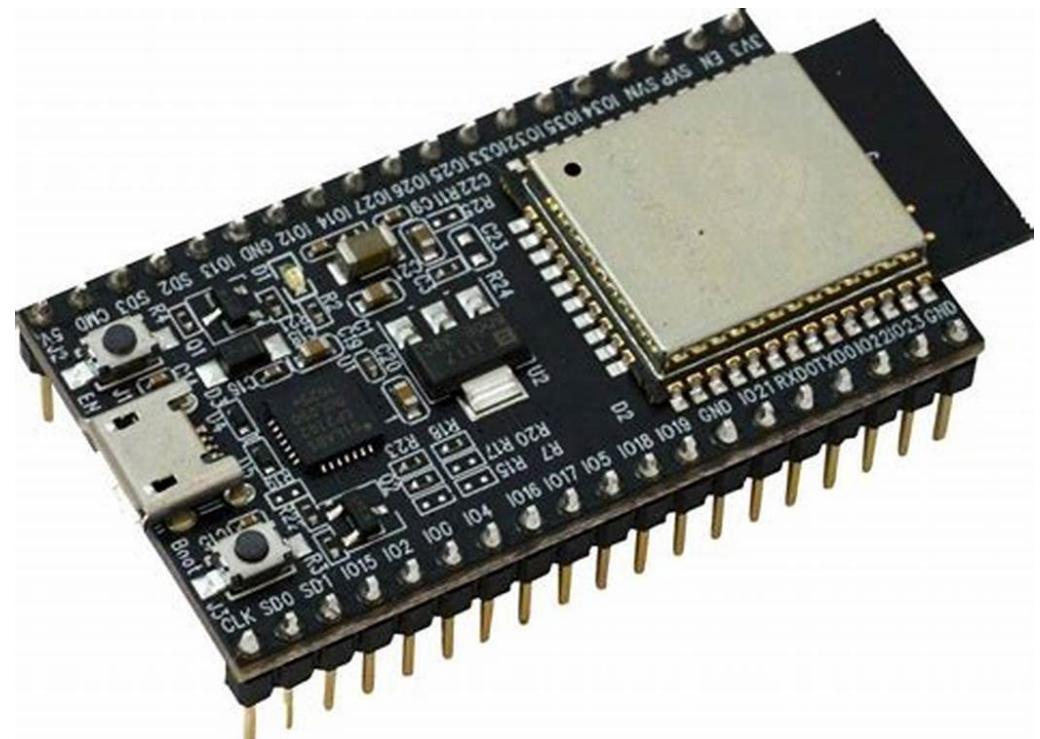
Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

Section 4 : PI Pico

- ❑ L'**ESP32**, développé par le fabricant chinois **Espressif**, est un SoC à faible coût et à faible consommation d'énergie qui inclut des capacités sans fil Wi-Fi et Bluetooth et un processeur double cœur.
 - ❑ Chargé de nombreuses nouvelles fonctionnalités, L'ESP32 est le successeur de l'ESP8266. Il ajoute un cœur de processeur supplémentaire, une connexion WiFi plus rapide, plus de GPIO et prend en charge Bluetooth 4.2 et Bluetooth Low Energy.
 - ❑ De plus, l'ESP32 est livré avec des broches tactiles qui peuvent être utilisées pour réveiller l'ESP32 d'un sommeil profond, un capteur à effet Hall intégré et un capteur de température intégré

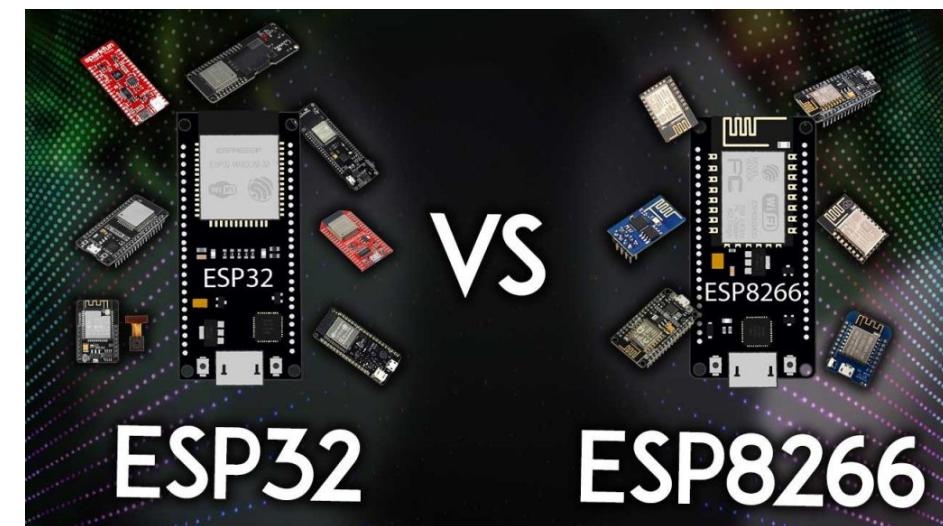


Présentation de l'ESP32:

- **Capacités WiFi:** l'ESP32 peut facilement se connecter à un réseau Wi-Fi (mode station), ou créer son propre réseau sans fil Wi-Fi (mode point d'accès) afin que d'autres appareils puissent s'y connecter.
- **Bluetooth :** l'ESP32 prend en charge Bluetooth classique et Bluetooth Low Energy BLE, ce qui est utile pour une grande variété d'applications IoT ;
- **Dual-core :** la plupart des ESP32 sont dual-core, ils sont équipés de 2 microprocesseurs Xtensa 32-bit LX6 : core 0 et core 1.
- **Interface d'entrée/sortie:** l'ESP32 prend en charge une grande variété de périphériques d'entrée (lecture de données du monde extérieur) et de sortie (pour envoyer des commandes/signaux au monde extérieur) tels que les écrans tactiles capacitifs, ADC, DAC, UART, SPI, I2C, PWM, etc...
- **Compatible** avec le langage de programmation Arduino et avec Micropython
- **Application** aux projets IoT et domotique

Principales différences avec l'ESP8266:

- **L'ESP32** est plus rapide que l'ESP8266 ;
- **L'ESP32** est livré avec plus de GPIO avec de multiples fonctions ;
- **L'ESP32** prend en charge les mesures analogiques sur 18 canaux (broches analogiques) contre une seule broche ADC 10 bits sur le ESP8266 ;
- **L'ESP32** prend en charge le Bluetooth alors que le ESP8266 ne le fait pas ;
- **L'ESP32** est à double cœur (la plupart des modèles) et le ESP8266 est à un seul cœur ;
- **L'ESP32** est un peu plus cher que le ESP8266.



Modules et cartes de développement ESP32:

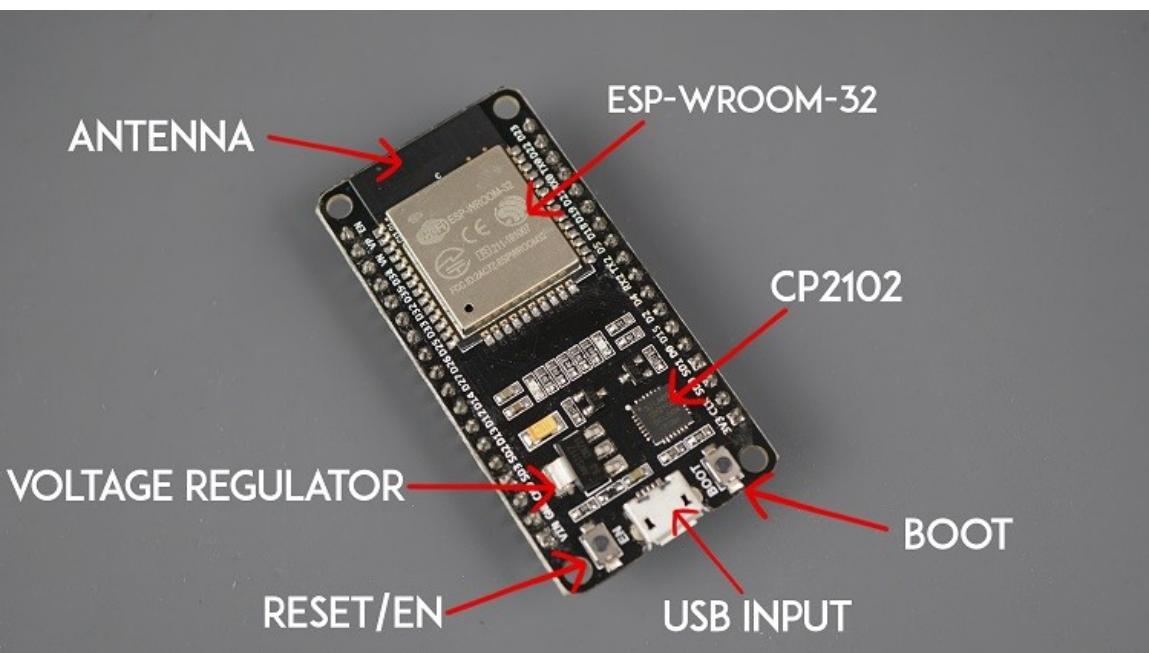
- **Module ESP32:** Il s'agit de modules montables en surface qui contiennent la puce. Les modules peuvent être fixés à une carte de circuit imprimé. L'avantage ici est que vous pouvez facilement monter ces modules sur une carte MCU. Les modules les plus populaires: **ESP32-WROOM-32, ESP32-WROOVER.**
- **Carte de développement ESP32:** Il s'agit de cartes de développement de microcontrôleurs IoT sur lesquelles les modules contenant la puce **ESP32** sont préinstallés. Ils sont utilisés par les amateurs, les fabricants d'appareils et les développeurs pour tester et prototyper des appareils IoT avant d'entrer dans la production de masse. Il existe une grande variété de marques et de modèles de cartes de développement ESP32, produites par différents fabricants, les plus populaires: **ESP32 CAM, ESP32-DevKit V1 DOIT, HUZZAH32**



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Caractéristiques ESP32- DevKit V1



Number of cores	2 (dual core)
Wi-Fi	2.4 GHz up to 150 Mbits/s
Bluetooth	BLE (Bluetooth Low Energy) and legacy Bluetooth
Architecture	32 bits
Clock frequency	Up to 240 MHz
RAM	512 KB
Pins	30, 36, or 38 (depending on the model)
Peripherals	Capacitive touch, ADC (analog to digital converter), DAC (digital to analog converter), I2C (Inter-Integrated Circuit), UART (universal asynchronous receiver/transmitter), CAN 2.0 (Controller Area Network), SPI (Serial Peripheral Interface), I2S (Integrated Inter-IC Sound), RMII (Reduced Media-Independent Interface), PWM (pulse width modulation), and more.
Built-in buttons	RESET and BOOT buttons
Built-in LEDs	built-in blue LED connected to GPIO2; built-in red LED that shows the board is being powered
USB to UART bridge	CP2102

Configuration Pinout ESP32- DevKit V1

L'un des avantages de l'ESP32 est qu'il dispose de beaucoup plus de GPIO que le ESP8266.

Bien que l'ESP32 dispose de 48 broches GPIO au total, seules 25 d'entre elles sont réparties sur les en-têtes de broches des deux côtés de la carte de développement. Ces broches peuvent être affectées à une variété de tâches périphériques.

La fonction de multiplexage des broches de l'ESP32 permet à plusieurs périphériques de partager une seule broche GPIO. Par exemple, une seule broche GPIO peut servir d'entrée ADC, de sortie DAC ou de pavé tactile.

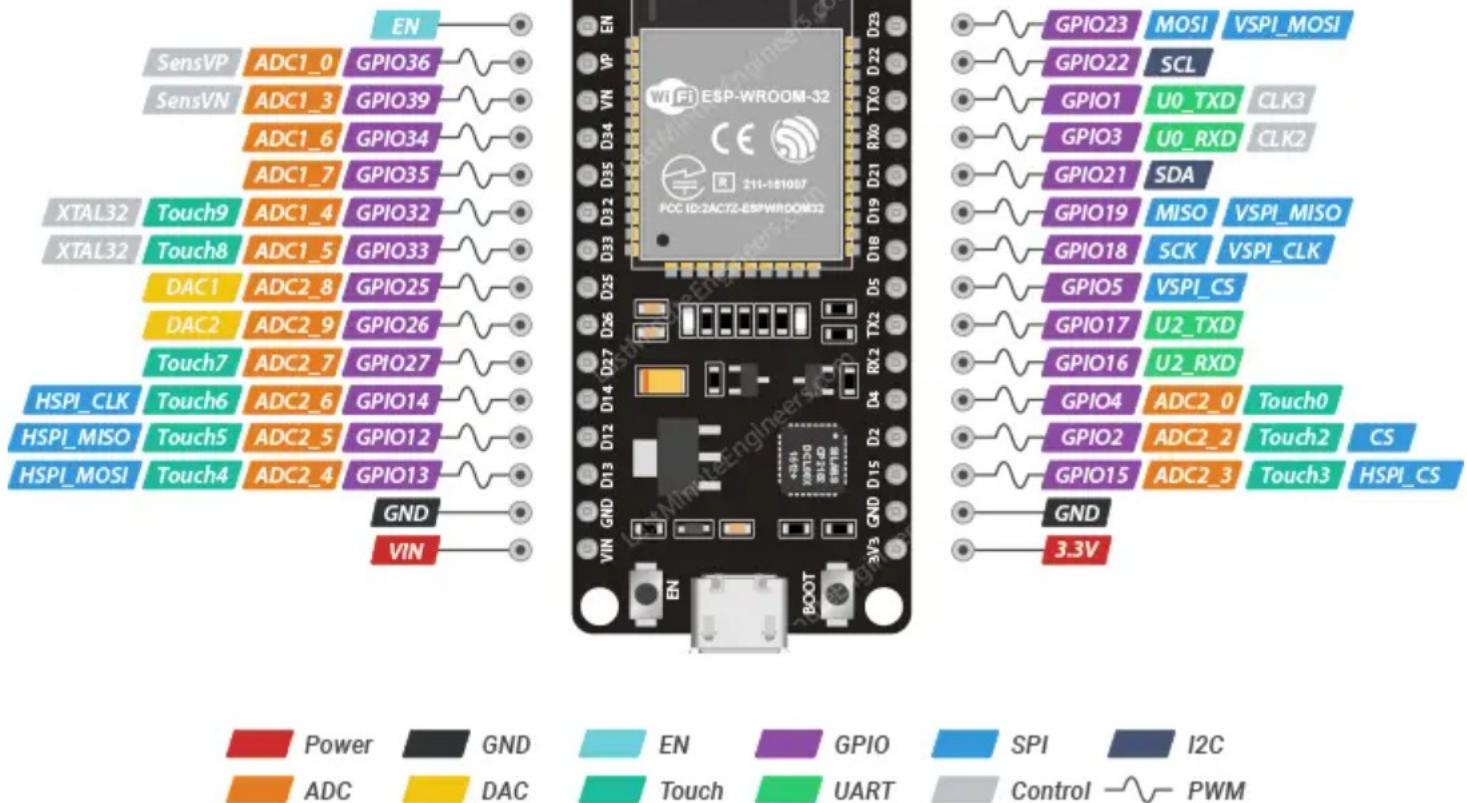
15 canaux ADC	15 canaux de CAN SAR 12 bits avec des plages sélectionnables de 0 à 1 V, 0 à 1,4 V, 0 à 2 V ou 0 à 4 V
2 interfaces UART	2 interfaces UART avec contrôle de flux et prise en charge IrDA
25 sorties PWM	25 broches PWM pour contrôler des éléments tels que la vitesse du moteur ou la luminosité des LED
2 canaux DAC	Deux convertisseurs numériques 8 bits pour générer de vraies tensions analogiques
Interface SPI, I2C et I2S	Trois interfaces SPI et une interface I2C pour connecter divers capteurs et périphériques, ainsi que deux interfaces I2S pour ajouter du son à votre projet
9 pavés tactiles	9 GPIO avec détection tactile capacitive

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Configuration Pinout ESP32- DevKit V1

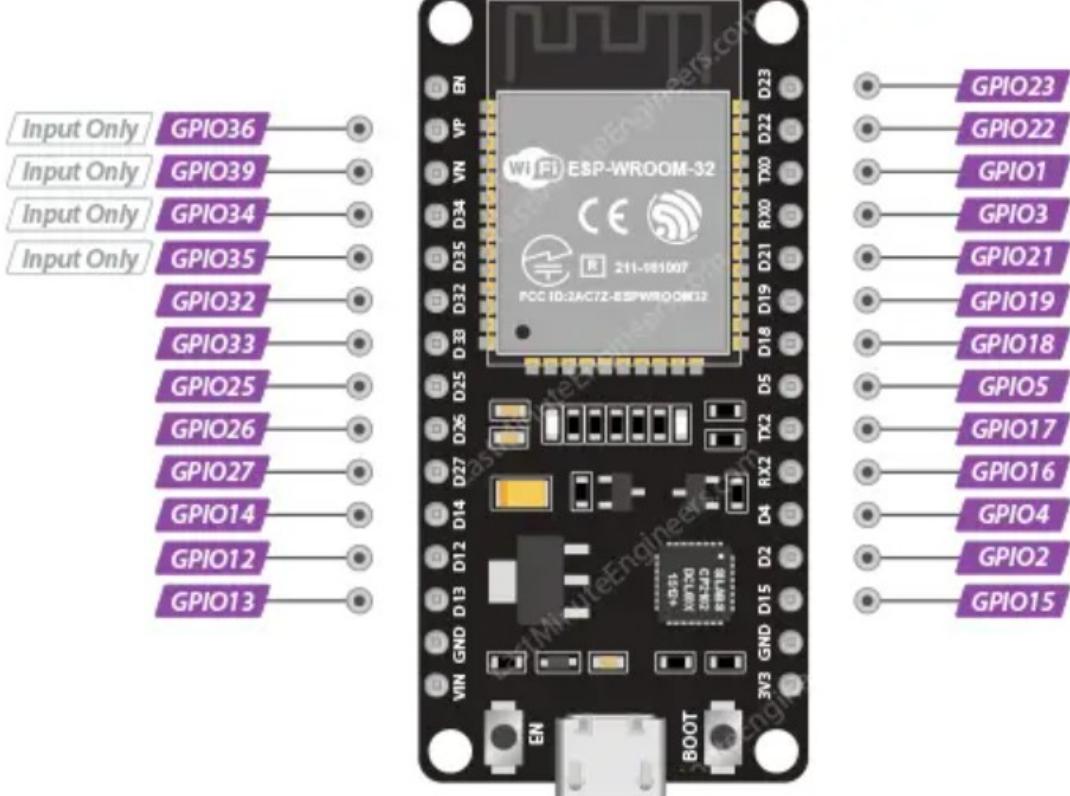
Veuillez noter que le Pinout suivant concerne la carte de développement ESP32 DevKit V1 à 30 broches.



Configuration Pinout ESP32- DevKit V1

Broches GPIO

La carte de développement ESP32 dispose de 25 broches **GPIO** qui peuvent être affectées à différentes fonctions en programmant les registres appropriés. Il existe plusieurs types de GPIO : numériques uniquement, analogiques, tactiles, etc. Les GPIO analogiques et les GPIO tactiles capacitifs peuvent être configurés en tant que GPIO numériques.



Configuration Pinout ESP32- DevKit V1

Broches GPIO

Bien que l'ESP32 dispose de nombreuses broches avec diverses fonctions, certaines d'entre elles peuvent ne pas convenir à vos projets. Le tableau ci-dessous indique quelles broches peuvent être utilisées en toute sécurité et quelles broches doivent être utilisées avec prudence.

Label	GPIO	Safe to use?	Reason
D0	0	!	must be HIGH during boot and LOW for programming
TX0	1	✗	Tx pin, used for flashing and debugging
D2	2	!	must be LOW during boot and also connected to the on-board LED
RX0	3	✗	Rx pin, used for flashing and debugging
D4	4	✓	
D5	5	!	must be HIGH during boot
D6	6	✗	Connected to Flash memory
D7	7	✗	Connected to Flash memory
D8	8	✗	Connected to Flash memory
D9	9	✗	Connected to Flash memory
D10	10	✗	Connected to Flash memory
D11	11	✗	Connected to Flash memory
D12	12	!	must be LOW during boot

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Configuration Pinout ESP32- DevKit V1

Broches GPIO

Bien que l'ESP32 dispose de nombreuses broches avec diverses fonctions, certaines d'entre elles peuvent ne pas convenir à vos projets. Le tableau ci-dessous indique quelles broches peuvent être utilisées en toute sécurité et quelles broches doivent être utilisées avec prudence.

D34	34	!	Input only GPIO, cannot be configured as output
D35	35	!	Input only GPIO, cannot be configured as output
VP	36	!	Input only GPIO, cannot be configured as output
VN	39	!	Input only GPIO, cannot be configured as output

D13	13	✓	
D14	14	✓	
D15	15	!	must be HIGH during boot, prevents startup log if pulled LOW
RX2	16	✓	
TX2	17	✓	
D18	18	✓	
D19	19	✓	
D21	21	✓	
D22	22	✓	
D23	23	✓	
D25	25	✓	
D26	26	✓	
D27	27	✓	
D32	32	✓	
D33	33	✓	

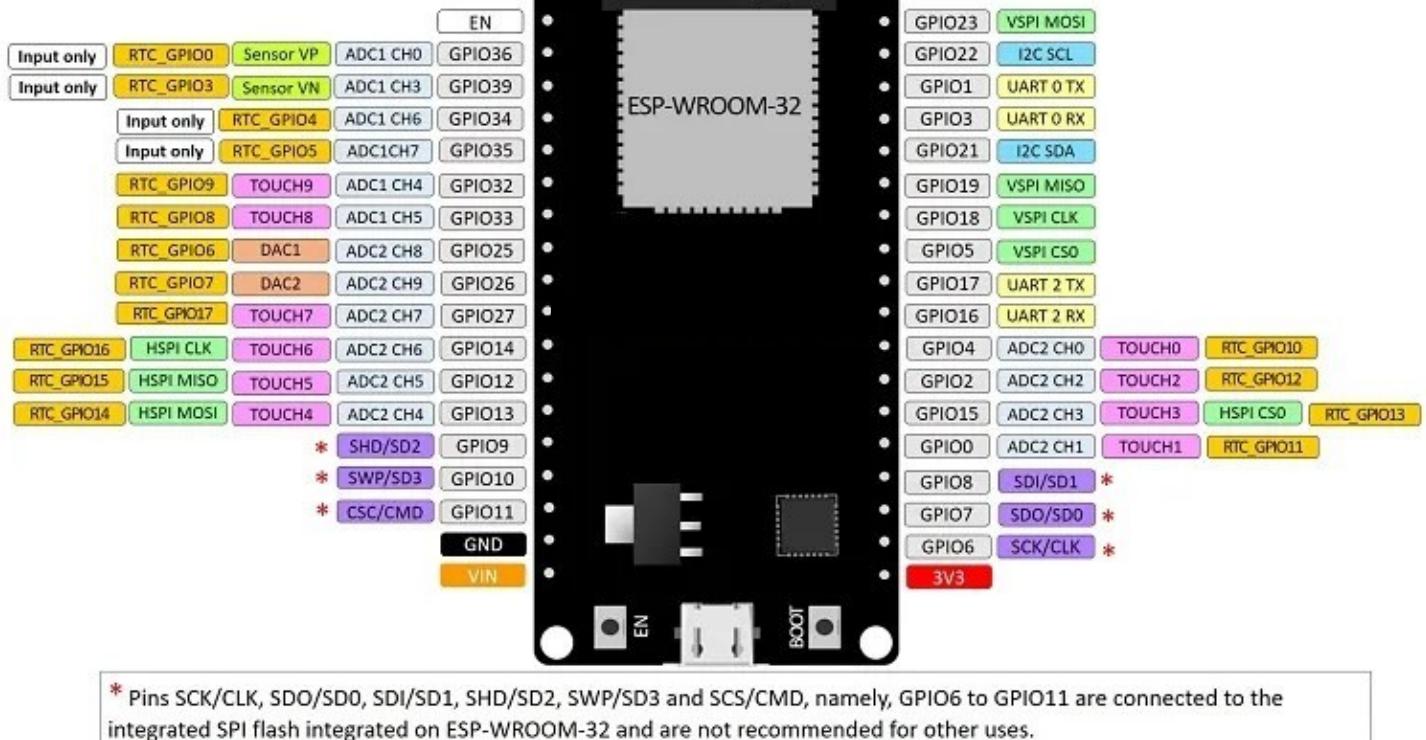
Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Configuration Pinout ESP32- DevKit V1

Broches GPIO

Le Pinout suivant concerne la carte de développement
ESP32 DevKit V1 à 36 broches.



□ Programmer ESP32 avec Micropython:

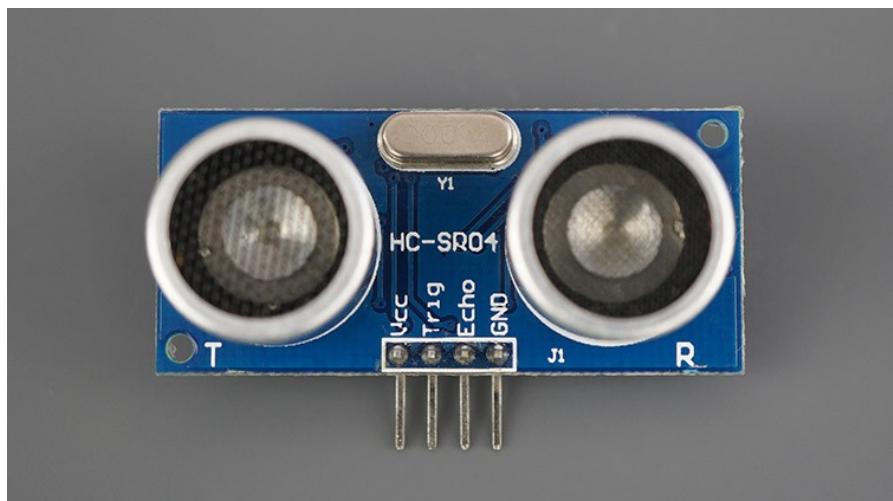
Applications à réaliser..

- LED Clignotante
- GPIO
- Push button
- Analog inputs (ADC)
- PWM
- Stepper motor
- Servo motor
- Interrupts Timers
- Deep sleep
- Capteur à ultrasons
- Capteurs Température / humidité / pression
- Les relais
- RFID
- Web server
- MQTT
- WIFI
- Etc..

Programmer ESP32 avec micropython

Exemple 1: Mesurer des distances avec le capteur ultrason HC-SR04

Le capteur HC-SR04 permet de mesurer des distances par rapport à un obstacle en se basant sur des ondes ultrasons. Les modules disponibles sont constitués en général d'un émetteur et récepteur ultrason ainsi que d'une puce qui s'occupe de les piloter.



Caractéristiques techniques du module HC-SR04

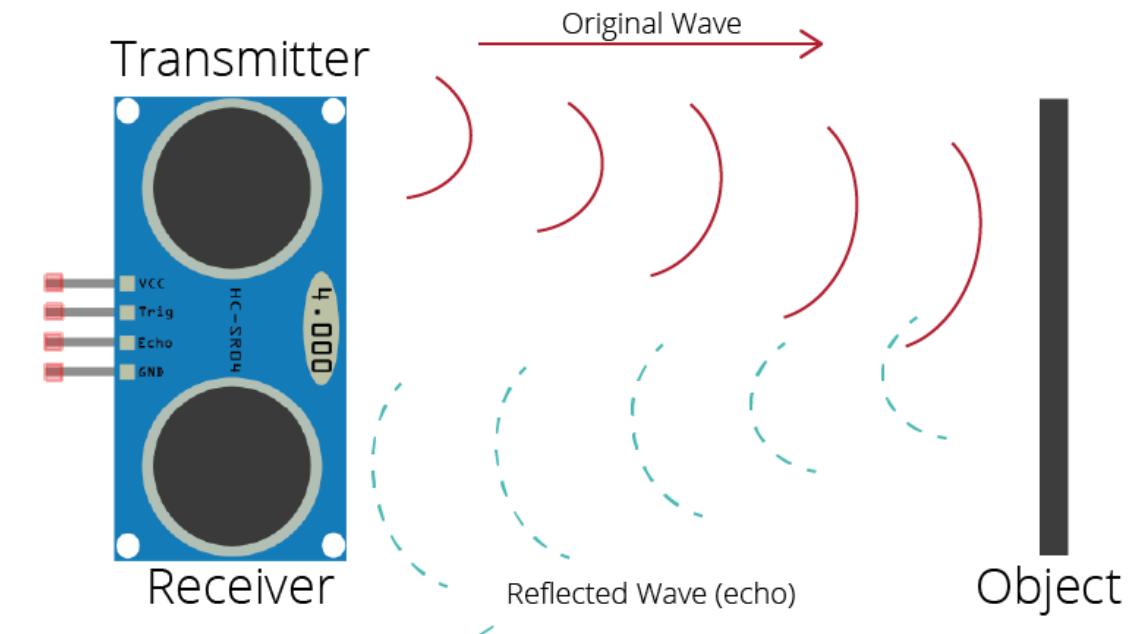
Alimentation	5V CC
Courant de travail	15 mA
Fréquence de travail	40 kHz
Portée maximale	4 mètres
Portée minimale	2 cm
Angle de mesure	15°
Résolution	0,3 cm
Signal d'entrée de déclenchement	Impulsion TTL 10uS
Signal de sortie d'écho	Impulsion TTL proportionnelle à la plage de distance
Taille	45mm x 20mm x 15mm

Programmer ESP32 avec micropython

Comment fonctionne le capteur ultrason HC-SR04

Le capteur à ultrasons utilise un sonar pour déterminer la distance d'un objet:

- 1.L'émetteur à ultrasons (broche trig) émet un son à haute fréquence (40 kHz).
- 2.Le son se propage dans l'air. S'il trouve un objet, il rebondit vers le module.
- 3.Le récepteur à ultrasons (broche d'écho) reçoit le son réfléchi (écho).



Le temps entre l'émission et la réception du signal permet de calculer la distance à un objet:

$$\text{distance to an object} = ((\text{speed of sound in the air}) * \text{time}) / 2$$

*vitesse du son dans l'air à 20°C (68°F) = 343 m/s

Chapitre 3 : Plateformes MicroPython

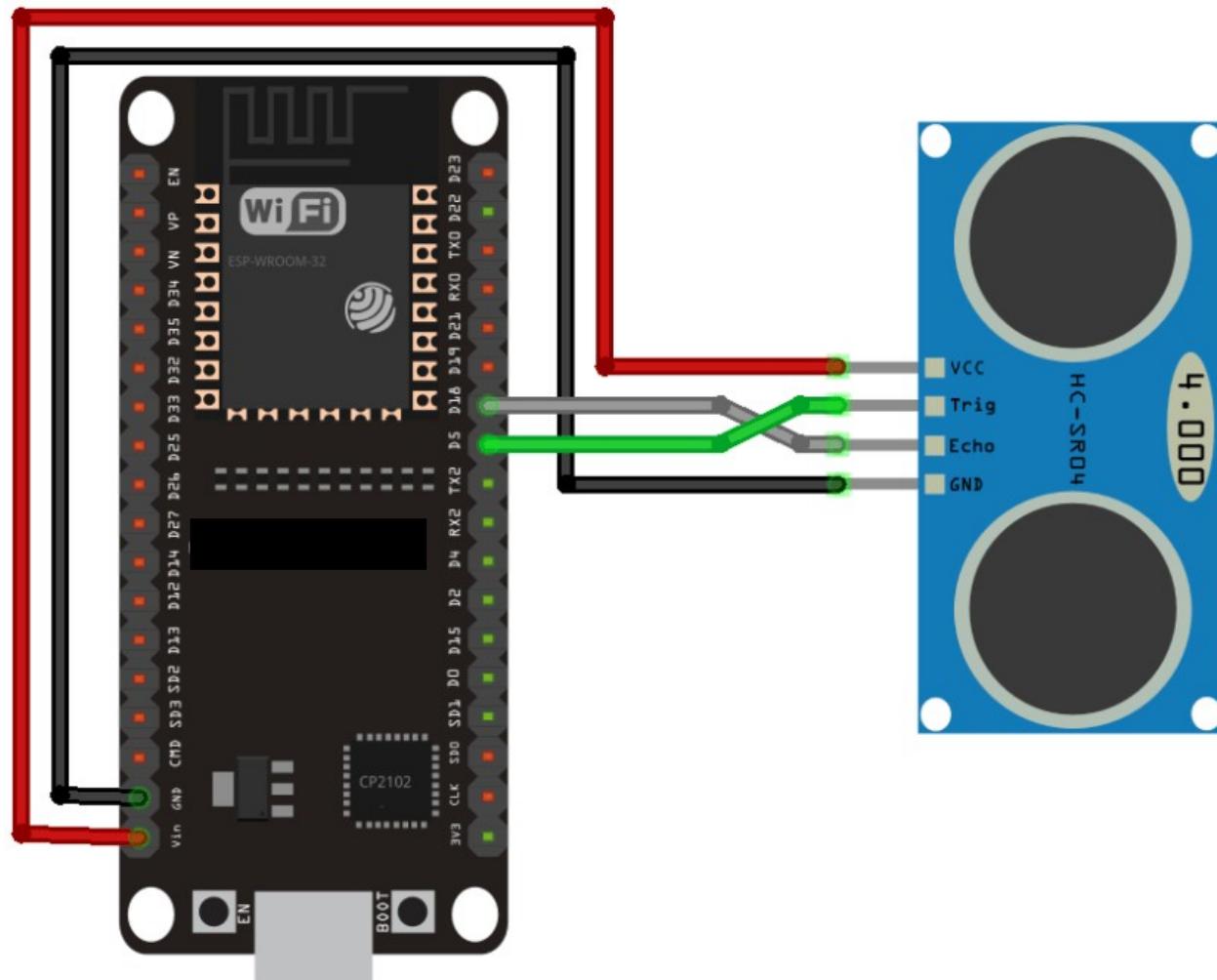
Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Comment fonctionne le capteur ultrason HC-SR04

Brochage:

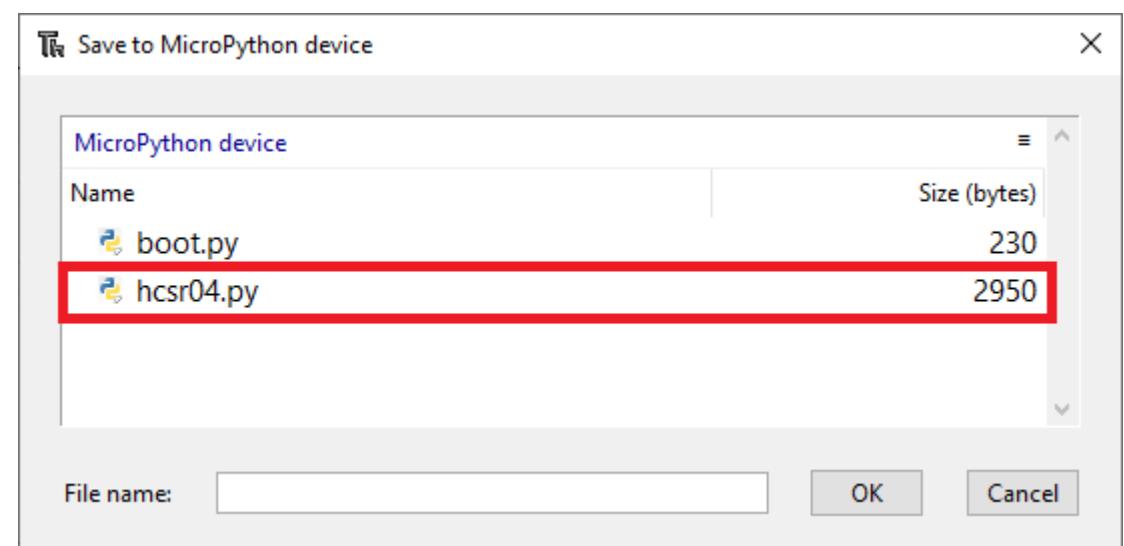
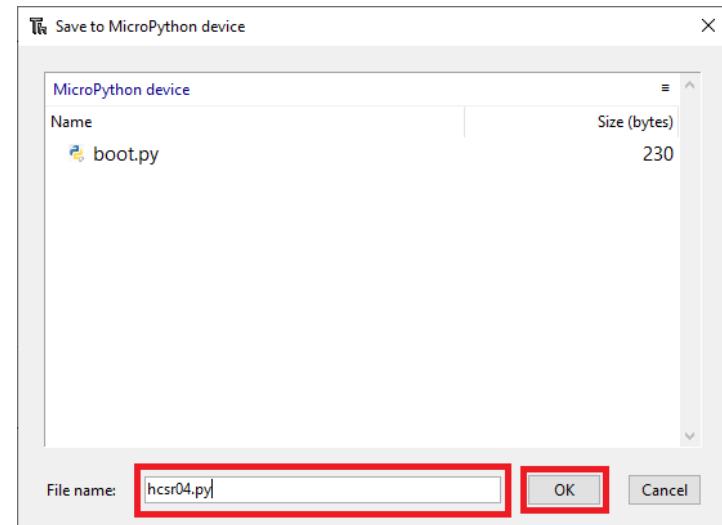
1. Connectez la broche **VCC** du capteur HC-SR04 à la broche VCC (ou 5V) de votre microcontrôleur.
2. Connectez la broche **GND** du capteur HC-SR04 à la broche GND de votre microcontrôleur.
3. Connectez la broche **TRIG** du capteur HC-SR04 à une broche numérique (ici GPIO5) de votre microcontrôleur. Cette broche sera utilisée pour envoyer un signal de déclenchement au capteur.
4. Connectez la broche **ECHO** du capteur HC-SR04 à une autre broche numérique (ici GPIO18) de votre microcontrôleur. Cette broche sera utilisée pour recevoir le signal de retour du capteur.



Programmer ESP32 avec micropython

Bibliothèque pour le capteur ultrason HC-SR04

La bibliothèque que nous allons utiliser ne fait pas partie de la bibliothèque MicroPython standard par défaut. Vous devez donc télécharger cette bibliothèque sur votre carte ESP32/ESP8266 (enregistrez-la sous le nom **hcsr04.py**) avec thonny



Programmer ESP32 avec micropython

Programme Micropython

```
from hcsr04 import HCSR04
from time import sleep
sensor = HCSR04(trigger_pin=5, echo_pin=18, echo_timeout_us=10000)
while True:
    distance = sensor.distance_cm()
    print('Distance:', distance, 'cm')
    sleep(1)
```

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

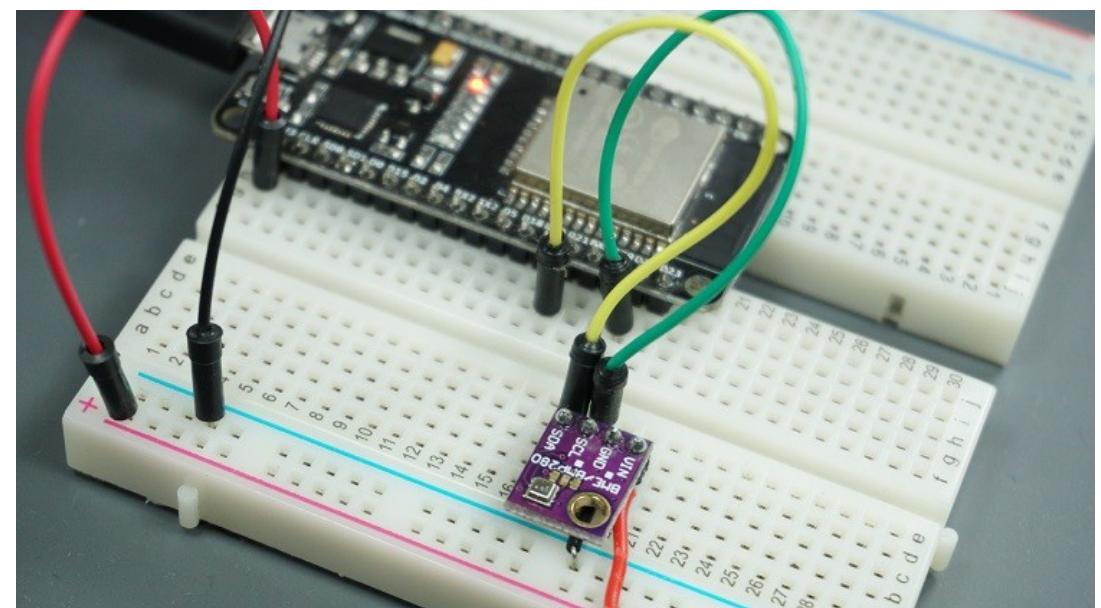
Programmer ESP32 avec micropython

- Exemple 2: Pression, température, humidité : capteur

BME280 avec l'ESP32

Le module de capteur **BME280** lit la pression barométrique, la température et l'humidité. Ce capteur communique à l'aide du protocole de communication I2C, le câblage est donc très simple. Vous pouvez utiliser les broches ESP32, comme indiqué dans le tableau suivant :

BME280	ESP32	ESP8266
Vin	3.3V	3.3V
GND	GND	GND
SCL	GPIO 22	GPIO 5 (D1)
SDA	GPIO 21	GPIO 4 (D2)

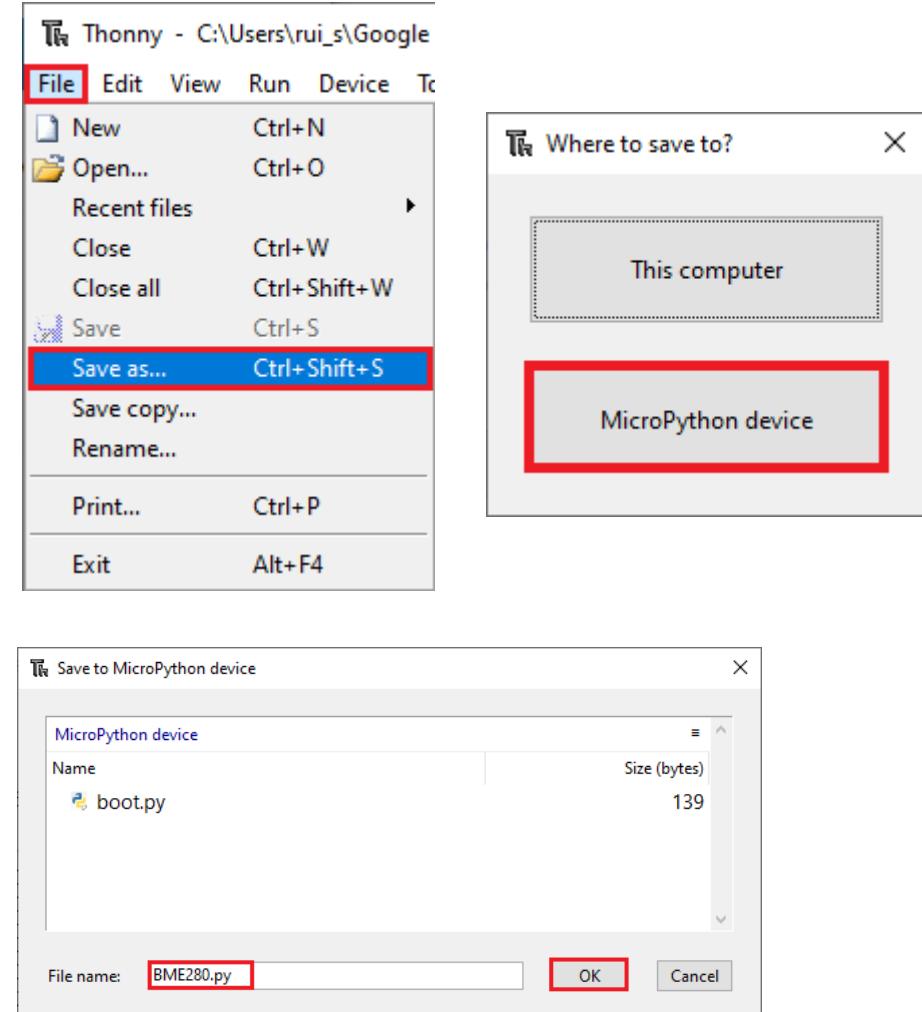


Programmer ESP32 avec micropython

- Pression, température, humidité : capteur BME280 avec l'ESP32

La bibliothèque à lire à partir du capteur BME280 ne fait pas partie de la bibliothèque MicroPython standard par défaut. Vous devez donc télécharger la bibliothèque de ce capteur sur votre carte ESP32/ESP8266 (enregistrez-la sous le nom *BME280.py*).

```
from machine import I2C
import time
# BME280 default address.
BME280_I2CADDR = 0x76
# Operating Modes
BME280_OSAMPLE_1 = 1
BME280_OSAMPLE_2 = 2
BME280_OSAMPLE_4 = 3
BME280_OSAMPLE_8 = 4
BME280_OSAMPLE_16 = 5
.....
```

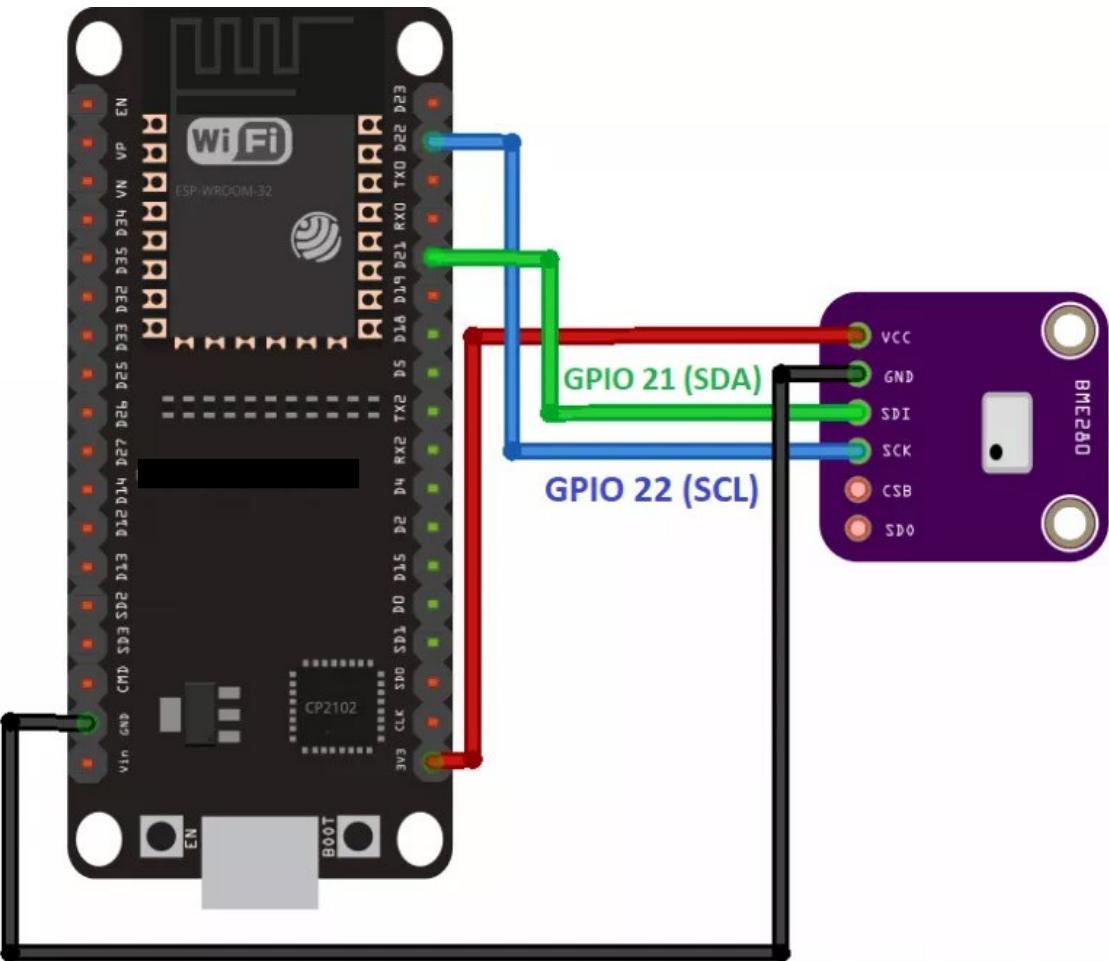


Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

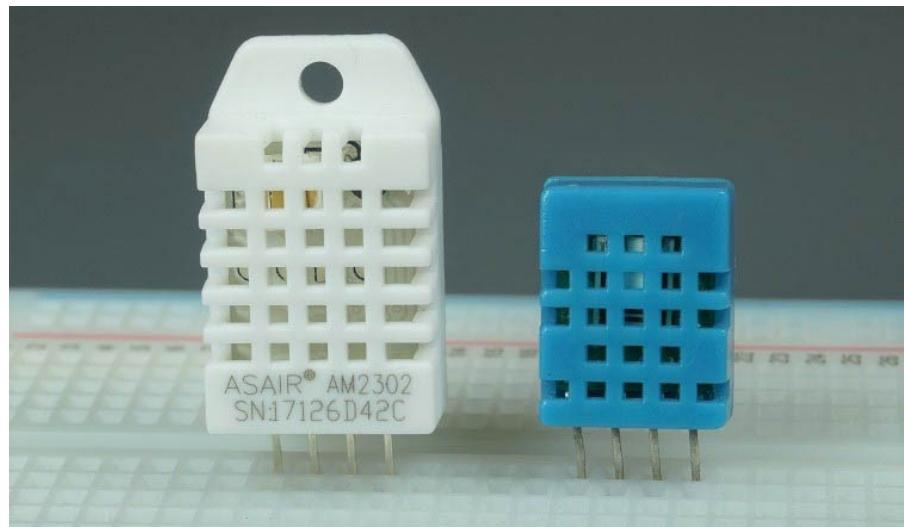
```
from machine import Pin, I2C
from time import sleep
import BME280
# ESP32 - Pin assignment
i2c = I2C(scl=Pin(22), sda=Pin(21), freq=10000)
while True:
    bme = BME280.BME280(i2c=i2c) # créer un objet BME280
    temp = bme.temperature
    hum = bme.humidity
    pres = bme.pressure
    print('Temperature: ', temp)
    print('Humidity: ', hum)
    print('Pressure: ', pres)
    sleep(5)
```



Programmer ESP32 avec micropython

➤ Exemple 3: Température, humidité : capteur DH11 avec l'ESP32

Le module de capteur **DH11** lit la température et l'humidité. Il existe un modèle plus avancé, le DHT 22, qui est plus complet. On peut les différencier en regardant les couleurs de leur boîtier : le DHT11 a une couleur bleue alors que le DHT22 est blanc.



Ces capteurs contiennent une puce qui effectue une conversion analogique-numérique et crache un signal numérique avec la température et l'humidité. Cela les rend très faciles à utiliser avec n'importe quel microcontrôleur.

	DHT11	DHT22
Plage de température	0 à 50 °C +/- 2°C	-40 à 80 °C +/- 0,5 °C
Plage d'humidité	20 à 90% +/- 5%	0 à 100% +/- 2%
Résolution	Humidité : 1% Température : 1°C	Humidité : 0.1% Température : 0.1°C
Tension de fonctionnement	3 à 5,5 V CC	3 à 6 V CC
Alimentation en courant	0,5 à 2,5 mA	1 à 1,5 mA
Période d'échantillonnage	1 seconde	2 secondes

Chapitre 3 : Plateformes MicroPython

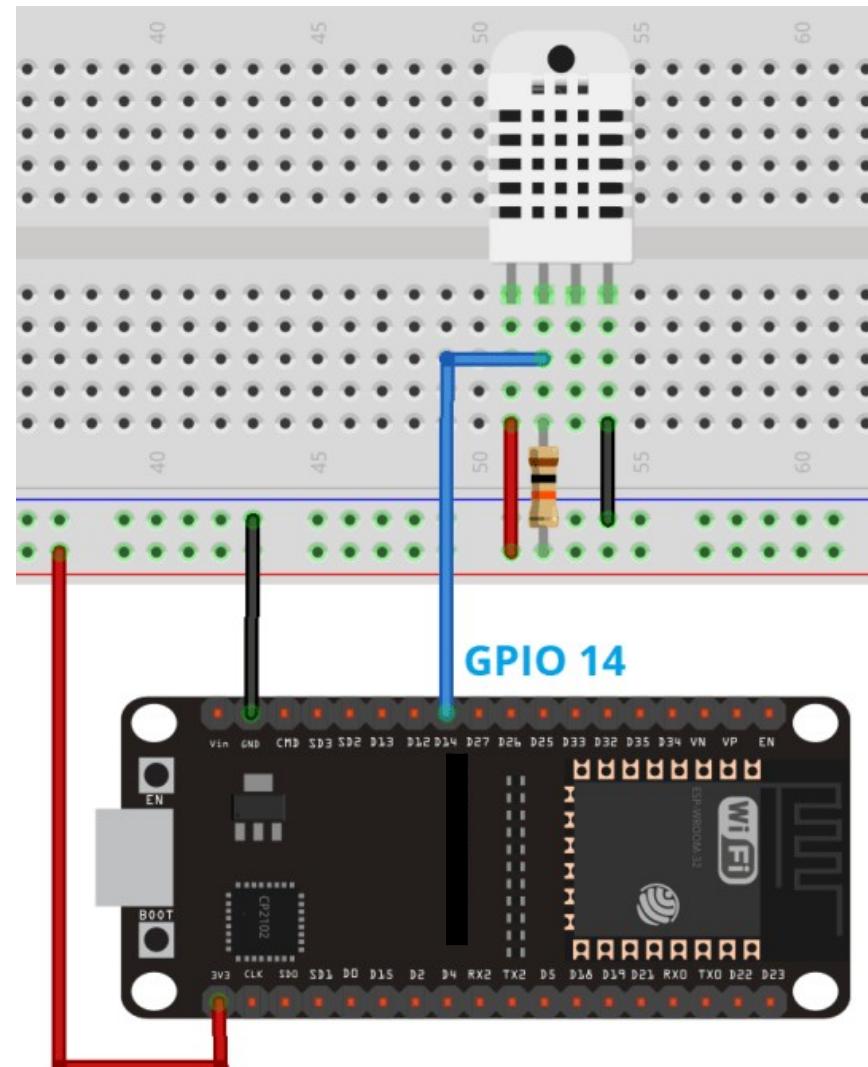
Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

- Température, humidité : capteur DH11 avec l'ESP32



DHT pin	Connect to
1	3.3V
2	Any digital GPIO; also connect a 10k Ohm pull-up resistor
3	Don't connect
4	GND



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

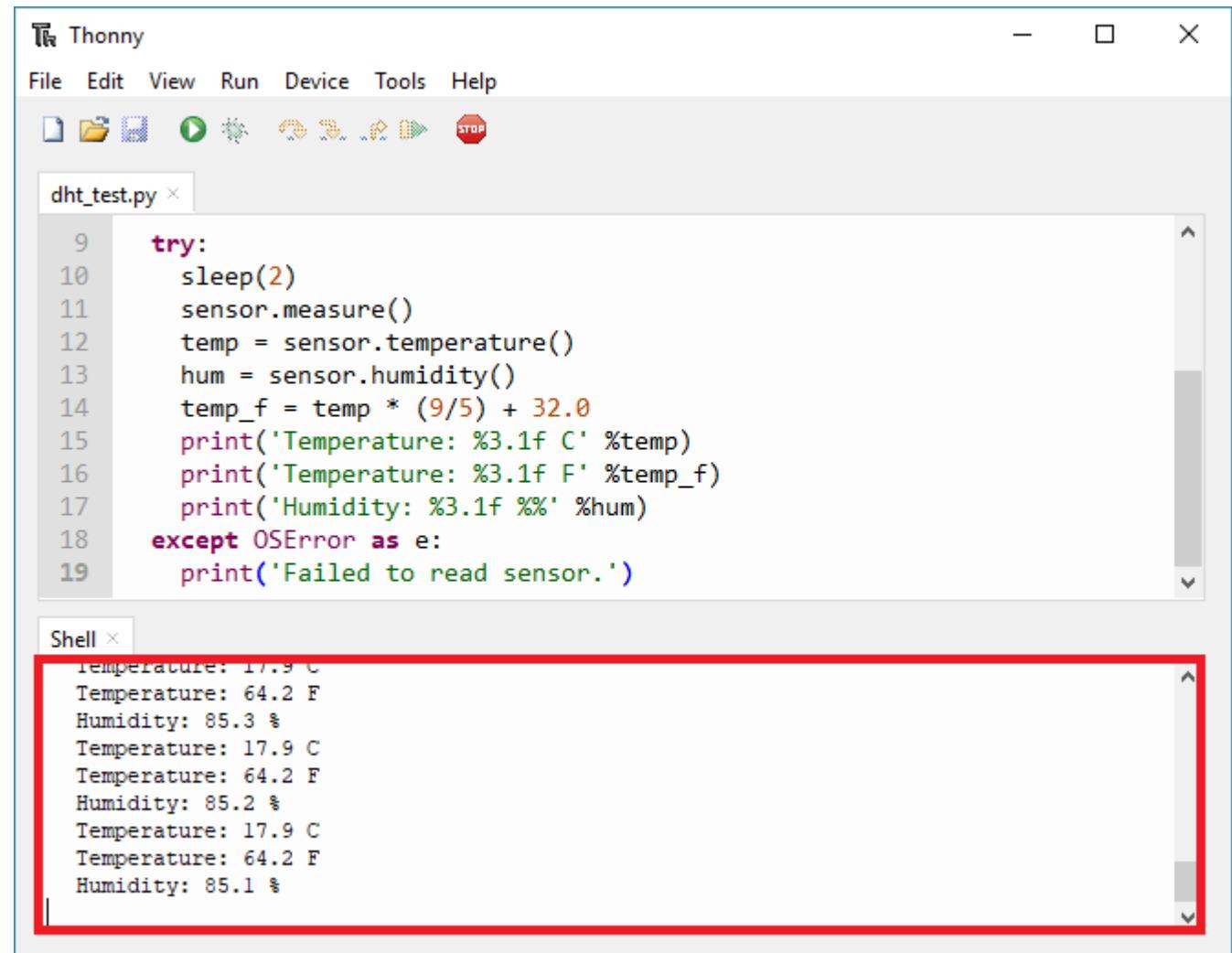
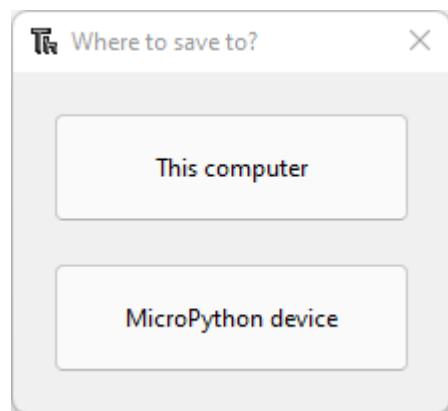
```
from machine import Pin
from time import sleep
import dht
sensor = dht.DHT22(Pin(14)) #sensor = dht.DHT11(Pin(14))
while True:
    try:
        sleep(2) # une seconde dans le cas de dh11
        sensor.measure() # on a besoin d'utiliser la methode measure dans l'objet sensor
        temp = sensor.temperature()
        hum = sensor.humidity()
        print('Temperature:',temp)
        print('Humidity:',hum)
    except OSError as e:
        print('Failed to read sensor.')
```

Note : try and except nous permet de continuer l'exécution du programme lorsqu'une exception se produit. Par exemple, lorsqu'une erreur se produit, l'exécution du code du bloc try est arrêtée et transférée vers le bloc except. Dans notre exemple, l'exception est particulièrement utile pour éviter que l'ESP32 ou le ESP8266 ne plante lorsque nous ne sommes pas en mesure de lire à partir du capteur.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython



```
dht_test.py x
9
10
11
12
13
14
15
16
17
18
19
try:
    sleep(2)
    sensor.measure()
    temp = sensor.temperature()
    hum = sensor.humidity()
    temp_f = temp * (9/5) + 32.0
    print('Temperature: %3.1f C' %temp)
    print('Temperature: %3.1f F' %temp_f)
    print('Humidity: %3.1f %%' %hum)
except OSError as e:
    print('Failed to read sensor.')

Shell x
temperature: 17.9 C
Temperature: 64.2 F
Humidity: 85.3 %
Temperature: 17.9 C
Temperature: 64.2 F
Humidity: 85.2 %
Temperature: 17.9 C
Temperature: 64.2 F
Humidity: 85.1 %
```

Chapitre 3 : Plateformes MicroPython

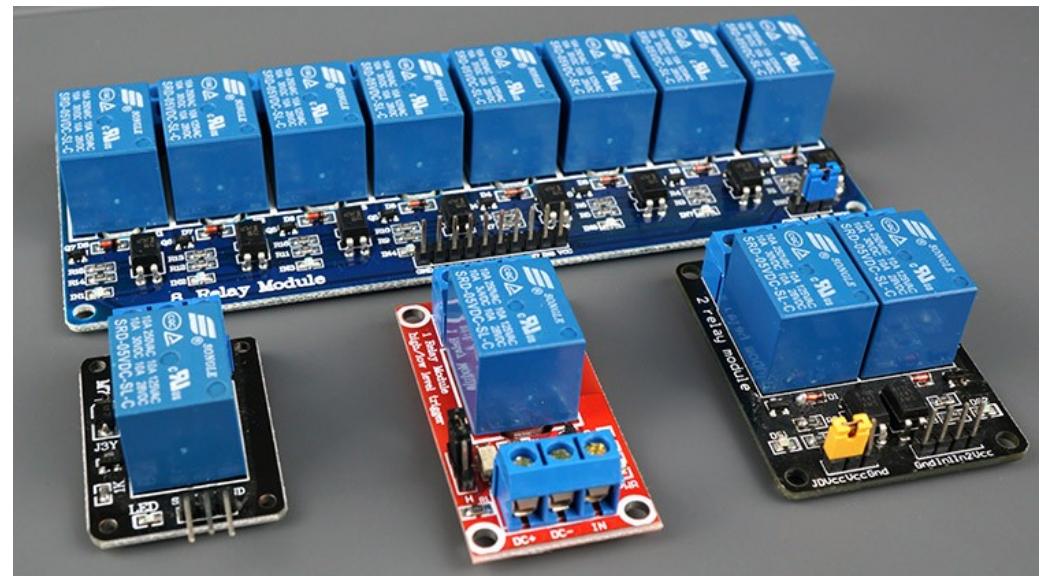
Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Exemple 4: Piloter des appareils électriques avec un relais

Un relais est un interrupteur à commande électrique et comme tout autre interrupteur, il peut être allumé ou éteint, laissant passer le courant ou non. Il peut être contrôlé avec des basses tensions, comme les 3,3 V fournis par les GPIO ESP32/ESP8266 et nous permet de contrôler des tensions élevées comme 12 V, 24 V ou la tension secteur.

Il existe différents modules de relais avec un nombre différent de canaux. Il existe des modules relais dont l'électroaimant peut être alimenté en 5V et en 3.3V. Les deux peuvent être utilisés avec l'ESP32 ou ESP8266 - vous pouvez utiliser la broche VIN (qui fournit 5 V) ou la broche 3,3 V.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter des appareils électriques avec un relais

Le fonctionnement du relais est basé sur l'attraction d'une pièce métallique (palette) par une bobine. Le mouvement de la palette ainsi produit se transmet à des contacts pour les ouvrir ou les fermer.

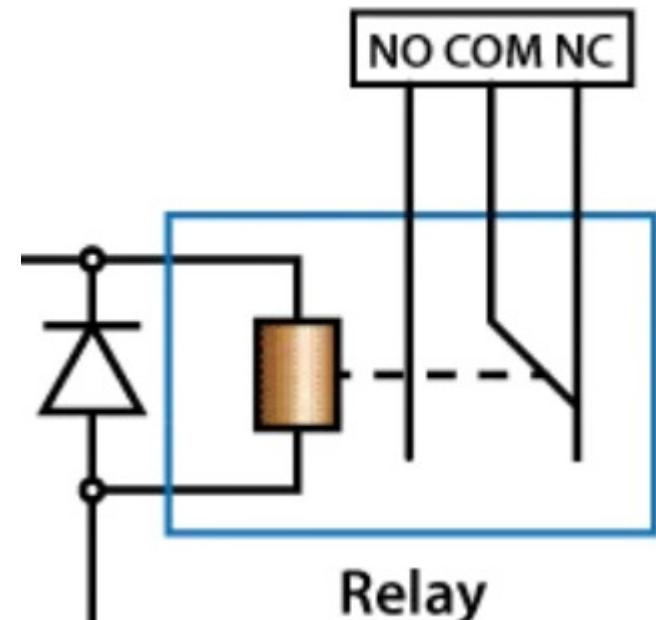
Relais à contacts normalement ouverts

Lorsque la bobine du relais n'est pas alimentée le contact de sortie est ouvert. La sortie est une recopie de l'entrée

Relais à contacts normalement fermés

Lorsque la bobine du relais est alimentée le contact de sortie est ouvert.

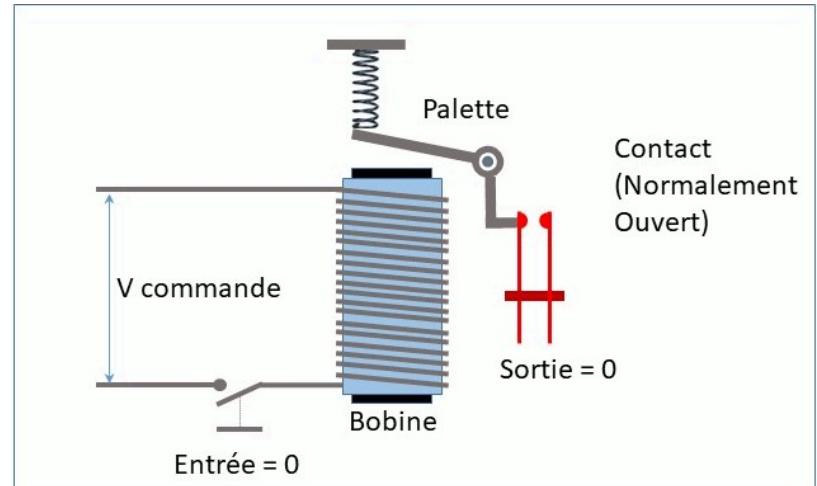
5V Relay Terminals and Pins



Programmer ESP32 avec micropython

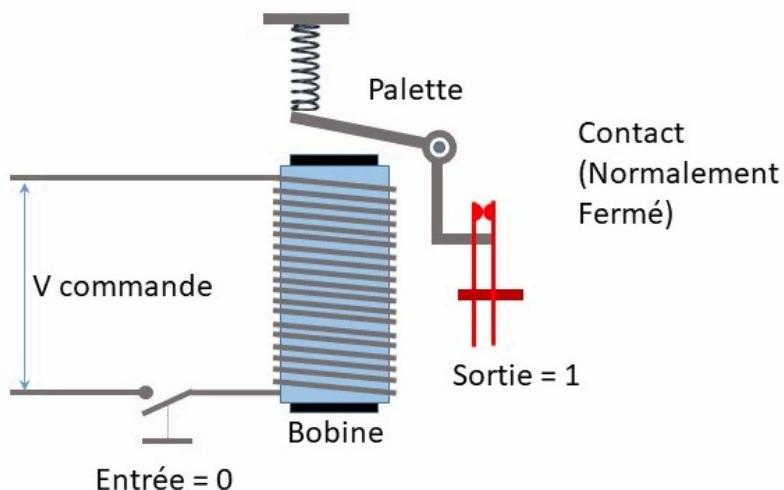
Piloter des appareils électriques avec un relais

Le fonctionnement du relais est basé sur l'attraction d'une pièce métallique (palette) par une bobine. Le mouvement de la palette ainsi produit se transmet à des contacts pour les ouvrir ou les fermer.



Relais à contacts normalement ouverts

Lorsque la bobine du relais n'est pas alimentée le contact de sortie est ouvert. La sortie est une recopie de l'entrée



Relais à contacts normalement fermés

Lorsque la bobine du relais est alimentée le contact de sortie est ouvert.

Programmer ESP32 avec micropython

Piloter des appareils électriques avec un relais

Le modèle SRD-05VDC-SL-C est conçu pour être alimenté en 5V, mais si vous utilisez une ESP32, la tension disponible sera 3.3V. Dans ce cas, le modèle SRD-03VDC-SL-C, qui s'alimente en 3.3V, est plus approprié.

Le modèle SRD-05VDC-SL-C comporte 3 broches pour piloter le relais :

- La broche de signal **S** pour piloter le relais, qu'on relie à une broche de l'ESP32.
- La broche d'alimentation que l'on relie au 5V de l'ESP32. (Ou au 3V3 si c'est le module SRD-03VDC-SL-C).
- La dernière broche, représentée par - est la masse, que l'on relie à une broche GND .

Il y a également un bornier à 3 plots à la sortie côté puissance

- **NC** : Normally Closed: Contact Normalement Fermé.
- **COM** : La broche du milieu, appelé commun (COM).
- **NO** : Normally Open : Contact Normallement Ouvert



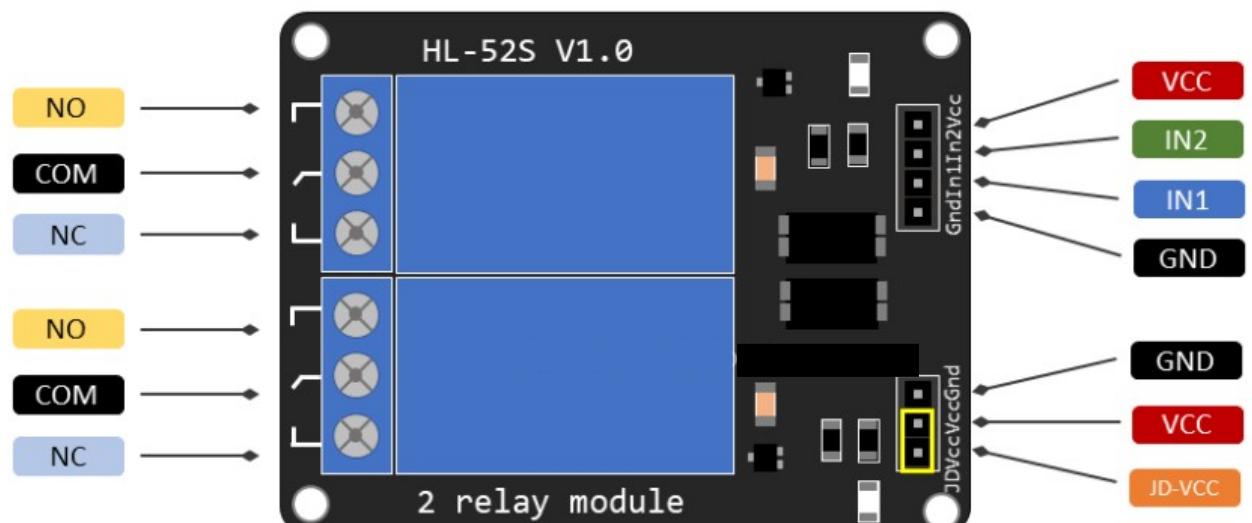
Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter des appareils électriques avec un relais

Module à deux relais:



Programmer ESP32 avec micropython

Piloter des appareils électriques avec un relais

Le module à deux relais possède deux connecteurs, chacun avec trois prises : commun (**COM**), normalement fermé (**NC**) et normalement ouvert (**NO**).

- **COM** : connectez le courant que vous souhaitez contrôler (tension secteur).
- **NC (Normalement fermé)** : la configuration normalement fermée est utilisée lorsque vous souhaitez que le relais soit fermé par défaut.
- **NO (Normalement ouvert)** : la configuration normalement ouverte est utilisée lorsque vous souhaitez que le relais soit ouvert par défaut.

▪ Configuration normalement ouverte (NO) :

- Signal HIGH – le courant circule
- Signal LOW – le courant ne circule pas

▪ Configuration normalement fermée (NC) :

- Signal HIGH – le courant ne circule pas
- Signal LOW – courant en circulation

Vous devez utiliser une configuration normalement ouverte lorsque le courant doit circuler la plupart du temps et que vous ne voulez l'arrêter qu'occasionnellement.

Utilisez une configuration normalement fermée lorsque vous souhaitez que le courant circule occasionnellement (par exemple, allumez une lampe de temps en temps).

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

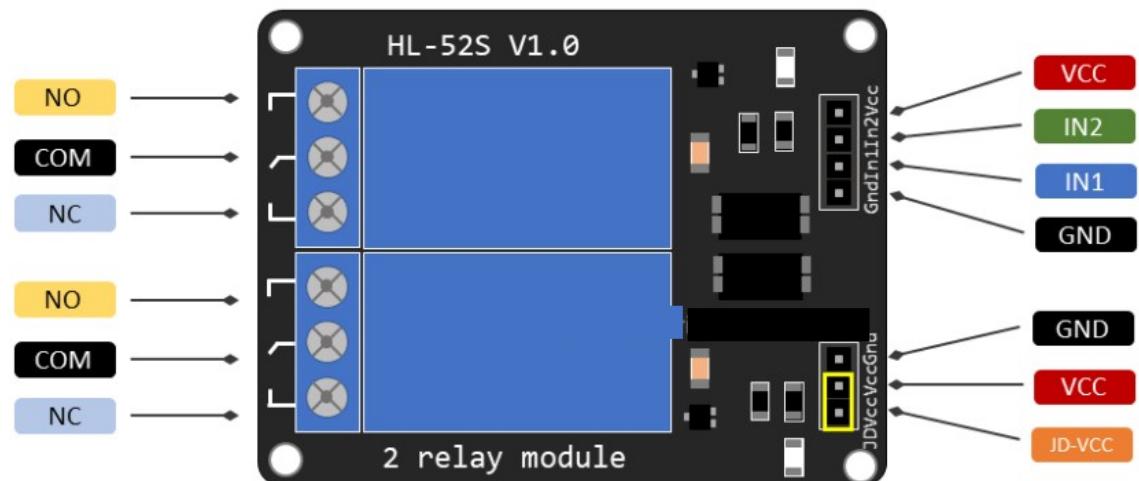
Programmer ESP32 avec micropython

Piloter des appareils électriques avec un relais

Module à deux relais:

La broche **JD-VCC** alimente l'électroaimant du relais. Notez que le module est doté d'un cavalier (**Jumper**) reliant les broches VCC et JD-VCC. Cela signifie que l'électroaimant du relais est directement alimenté par la broche d'alimentation de l'ESP, de sorte que le module de relais et les circuits de l'ESP ne sont pas physiquement isolés l'un de l'autre.

Sans jumper vous devez fournir une source d'alimentation indépendante pour alimenter l'électroaimant du relais via le JD-VCC pin. Cette configuration isole physiquement les relais de l'ESP grâce à l'optocoupleur intégré au module, ce qui évite d'endommager l'ESP en cas de pics électriques.



IN1 et IN2 : signaux de commande de relais

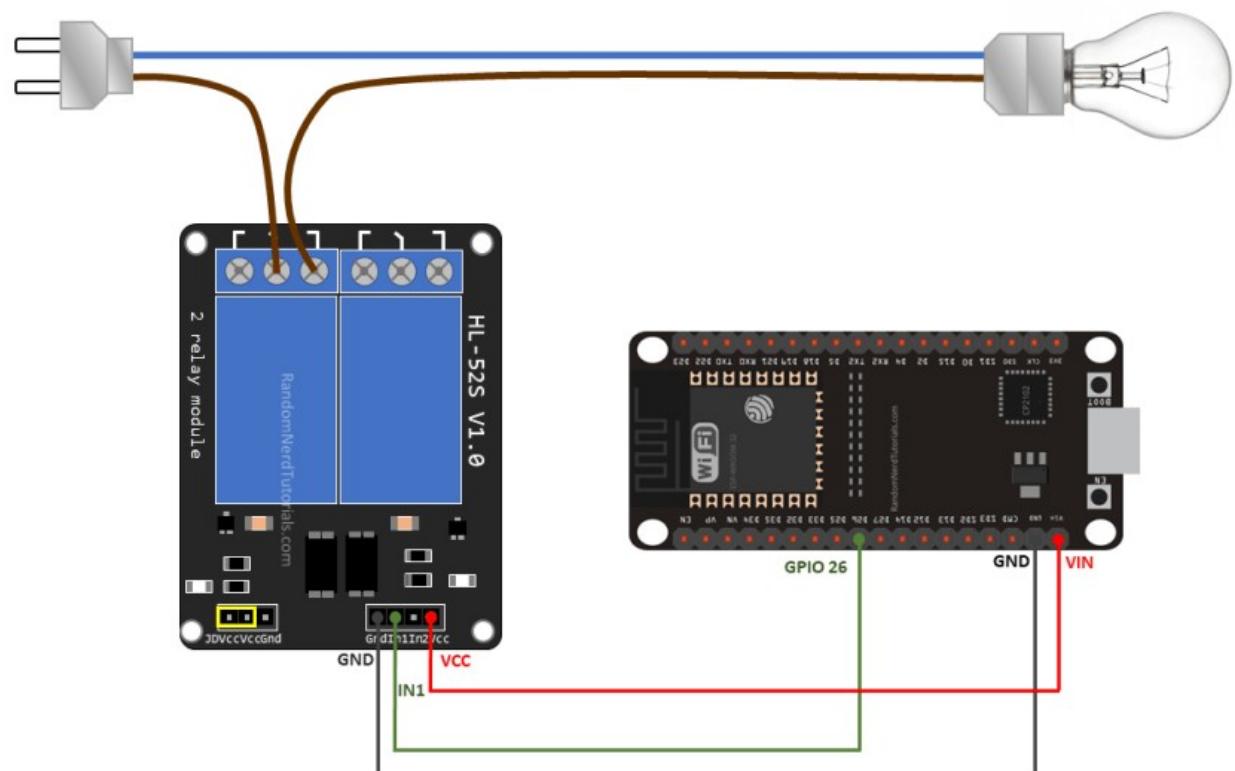
Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter des appareils électriques avec un relais

```
from machine import Pin
from time import sleep
# ESP32 GPIO 26
relay = Pin(26, Pin.OUT)
while True:
    # RELAY ON : allumer la lampe pendant 10
seconds.
    relay.value(1) # envoyer un signal high
    sleep(10)
    # RELAY OFF : éteindre la lampe pendant 10 secondes
    relay.value(0) # envoyer un signal low
    sleep(10)
```

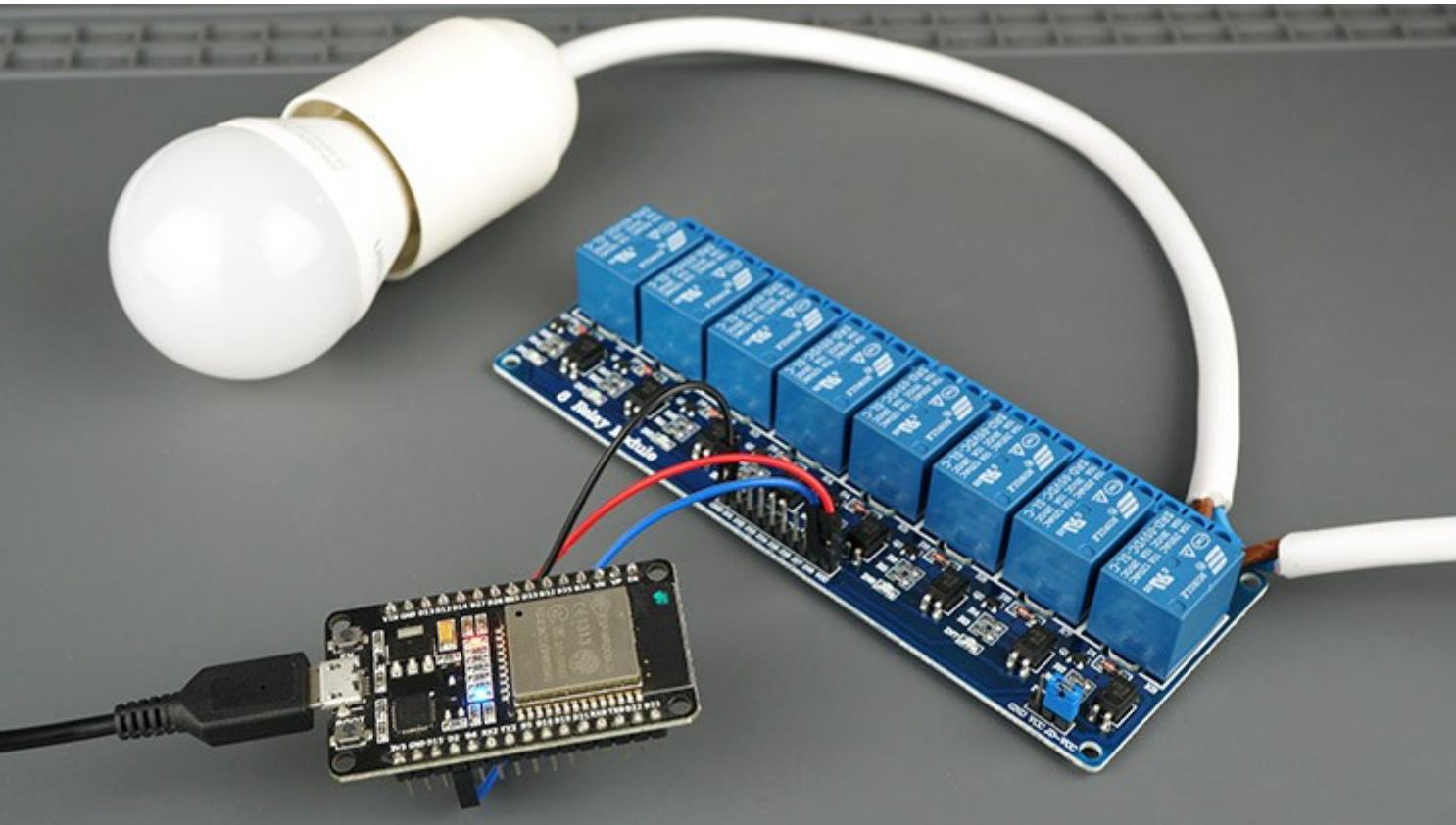


Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter des appareils électriques avec un relais

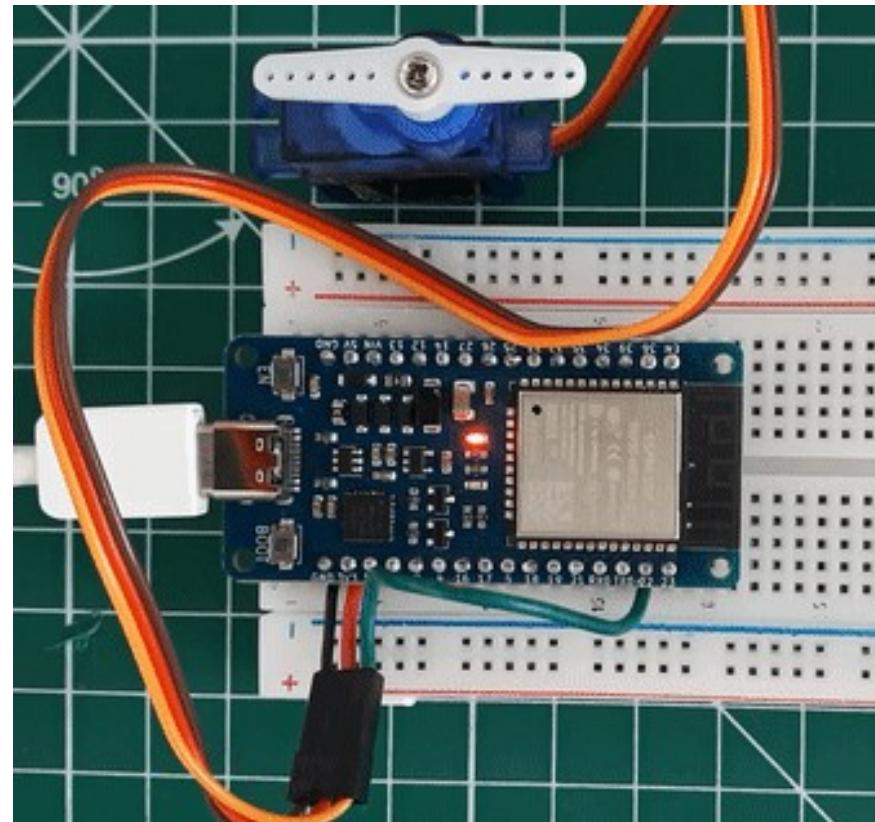


Programmer ESP32 avec micropython

Exemple 5: Piloter les servo-moteurs

Les servomoteurs, souvent abrégés en « servo », sont une forme spéciale de moteur qui peut être fixée à une position spécifique avec une grande précision. Cette position est maintenue jusqu'à ce qu'une nouvelle instruction soit donnée.

Ils peuvent tourner de 0 degré à 180 degrés.



Programmer ESP32 avec micropython

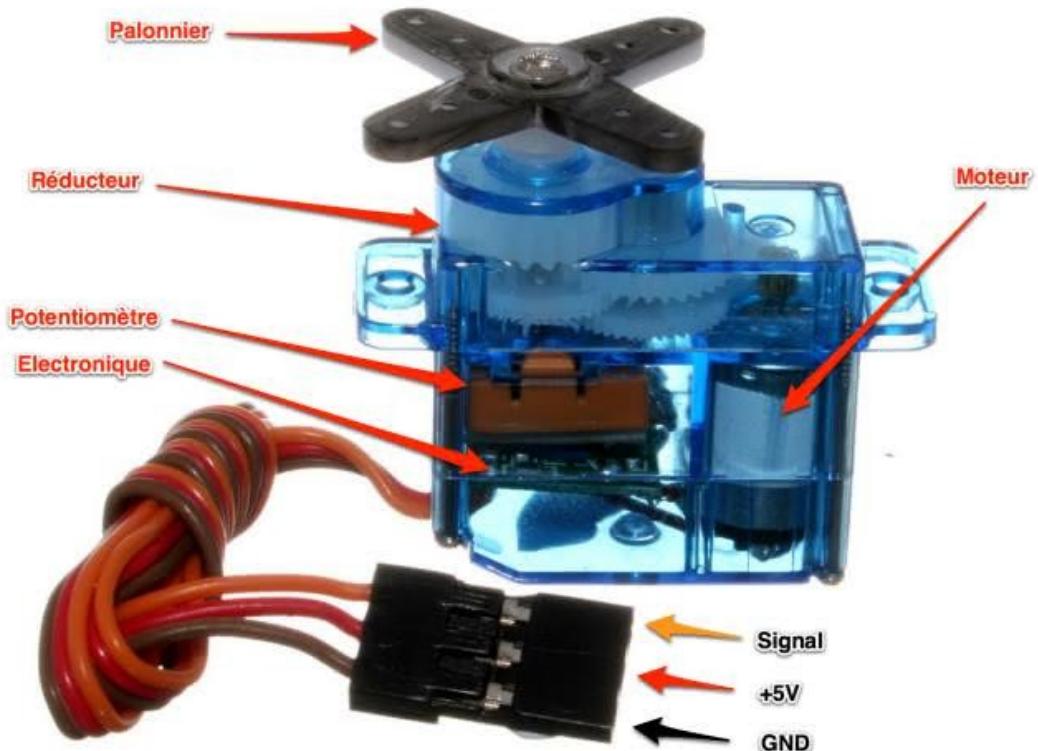
Piloter les servo-moteurs

Fonctionnement d'un SG90:

Un petit moteur CC est relié à un potentiomètre par l'intermédiaire d'un circuit électronique, ce qui permet de réguler finement la vitesse du moteur selon la position du potentiomètre.

Une série d'engrenages est attachée à l'axe de sortie du moteur, afin de multiplier le couple tout en réduisant sa vitesse de rotation.

Lorsque le moteur tourne, les engrenages entraînent le mouvement du bras qui à son tour actionne le potentiomètre. Si le mouvement s'arrête, le circuit électronique ajuste en continu la vitesse du moteur pour maintenir le potentiomètre et donc le bras à la même position.



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

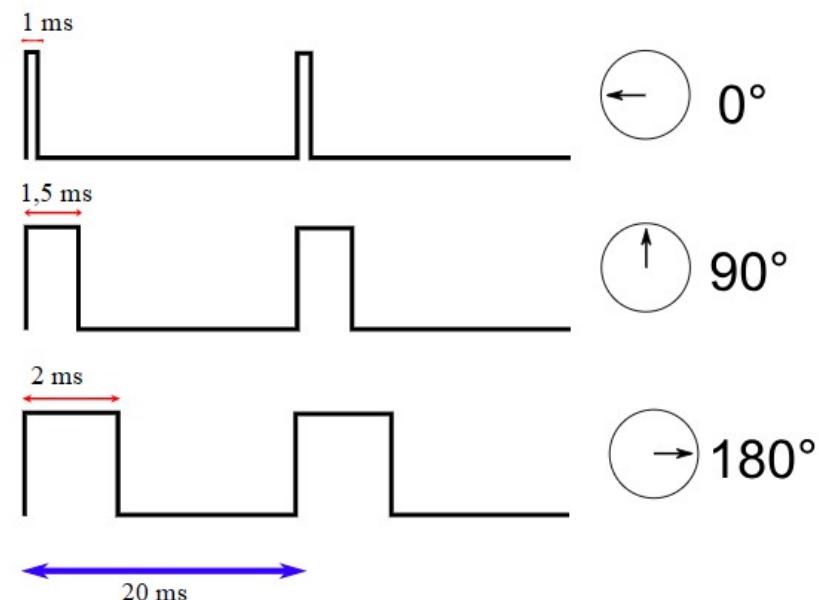
Programmer ESP32 avec micropython

Piloter les servo-moteurs

Fonctionnement d'un SG90:

Ce petit servomoteur est commandé en utilisant un signal modulé en largeur d'impulsion (PWM) de 50 Hz de fréquence, soit une impulsion toutes les 20ms. La position du servomoteur est déterminée par la durée des impulsions, généralement variant entre

Servomoteur SG90	Couleur du fil	ESP32
GND	Marron	GND
5V	Rouge	5V ou 3V3
Signal PWM	Orange	GPIO22



Chapitre 3 : Plateformes MicroPython

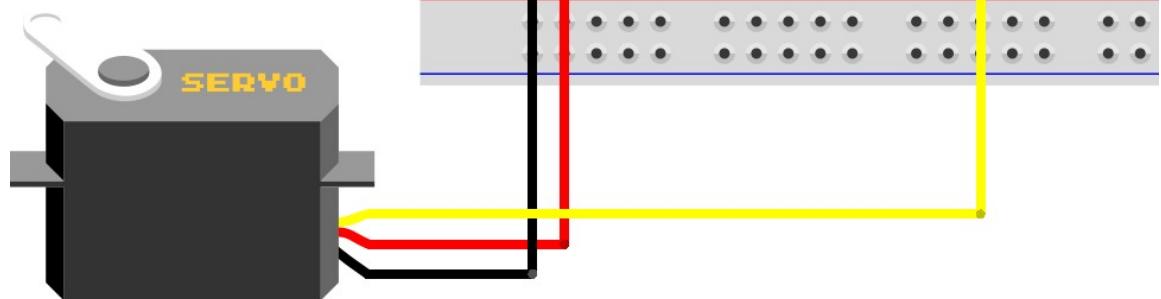
Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter les servo-moteurs

Brochage SG90 avec ESP32:

Servomoteur SG90	Couleur du fil	ESP32
GND	Marron	GND
5V	Rouge	5V ou 3V3
Signal PWM	Orange	GPIO22



Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter les servo-moteurs

```
from machine import Pin, PWM
from time import sleep
sg90 = PWM(Pin(22, Pin.OUT))
sg90.freq(50)

# pos min correspond à 0.5ms/20ms = 0.025 = 2.5% duty cycle
# pos max correspond à 2.4ms/20ms = 0.12 = 12% duty cycle

#0.025*1024=25.6
# 0.12*1024=122.88

while True:
    sg90.duty(26)
    sleep(1)
    sg90.duty(123)
    sleep(1)
```

```
from machine import Pin, PWM
from time import sleep
x = PWM(Pin(22))
x.freq(50)

while True:
    for position in range(1000, 9000, 50):
        pwm.duty_u16(position)
        sleep(0.1)
    for position in range(9000, 1000, -50):
        pwm.duty_u16(position)
        sleep(0.1)
```

Programmer ESP32 avec micropython

Piloter les servo-moteurs

```
from machine import Pin, PWM
from time import sleep
sg90 = PWM(Pin(22, Pin.OUT))
sg90.freq(50)
# pos min correspond à 0.5ms/20ms = 0.025 = 2.5% duty cycle
# pos max correspond à 2.4ms/20ms = 0.12 = 12% duty cycle
#0.025*1024=25.6
# 0.12*1024=122.88
while True:
    sg90.duty(26)
    sleep(1)
    sg90.duty(123)
    sleep(1)
```

```
from machine import Pin, PWM
from time import sleep
x= Pin(22, Pin.OUT)
servo = PWM(x,freq=50)
servo.duty(100)
sleep(1)
servo.duty(30)
sleep(1)
servo.duty(75)
sleep(1)
```

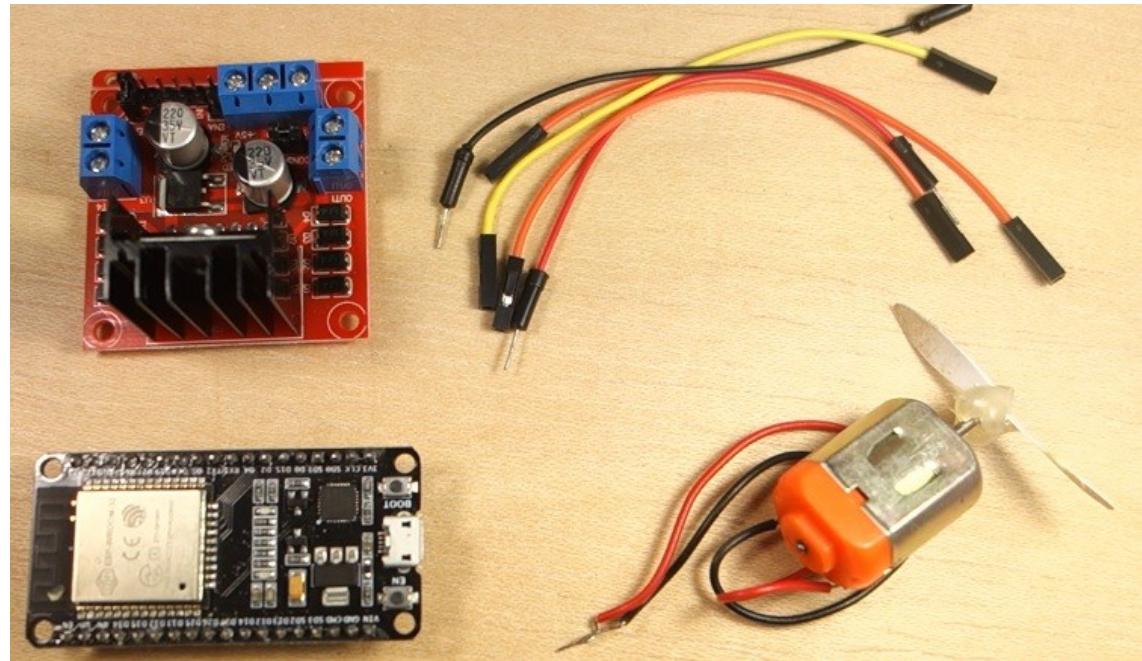
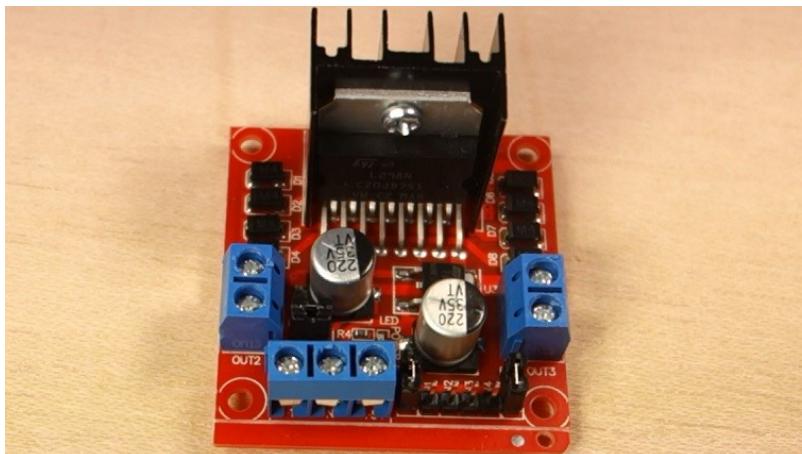
Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Exemple 6: Piloter DC motors et driver de moteurs L298N

Nous allons utiliser le pilote de moteur L298N qui peut gérer jusqu'à 3A à 35V. De plus, il nous permet de piloter deux moteurs à courant continu simultanément, ce qui est parfait pour construire un robot.



Chapitre 3 : Plateformes MicroPython

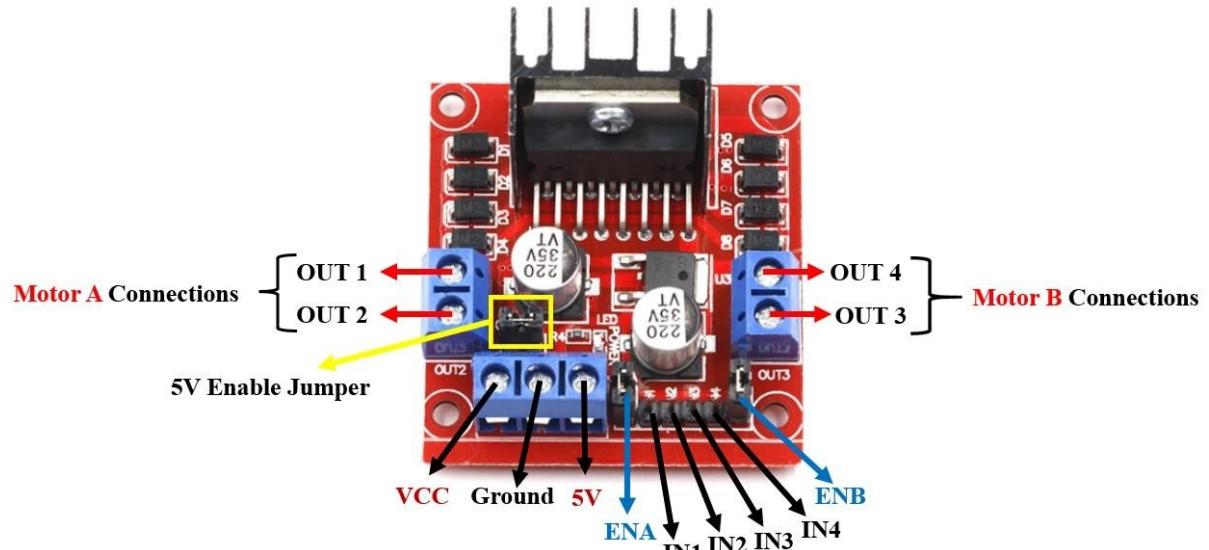
Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter DC motors et driver de moteursL298N

Le pilote de moteur dispose d'un bloc de jonction de deux côtés pour chaque moteur. OUT1 et OUT2 à gauche et OUT3 et OUT4 à droite.

- **OUT1** : Moteur à courant continu A + borne
 - **OUT2** : Moteur à courant continu A - borne
 - **OUT3** : Moteur à courant continu B + borne
 - **OUT4** : Moteur à courant continu B - borne
- **+12V** : l'alimentation
- **GND** : GND
- **+5V** : fournir 5V si le cavalier est retiré. Agit comme une sortie 5V si le cavalier est en place
- **jumper**: cavalier en place - utilise l'alimentation du moteur pour alimenter la puce. Cavalier retiré : vous devez fournir 5V à la borne +5V. Si vous fournissez plus de 12V, vous devez retirer le cavalier



- **IN1** : Entrée 1 pour le moteur A
- **IN2** : Entrée 2 pour le moteur A
- **IN3** : Entrée 1 pour le moteur B
- **IN4** : Entrée 2 pour le moteur B
- **EN1** : Broche d'activation pour le moteur A
- **EN2** : Broche d'activation pour le moteur B

Programmer ESP32 avec micropython

Piloter DC motors et driver de moteursL298N

Les broches d'activation sont comme un interrupteur ON et OFF pour vos moteurs. Par exemple:

- Si vous envoyez un **signal HIGH** à la broche d'activation 1, le moteur A est prêt à être contrôlé et à la vitesse maximale ;
- Si vous envoyez un **signal LOW** à la broche d'activation 1, le moteur A s'éteint ;
- Si vous envoyez un **signal PWM**, vous pouvez contrôler la vitesse du moteur. La vitesse du moteur est proportionnelle au rapport cyclique. Cependant, notez que pour les petits cycles de service, les moteurs peuvent ne pas tourner et émettre un bourdonnement continu.

Les broches d'entrée contrôlent la direction de rotation des moteurs.

L'entrée 1 et l'entrée 2 contrôlent le moteur A, et les entrées 3 et 4 contrôlent le moteur B.

- Si vous appliquez LOW à l'entrée 1 et HIGH à l'entrée 2, le moteur tournera vers l'avant ;
- Si vous appliquez la puissance dans l'autre sens : HIGH à l'entrée 1 et LOW à l'entrée 2, le moteur tournera vers l'arrière.

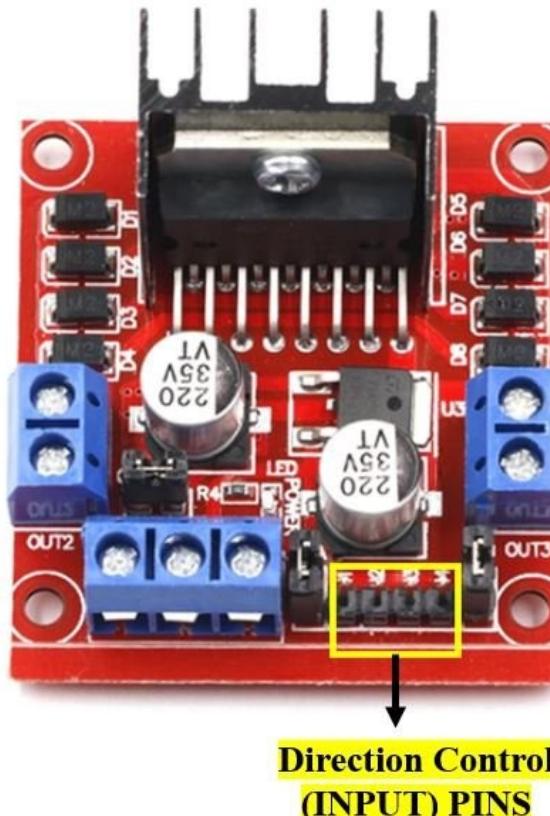
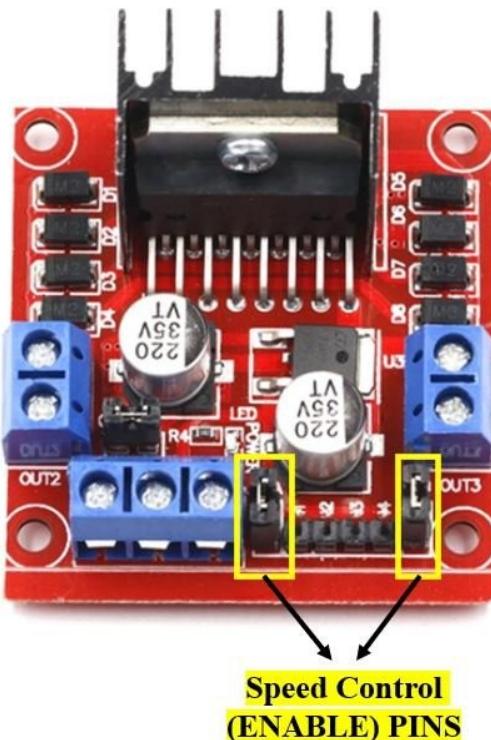
Le moteur B peut être contrôlé en utilisant la même méthode, mais en appliquant HIGH ou LOW à l'entrée 3 et à l'entrée 4.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter DC motors et driver de moteursL298N



Chapitre 3 : Plateformes MicroPython

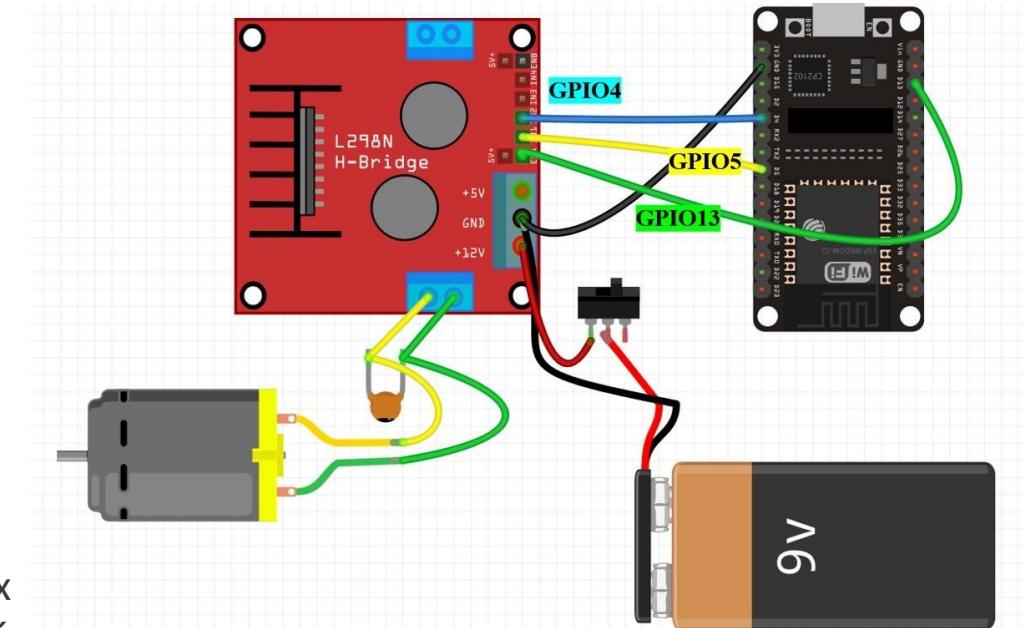
Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter DC motors et driver de moteursL298N

DIRECTION	ENTRÉE 1	ENTRÉE 2	ENTRÉE 3	ENTRÉE 4
En avant	0	1	0	1
En arrière	1	0	1	0
Droite	0	1	0	0
Gauche	0	0	0	1
Arrêter	0	0	0	0

Il est recommandé de souder un condensateur céramique de 0,1 uF aux bornes positive et négative du moteur à courant continu, comme indiqué sur le schéma, pour aider à lisser les pics de tension.
(Remarque : les moteurs fonctionnent également sans le condensateur.)



Programmer ESP32 avec micropython

Piloter DC motors et driver de moteursL298N

Maintenant que nous avons connecté tous nos équipements ensemble, écrivons le code microPython pour contrôler le moteur à cc.

- 1.Nous allons d'abord créer un module **dcmotor** pour rendre le code plus facile à implémenter à l'aide d'une seule commande.
- 2.À l'intérieur du module, nous allons créer une classe : **DCMotor**.
- 3.Des méthodes seront en outre affectées à la classe DCMotor.
- 4.Nous allons initialiser le moteur avec trois paramètres :
 - Broche 1 (IN1)
 - Broche 2 (IN2)
 - activer (ENA)

Les deux broches agiront comme des broches de sortie et l'activation agira comme une broche pwm.

Nous allons créer un objet DCMotor appelé 'motor' et l'initialiser de la manière suivante :
motor = DCMotor (Pin1, Pin2, enable)

- 5.Enfin, nous utiliserons les méthodes de la classe DCMotor à savoir : **forward()**, **backwards()** et **stop()** sur l'objet DCMotor 'motor' pour contrôler le moteur DC.
(On peut télécharger la bibliothèque du module dcmotor du net, l'enregistrer sous dcmotor.py et la télécharger dans l'esp32 avec thonny.)

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter DC motors et driver de moteurs L298N

```
from dcmotor import DCMotor
from machine import Pin, PWM
from time import sleep
frequency = 15000
pin1 = Pin(5, Pin.OUT)
pin2 = Pin(4, Pin.OUT)
enable = PWM(Pin(13), frequency)
dc_motor = DCMotor(pin1, pin2, enable)
dc_motor = DCMotor(pin1, pin2, enable, 350,
1023)
```

```
dc_motor.forward(50)
sleep(10)
dc_motor.stop()
sleep(10)
dc_motor.backwards(100)
sleep(10)
dc_motor.forward(60)
sleep(10)
dc_motor.stop()
```

Faire tourner le moteur vers l'arrière à la vitesse maximale (100), puis tourne vers l'avant à une vitesse de 60, puis s'arrête

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny
Section 2 : ESP8266 NodeMCU
Section 3 : ESP32
Section 4 : PI Pico

Programmer ESP32 avec micropython

Piloter DC motors et driver de moteurs L298N

```
from dcmotor import DCMotor
from machine import Pin, PWM
from time import sleep
frequency = 15000
pin1 = Pin(5, Pin.OUT)
pin2 = Pin(4, Pin.OUT)
enable = PWM(Pin(13), frequency)
dc_motor = DCMotor(pin1, pin2, enable)
dc_motor = DCMotor(pin1, pin2, enable, 350,
1023)
```

```
dc_motor.forward(50)
sleep(10)
dc_motor.stop()
sleep(10)
dc_motor.backwards(100)
sleep(10)
dc_motor.forward(60)
sleep(10)
dc_motor.stop()
```

Faire tourner le moteur vers l'arrière à la vitesse maximale (100), puis tourne vers l'avant à une vitesse de 60, puis s'arrête.

Chapitre 3 : Plateformes MicroPython

Section 1 : IDE Thonny

Section 2 : ESP8266 NodeMCU

Section 3 : ESP32

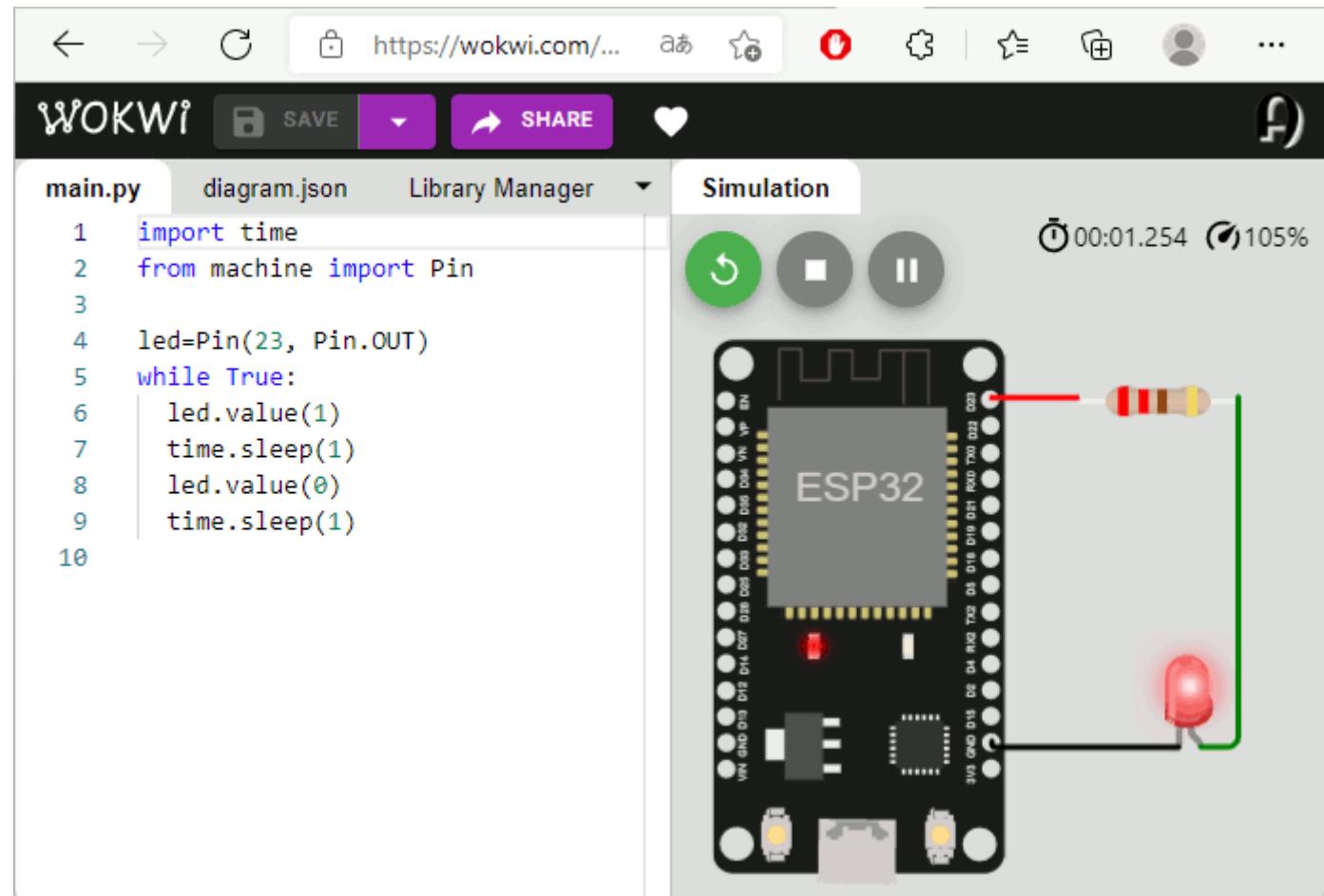
Section 4 : PI Pico

❑ Simulateur:

Rendez-vous sur le site:

[Wokwi - Online ESP32, STM32, Arduino Simulator](https://wokwi.com/)

pour utiliser un simulateur en ligne pour les cartes ESP32,
STM32, Arduino, Pi Pico



Programmation Python pour l'embarqué

La fin du chapitre

Fin chapitre 3