

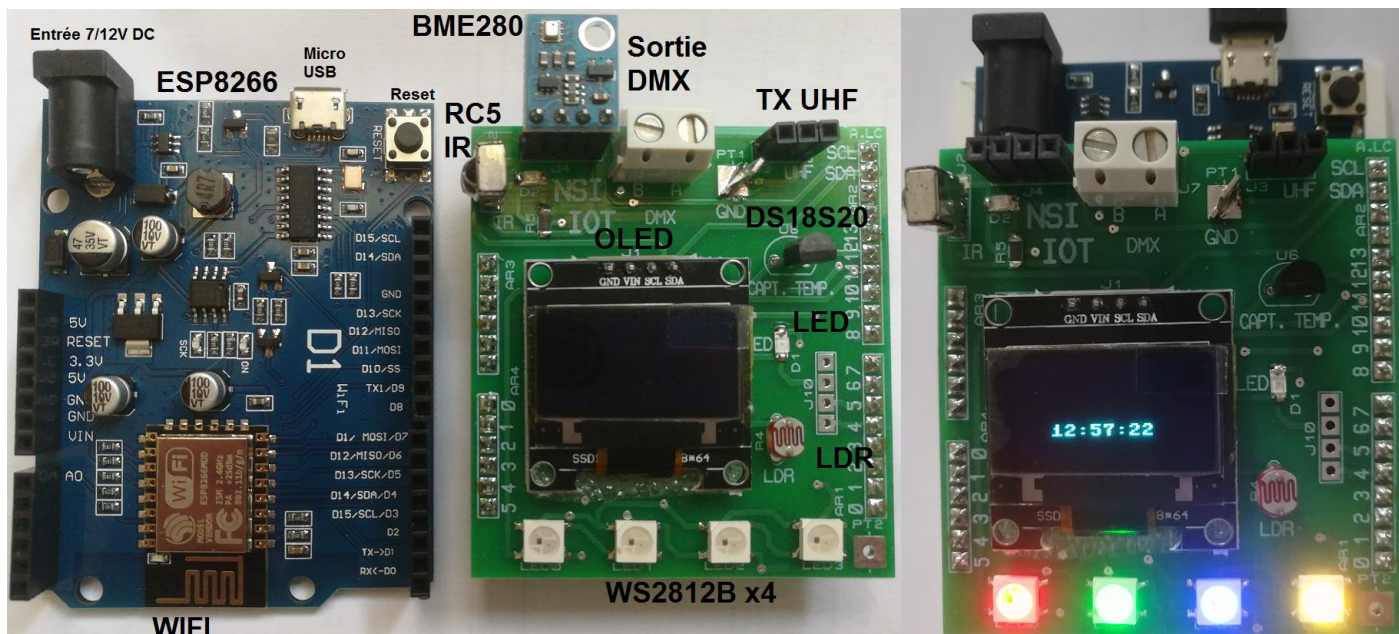


Carte IOT

(Description technique)



LE CREN Anthony



Description des périphériques montés sur la carte Shield

ESP32	ESP8266	Composant	Rôle
12	0	Logique (OUT)	Led rouge
17	2	Logique (OUT)	Sortie RS485
21	4	Logique	SDA : Afficheur OLED
22	5	Logique	SCL : Afficheur OLED
19	12	Logique	DS18S20
23	13	Logique (IN)	VS1838B
18	14	Logique (OUT)	WS2812B
5	15	Logique (OUT)	Emetteur UHF
35	A0	Analogique (IN)	Capteur LDR

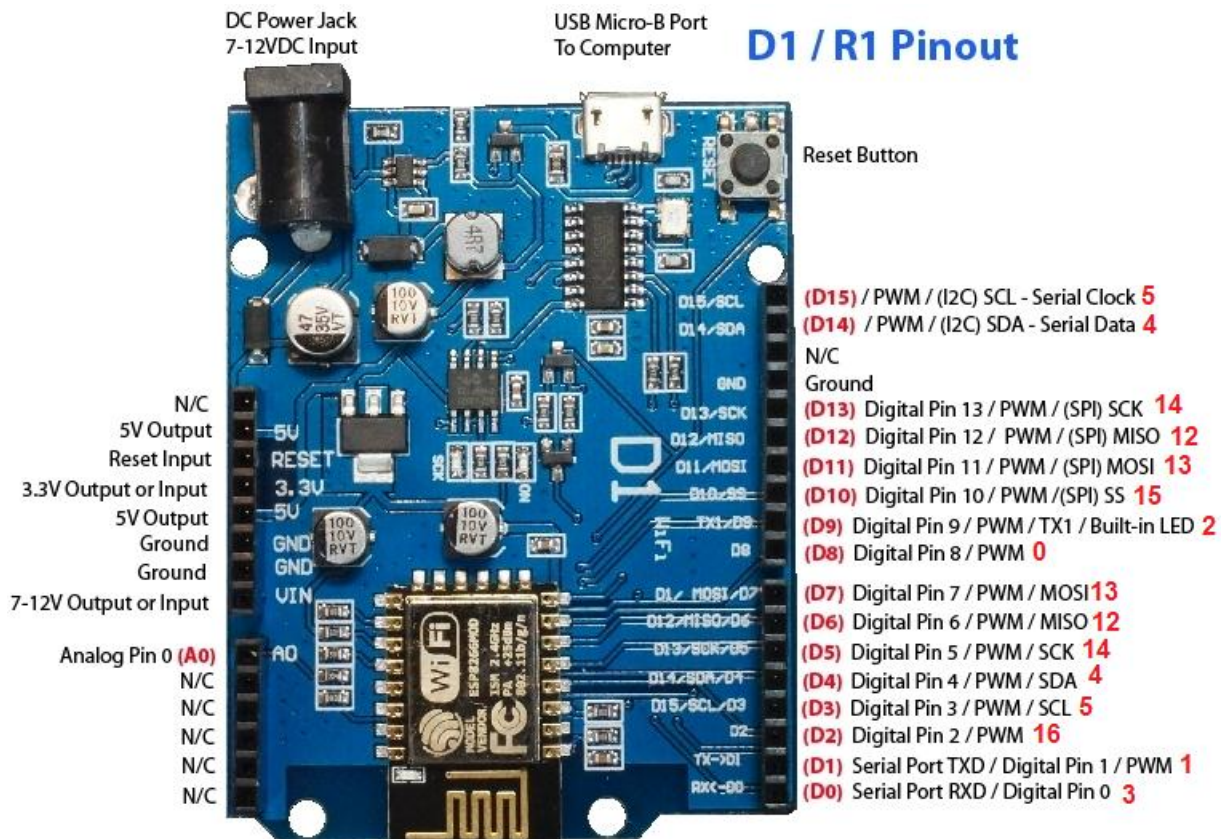
DMX : ESP8266 : UART1 ou ESP32 : UART2

Configuration du shield en fonction de la carte microcontrôleur :

	J5	J6	J8	J9
ESP8266	Ouvert	Fermé	Ouvert	Fermé
ESP32	Fermé	Ouvert	Fermé	Ouvert

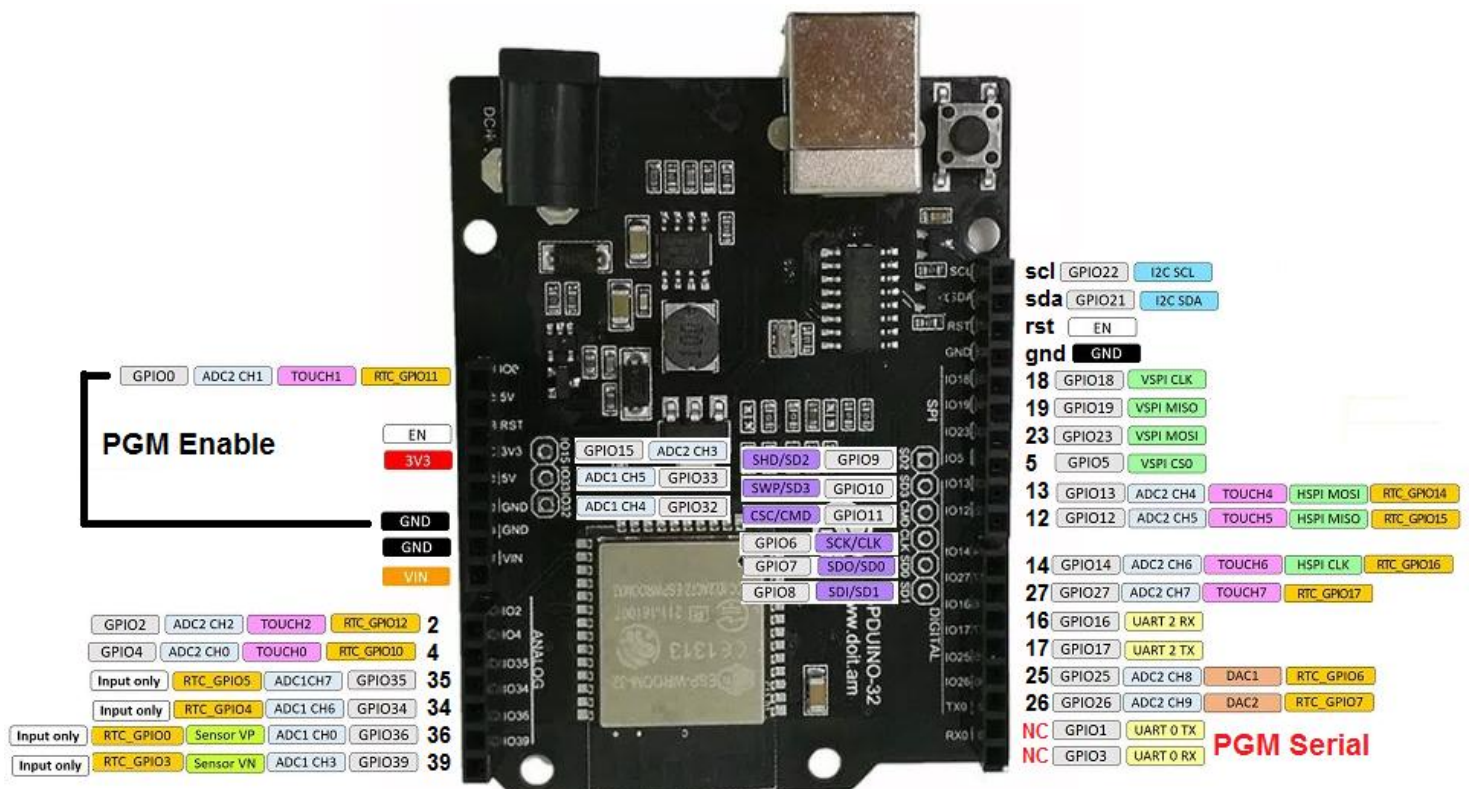
ESP8266

D1 / R1 Pinout



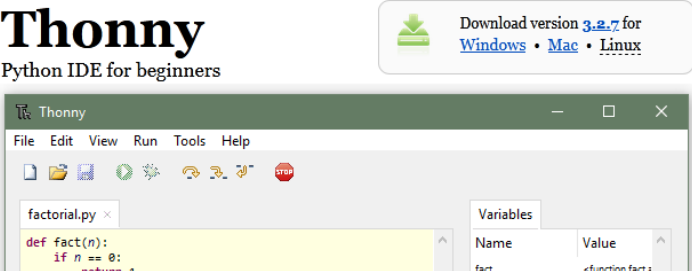
only pins 0, 2, 4, 5, 12, 13, 14, 15, and 16 can be used.

ESP32



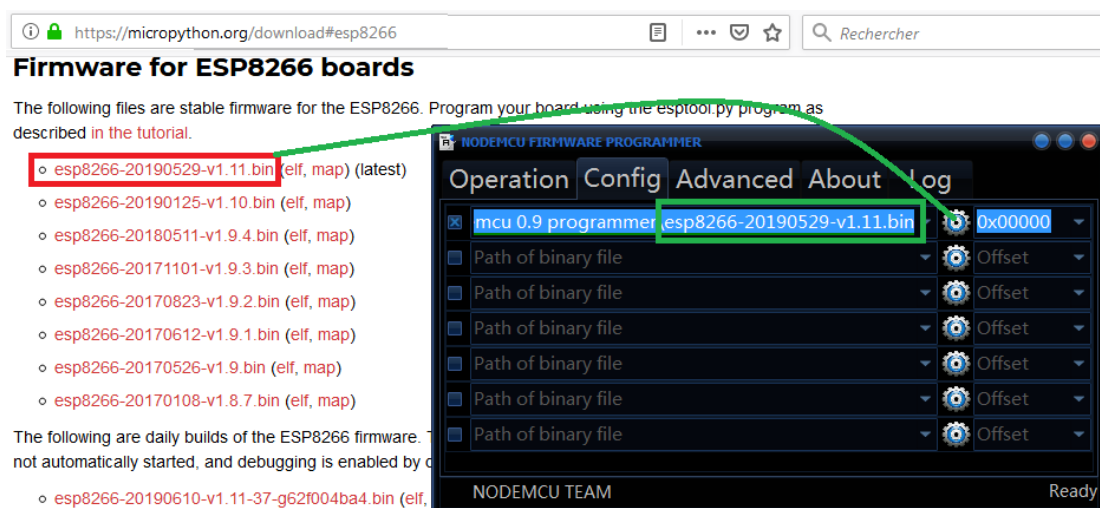
Annexe 1 : Implantation du firmware Micropython dans l'ESP8266

<https://buildmedia.readthedocs.org/media/pdf/pycopy/latest/pycopy.pdf>

Editeur	Firmware
https://thonny.org/ (Windows, linux installation via pip) ou https://codewith.mu/ (Windows, esp8266 non supporté sous linux)	https://micropython.org/download/esp8266/
 <p>Thonny Python IDE for beginners</p>	<p>Stable firmware, 1M or more of flash</p> <p>The following files are stable firmware for the ESP8266. Program yo described in the tutorial.</p> <ul style="list-style-type: none"> ◦ esp8266-20191220-v1.12.bin (elf, map) (latest) ◦ esp8266-20190529-v1.11.bin (elf, map)

Flasher le firmware dans le 8266 (<https://micropython.org/download/esp8266/>)

Avec nodemcu firmware programmer disponible ici : <https://github.com/f4goh/NSI>



<https://micropython.org/download#esp8266>

Firmware for ESP8266 boards

The following files are stable firmware for the ESP8266. Program your board using the esptool.py program as described in the tutorial.

- **esp8266-20190529-v1.11.bin** (elf, map) (latest)
- esp8266-20190125-v1.10.bin (elf, map)
- esp8266-20180511-v1.9.4.bin (elf, map)
- esp8266-20171101-v1.9.3.bin (elf, map)
- esp8266-20170823-v1.9.2.bin (elf, map)
- esp8266-20170612-v1.9.1.bin (elf, map)
- esp8266-20170526-v1.9.bin (elf, map)
- esp8266-20170108-v1.8.7.bin (elf, map)

The following are daily builds of the ESP8266 firmware. They are not automatically started, and debugging is enabled by default.

- esp8266-20190610-v1.11-37-g62f004ba4.bin (elf, map)

NODEMCU FIRMWARE PROGRAMMER

Operation Config Advanced About Log

mcu 0.9 programmer **esp8266-20190529-v1.11.bin** 0x00000

Path of binary file Offset

Path of binary file Offset

Path of binary file Offset

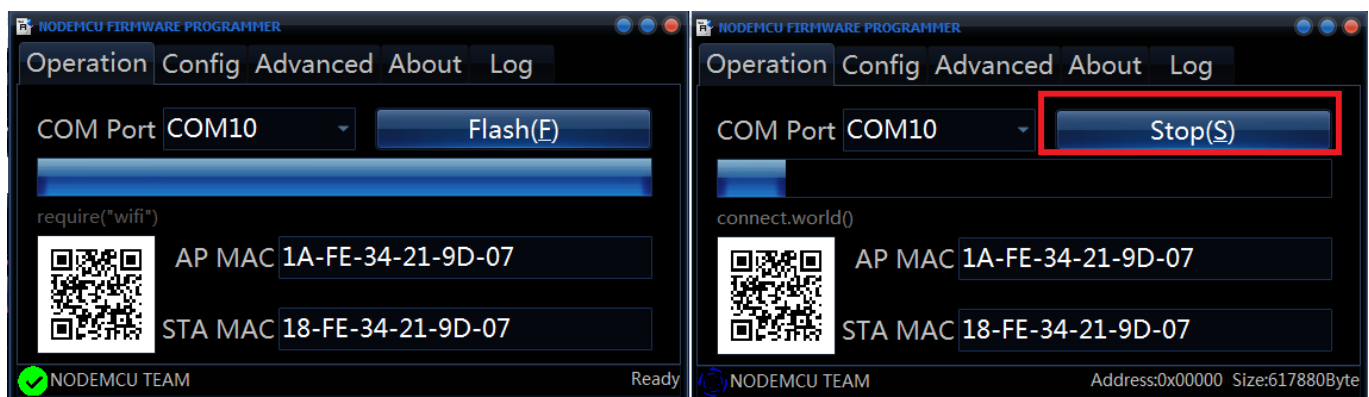
Path of binary file Offset

Path of binary file Offset

Path of binary file Offset

NODEMCU TEAM Ready

Puis bouton Flash.



NODEMCU FIRMWARE PROGRAMMER

Operation Config Advanced About Log

COM Port COM10 Flash(F)

require("wifi")

AP MAC 1A-FE-34-21-9D-07

STA MAC 18-FE-34-21-9D-07

✓ NODEMCU TEAM Ready

NODEMCU FIRMWARE PROGRAMMER

Operation Config Advanced About Log

COM Port COM10 Stop(S)

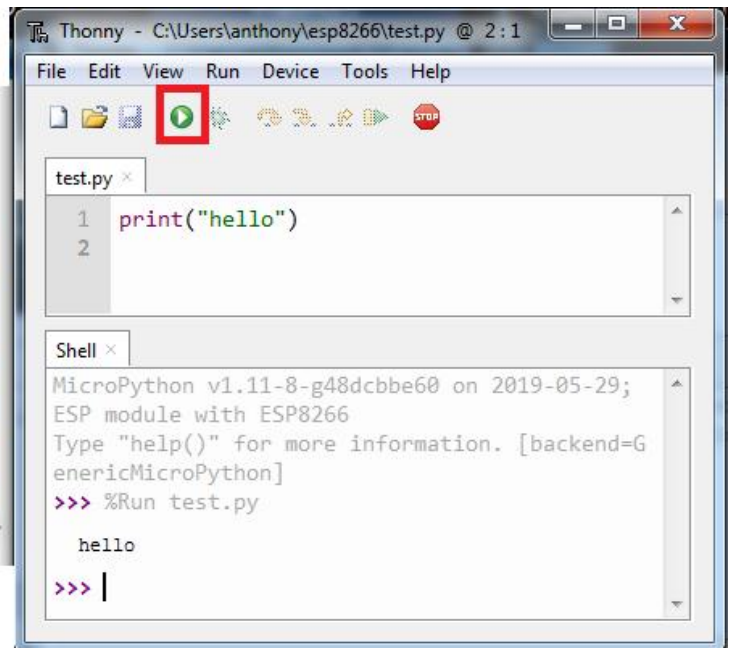
connect.world()

AP MAC 1A-FE-34-21-9D-07

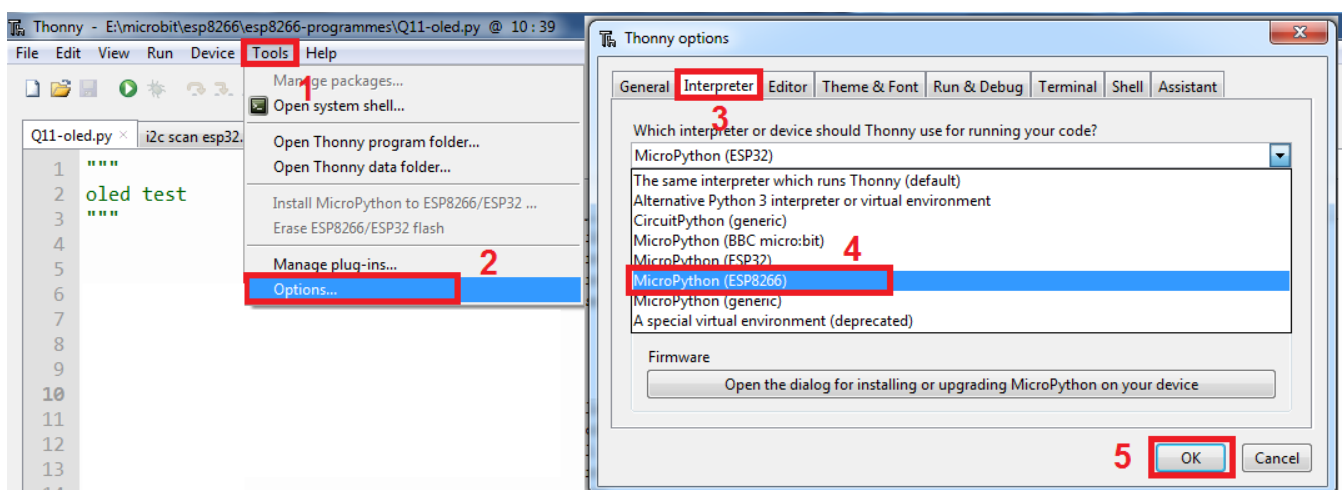
STA MAC 18-FE-34-21-9D-07

⚙️ NODEMCU TEAM Address:0x00000 Size:617880Byte

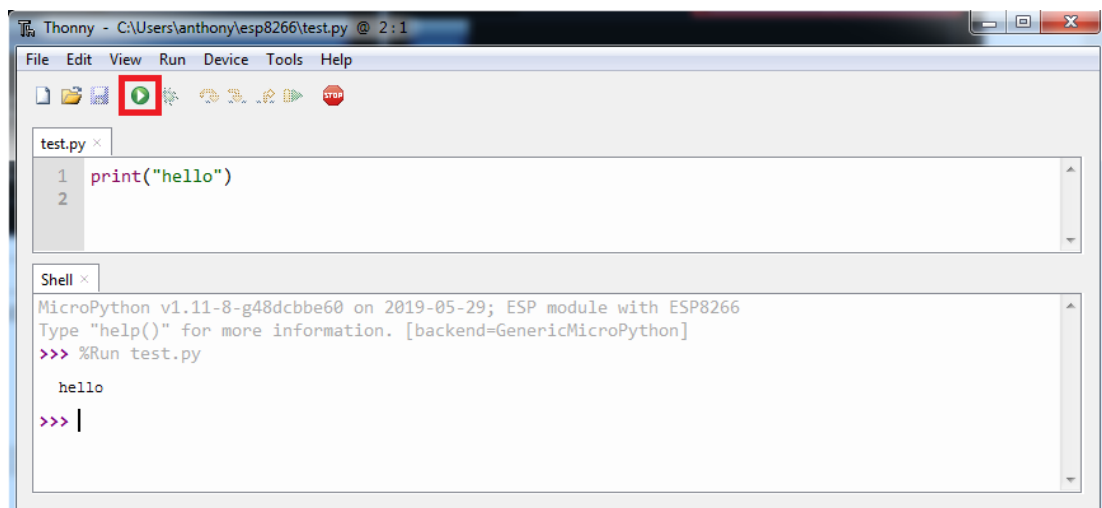
Faire un reset de la carte esp8266 **avant** de lancer Thonny



Configuration en mode esp8266



Finir par tester le bon fonctionnement



Annexe 2 : Implantation du firmware Micropython dans l'ESP32

Programmation du firmware Micropython sous Windows :

Télécharger l'utilitaire de [programmation expressif](#)

The screenshot shows the Espressif Systems website with the URL <https://www.espressif.com/en/products/socs/esp32/resources>. The navigation bar includes links for Products, Solutions, Support, Ecosystem, Company, Join Us, and Contact Us. The breadcrumb trail is Products > SoCs > ESP32 >. The 'Resources' tab is selected, showing sections for APKs, Documentation, and Tools. Under the Tools section, three results are listed:

Title	Platform	Version	Release Date	Download
ESP32&ESP8266&ESP32S2 RF Performance Test Demonstration	ZIP	V2.4	2020.05.28	
Flash Download Tools (ESP8266 & ESP32 & ESP32-S2)	Windows PC	V3.8.5	2020.04.26	
ESP-Tuning Tool for TouchSensor	ZIP	V1.0	2018.09.21	

Décompresser le fichier zip, puis exécuter l'utilitaire flash_download_tools

The image shows three screenshots illustrating the steps to use the flash_download_tool_3.8.5.exe utility:

- Step 1:** A file explorer window showing the contents of the extracted zip file. The file **flash_download_tool_3.8.5.exe** is highlighted with a red box and a red number 1.
- Step 2:** The 'DOWNLOAD TOOL MODE' window is shown. The 'Developer Mode' button is highlighted with a red box and a red number 2.
- Step 3:** The 'ESPRESSIF DOWNLOAD TO...' window is shown. The 'ESP32 DownloadTool' button is highlighted with a red box and a red number 3.

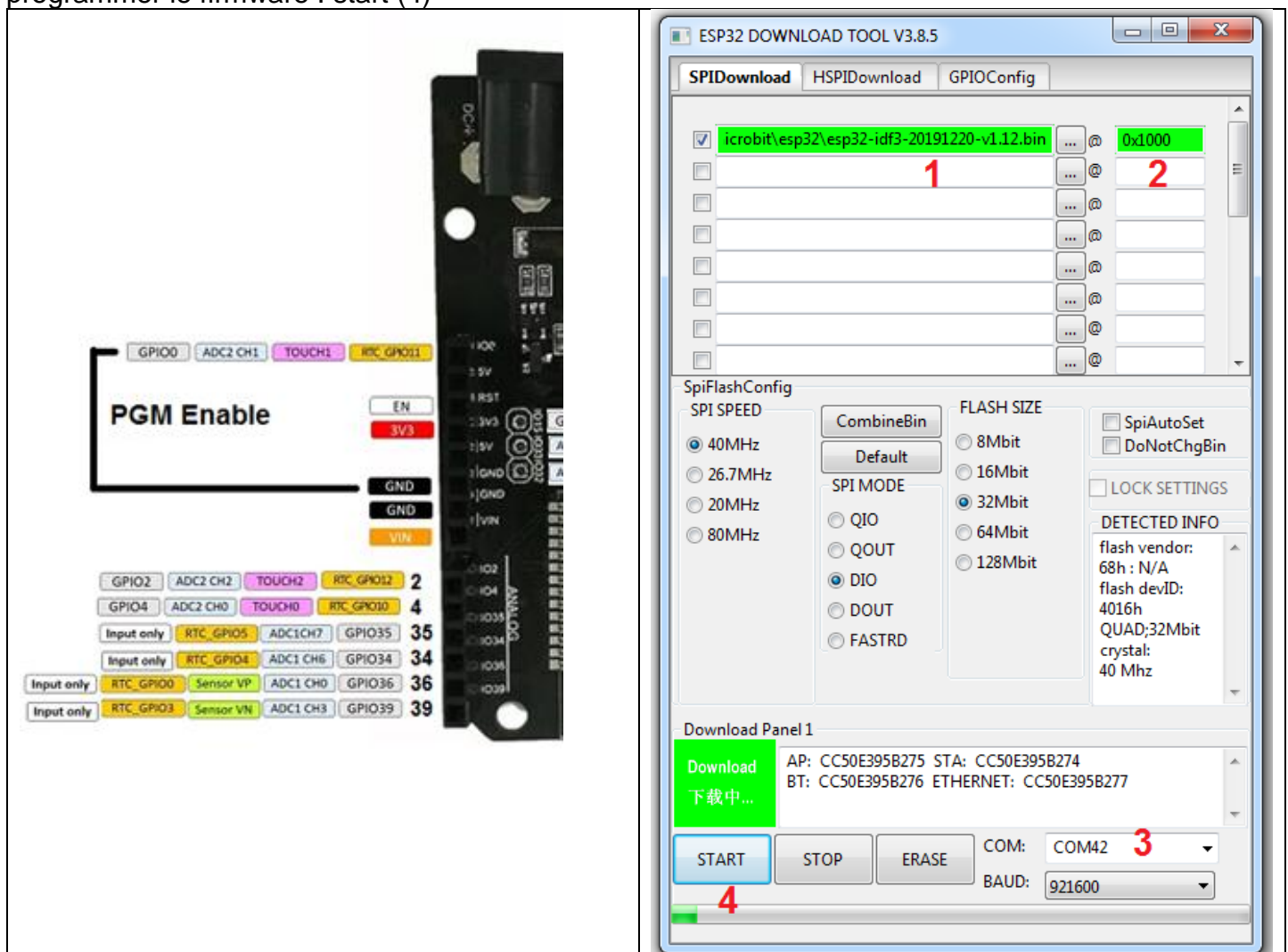
Télécharger le fichier firmware pour l'esp32 (<https://micropython.org/download/esp32/>)

Firmware with ESP-IDF v3.x

Firmware built with ESP-IDF v3.x, with support for BLE, LAN and PPP:

- GENERIC : [esp32-idf3-20200607-unstable-v1.12-510-g1e6d18c91.bin](#)
- GENERIC : [esp32-idf3-20200605-unstable-v1.12-496-ge54626f4c.bin](#)
- GENERIC : [esp32-idf3-20200602-unstable-v1.12-483-g22806ed5d.bin](#)
- GENERIC : [esp32-idf3-20200529-unstable-v1.12-478-g1662a0b06.bin](#)
- GENERIC : [esp32-idf3-20191220-v1.12.bin](#)
- GENERIC : [esp32-idf3-20190529-v1.11.bin](#)
- GENERIC : [esp32-idf3-20190125-v1.10.bin](#)
- GENERIC : [esp32-idf3-20180511-v1.9.4.bin](#)

Sélectionner le fichier binaire (1), préciser l'adresse de programmation (2), sélectionner le bon port de communication (3), relier un fil entre la masse GND et l'entrée GPIO 0 (PGM Enable) puis programmer le firmware : start (4)



Faire un reset de la carte esp32 **avant** de lancer Thonny IDE

Programmation du firmware Micropython sous Linux :

Pour programmer le firmware de l'esp32, il faudra installer esptool sous linux en ligne de commande

<https://github.com/espressif/esptool>

```
sudo apt install python-pip
sudo pip install --upgrade pip
sudo pip install esptool
pip install pyserial
sudo pip install pyserial
```

Repérer le port série de la carte esp32

```
nsi@nsi-LIFEBLOCK-A555:~$ ls /dev/tty*
/dev/tty17 /dev/tty32 /dev/tty48 /dev/tty63 /dev/ttyS2 /dev/ttyS7
/dev/tty18 /dev/tty33 /dev/tty49 /dev/tty7 /dev/ttyS20 /dev/ttyS8
/dev/tty19 /dev/tty34 /dev/tty5 /dev/tty8 /dev/ttyS21 /dev/ttyS9
/dev/tty2 /dev/tty35 /dev/tty50 /dev/tty9 /dev/ttyS22 /dev/ttyUSB0
/dev/tty20 /dev/tty36 /dev/tty51 /dev/ttyprintk /dev/ttyS23
/dev/tty21 /dev/tty37 /dev/tty52 /dev/ttyS0 /dev/ttyS24
/dev/tty22 /dev/tty38
/dev/tty53 /dev/ttyS1 /dev/ttyS25
```

Télécharger le firmware pour l'esp32 (<https://micropython.org/download/esp32/>)

Standard firmware:

- [esp32-20190610-v1.11-37-g62f004ba4.bin](#) (latest)
- [esp32-20190529-v1.11.bin](#)
- [esp32-20190125-v1.10.bin](#)
- [esp32-20180511-v1.9.4.bin](#)
- [esp32--bluetooth.bin](#)



Renommer le fichier par esp32.bin

Exécuter les 2 commandes suivantes :

```
python esptool.py --port /dev/ttyUSB0 erase_flash
python esptool.py --chip esp32 --port /dev/ttyUSB0 write_flash -z 0x1000
esp32.bin
```

Commandes afin de changer l'adresse MAC de l'ESP32 si nécessaire

```
python espfuse.py --port /dev/ttyUSB0 summary
python espfuse.py --port /dev/ttyUSB0 dump
python espfuse.py --port /dev/ttyUSB0 mac
python espfuse.py --port /dev/ttyUSB0 get_custom_mac
python espfuse.py --port /dev/ttyUSB0 burn_custom_mac de:ad:be:ef:fe:00
python espfuse.py --port /dev/ttyUSB0 get_custom_mac
```

Différences des modules préinstallés entre l'ESP8266 et l'ESP32

Esp8266

MicroPython v1.12 on 2019-12-20; ESP module with ESP8266

Type "help()" for more information.

```
>>> help("modules")
```

<code>__main__</code>	<code>http_client_ssl</code>	<code>uasyncio/__init__</code>	<code>upysh</code>
<code>_boot</code>	<code>http_server</code>	<code>uasyncio/core</code>	<code>urandom</code>
<code>_onewire</code>	<code>http_server_ssl</code>	<code>ubinascii</code>	<code>ure</code>
<code>_webrepl</code>	<code>inisetup</code>	<code>ucollections</code>	<code>urequests</code>
<code>apa102</code>	<code>lwip</code>	<code>ucryptolib</code>	<code>urllib/urequest</code>
<code>btree</code>	<code>machine</code>	<code>uctypes</code>	<code>uselect</code>
<code>builtins</code>	<code>math</code>	<code>uerrno</code>	<code>usocket</code>
<code>dht</code>	<code>micropython</code>	<code>uhashlib</code>	<code>ussl</code>
<code>ds18x20</code>	<code>neopixel</code>	<code>uheapq</code>	<code>ustruct</code>
<code>esp</code>	<code>network</code>	<code>uio</code>	<code>utime</code>
<code>example_pub_button</code>		<code>ntptime</code>	<code>ujson</code>
<code>utimeq</code>			
<code>example_sub_led</code>	<code>onewire</code>	<code>umqtt/robust</code>	<code>uwebsocket</code>
<code>flashbdev</code>	<code>port_diag</code>	<code>umqtt/simple</code>	<code>uzlib</code>
<code>framebuf</code>	<code>ssd1306</code>	<code>uos</code>	<code>webrepl</code>
<code>gc</code>	<code>sys</code>	<code>upip</code>	<code>webrepl_setup</code>
<code>http_client</code>	<code>uarray</code>	<code>upip_utarfile</code>	<code>websocket_helper</code>

Plus any modules on the filesystem

```
>>>
```

Esp32

MicroPython v1.12 on 2019-12-20; ESP32 module with ESP32

Type "help()" for more information.

```
>>> help("modules")
```

<code>__main__</code>	<code>framebuf</code>	<code>ucryptolib</code>	<code>urandom</code>
<code>_boot</code>	<code>gc</code>	<code>uctypes</code>	<code>ure</code>
<code>_onewire</code>	<code>inisetup</code>	<code>uerrno</code>	<code>urequests</code>
<code>_thread</code>	<code>machine</code>	<code>uhashlib</code>	<code>uselect</code>
<code>_webrepl</code>	<code>math</code>	<code>uhashlib</code>	<code>usocket</code>
<code>apa106</code>	<code>micropython</code>	<code>uheapq</code>	<code>ussl</code>
<code>btree</code>	<code>neopixel</code>	<code>uio</code>	<code>ustruct</code>
<code>builtins</code>	<code>network</code>	<code>ujson</code>	<code>utime</code>
<code>cmath</code>	<code>ntptime</code>	<code>umqtt/robust</code>	<code>utimeq</code>
<code>dht</code>	<code>onewire</code>	<code>umqtt/simple</code>	<code>uwebsocket</code>
<code>ds18x20</code>	<code>sys</code>	<code>uos</code>	<code>uzlib</code>
<code>esp</code>	<code>uarray</code>	<code>upip</code>	<code>webrepl</code>
<code>esp32</code>	<code>ubinascii</code>	<code>upip_utarfile</code>	<code>webrepl_setup</code>
<code>flashbdev</code>	<code>ucollections</code>	<code>upysh</code>	<code>websocket_helper</code>

Plus any modules on the filesystem

```
>>>
```

La liste dans l'ESP32 est moins abondante que dans l'ESP8266.

Il y a moins de modules préinstallés dans l'ESP32.

Pour ajouter le module gérant l'afficheur OLED ssd1306, procéder comme suit :

Exécuter le script suivant en adaptant le nom du point d'accès WIFI (SSID) avec son mot de passe

```
"""
wifi connexion
"""
import network
# user data
ssid = "Livebox-xxxx" # wifi router name
pw = "xxxxxxxxxxxxxxxxxxxx" # wifi router password

# wifi connection station mode
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(ssid, pw)

# wait for connection
while not wifi.isconnected():
    pass

# wifi connected
print(wifi.ifconfig())
```

Vérifier la connexion sur le point d'accès WIFI. Adresse IP.

Puis dans la console exécuter les 2 lignes suivantes :

```
>>> import upip
>>> upip.install('micropython-ssd1306')
```

```
>>> %Run -c $EDITOR_CONTENT
I (601679) phy: phy_version: 4102, 2fa7a43, Jul 15 2019, 13:06:06, 0, 2
('192.168.1.126', '255.255.255.0', '192.168.1.1', '192.168.1.1')

>>> upip.install('micropython-ssd1306')
Installing to: /lib/
Installing micropython-ssd1306 0.2 from https://files.pythonhosted.org/packages/4d/2b/de6cd81e86782c87bdfd57066ae917aba453d0f89b54f2080c6d4b150e45/micropython-ssd1306-0.2.tar.gz
>>> import ssd1306
>>> |
```

Il en va de même pour l'installation d'autres modules ou bibliothèques ci celles-ci sont disponibles sur le net. <https://pypi.org/project/micropython-ssd1306/>

Si le message suivant apparaît, cela veut dire que l'esp32 n'est pas connecté au point d'accès Internet

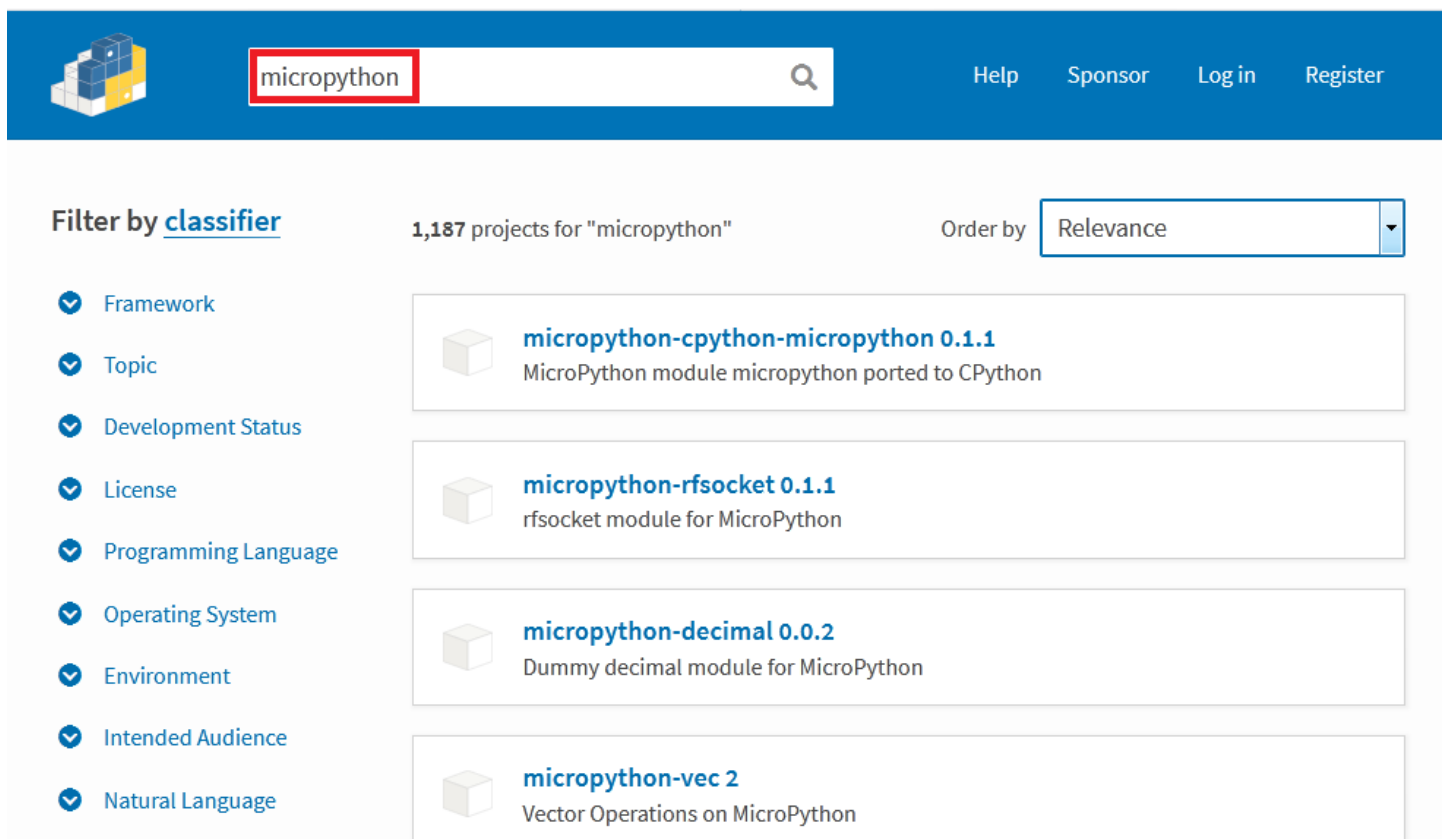
```
>>> upip.install('micropython-ssd1306')
Installing to: /lib/
Error installing 'micropython-ssd1306': list index out of range, packages
may be partially installed
```

Quelques exemples d'installation





```
>>> import upip
>>> upip.install('micropython-ssd1306')
Installing to: /lib/
Warning: micropython.org SSL certificate is not validated
Installing micropython-ssd1306 0.2 from
https://files.pythonhosted.org/packages/4d/2b/de6cd81e86782c87bdfd57066ae9
17aba453d0f89b54f2080c6d4b150e45/micropython-ssd1306-0.2.tar.gz
>>> upip.install('micropython-uasyncio')
Installing to: /lib/
Installing micropython-uasyncio 2.0 from
https://micropython.org/pi/uasyncio/uasyncio-2.0.tar.gz
Installing micropython-uasyncio.core 2.0 from
https://micropython.org/pi/uasyncio.core/uasyncio.core-2.0.tar.gz
>>> upip.install('micropython-sqlite3')
Installing to: /lib/
Installing micropython-sqlite3 0.2.4 from
https://micropython.org/pi/sqlite3/sqlite3-0.2.4.tar.gz
Installing micropython-ffilib 0.1.3 from
https://micropython.org/pi/ffilib/ffilib-0.1.3.tar.gz
```

Pour connaître les bibliothèques compatibles avec micropython, effectuer une recherche sur le site pypi.org.

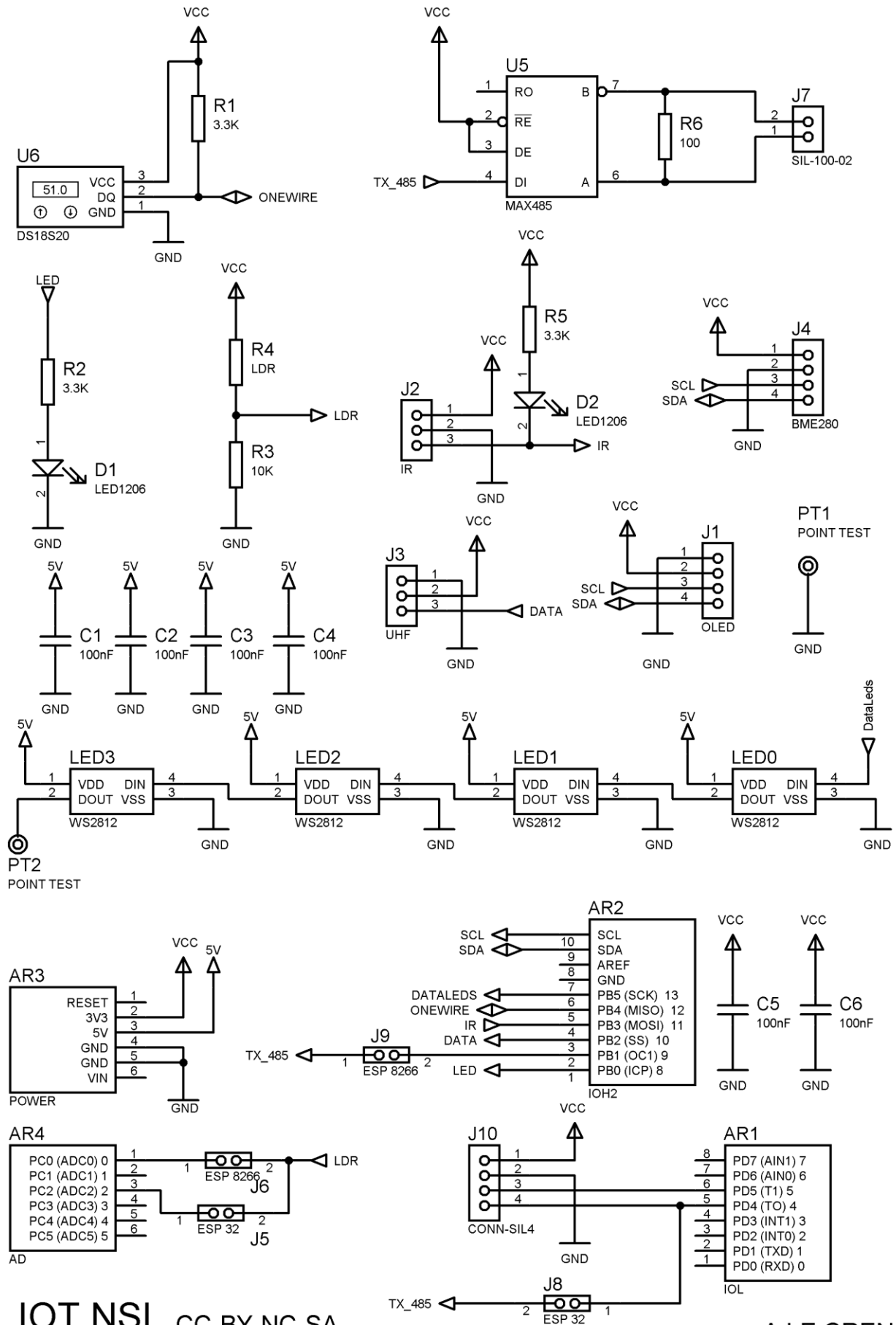
<https://pypi.org/search/?q=micropython>



The screenshot shows the PyPI search results for 'micropython'. The search bar at the top contains 'micropython'. Below the search bar, there are filters on the left and a list of results on the right. The filters include Framework, Topic, Development Status, License, Programming Language, Operating System, Environment, Intended Audience, and Natural Language. The results list shows four projects: micropython-cpython-micropython 0.1.1, micropython-rfsocket 0.1.1, micropython-decimal 0.0.2, and micropython-vec 2. Each result includes a small icon and a brief description.

Filter by classifier	1,187 projects for "micropython"	Order by Relevance
<input checked="" type="checkbox"/> Framework	 micropython-cpython-micropython 0.1.1 MicroPython module micropython ported to CPython	
<input checked="" type="checkbox"/> Topic	 micropython-rfsocket 0.1.1 rfsocket module for MicroPython	
<input checked="" type="checkbox"/> Development Status	 micropython-decimal 0.0.2 Dummy decimal module for MicroPython	
<input checked="" type="checkbox"/> License	 micropython-vec 2 Vector Operations on MicroPython	
<input checked="" type="checkbox"/> Programming Language		
<input checked="" type="checkbox"/> Operating System		
<input checked="" type="checkbox"/> Environment		
<input checked="" type="checkbox"/> Intended Audience		
<input checked="" type="checkbox"/> Natural Language		

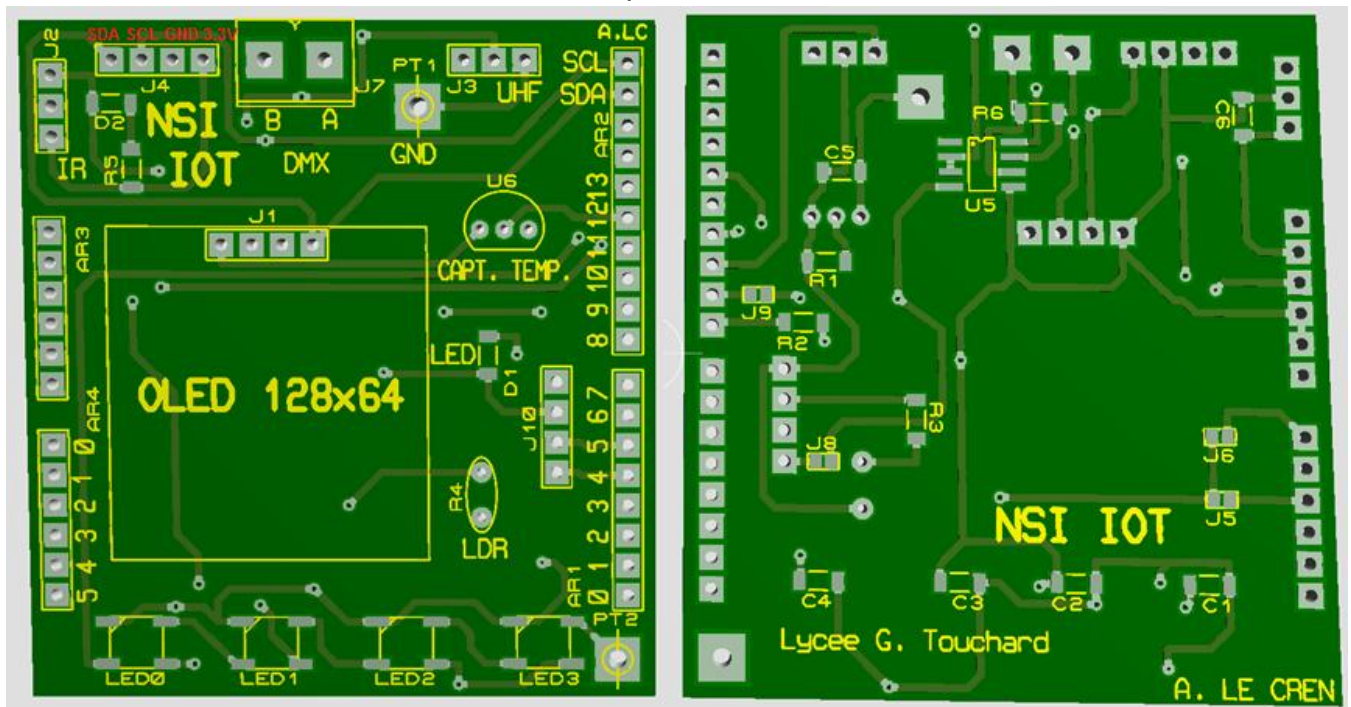
Annexe 3 : Schéma structurel



IOT NSI CC-BY-NC-SA

A.LE CREN

Circuit imprimé shield IOT



Nomenclature

Résistances

- 3 R1,R2,R5 1K cms1206
- 1 R3 10K cms1206
- 1 R4 LDR ldr classique trou traversant petit modèle
- 1 R6 100 cms1206

Condensateurs

- 6 C1-C6 100nF cms1206

Circuits intégrés

<https://www.ebay.fr/itm/100Pcs-MAX485-MAX485CSA-Txrx-RS485-RS422-Lowpwr-SOP-8-Date-CODE-12-wv/262963276497>

- 1 U5 MAX485 sop-8

<https://www.ebay.fr/itm/10PCS-IC-DS18S20-DS1820-Digital-Thermometer-IC-GOOD-QUALITY-Li2/132256267252>

- 1 U6 DS18S20 to-92

Diodes : 2 D1,D2 LED cms1206

Divers :

- 1 AR1 IOL barette male sécable 8pts
- 1 AR2 IOH2 barette male sécable 10pts
- 1 AR3 POWER barette male sécable 6pts
- 1 AR4 AD barette male sécable 6pts
- 1 J1 OLED oled ssd1306 128x64 I2C
- 1 J2 IR VS1838B recepteur IR
- 1 J3 barette femelle sécable 3pts
- 1 J4 barette femelle sécable 4pts
- 1 J7 SIL-100-02 bornier gris 2pts
- 1 J10 barette femelle sécable 4pts
- 4 LED0-LED3 WS2812b led RGB boîtier blanc cms a souder

<https://www.ebay.fr/itm/0-96-I2C-IIC-SPI-Serial-128X64-OLED-LCD-LED-Display-Module-for-Arduino-SSD1306/223119333626>

<https://www.ebay.fr/itm/WS2812B-5050-SMD-Addressable-Digital-RGB-LED-4-pin-Chip-5V-Black-or-White/183460578312>

<https://www.ebay.fr/itm/20PCS-VS1838-TL1838-VS1838B-Universal-Infrared-Receiving-Head-For-Remote-control/201249361916>

Ajouter une carte **esp8266** ou **esp32**

<https://www.ebay.fr/itm/OTA-WeMos-D1-CH340-WiFi-Development-Board-ESP8266-ESP-12E-For-Arduino-IDE-UNO-R3/163429353623>

<https://www.ebay.fr/itm/ESP32-UNO-R3-D1-R32-WIFI-Bluetooth-USB-B-CH340-Development-Board-For-Arduino/264083453537>

Sans oublier le cordon micro-usb

Annexe 4 : Installer un point d'accès Wifi RaspAP

<https://raspbrian-france.fr/creer-un-hotspot-wi-fi-en-moins-de-10-minutes-avec-la-raspberry-pi/>

```
sudo cp /etc/wpa_supplicant/wpa_supplicant.conf /etc/wpa_supplicant/wpa_supplicant.conf.sav
sudo cp /dev/null /etc/wpa_supplicant/wpa_supplicant.conf
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

ajouter dans le fichier

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
```

Ctrl+o, ctrl+x : sauver,quitter

```
wget -q https://git.io/voEUQ -O /tmp/raspap && bash /tmp/raspap
sudo reboot
```

IP address: 10.3.141.1

Username: **admin**

Password: **secret**

DHCP range: 10.3.141.50 to 10.3.141.255

SSID: raspi-webgui

Password: a definir dans le portail ci dessous

RaspAP Wifi Portail v1.5

- Tableau de bord
- Configurer client WiFi
- Configurer hotspot
- Configurer réseau
- Configurer server DHCP
- Configurer Auth
- Change Thème
- Data usage
- Système
- About RaspAP




Tableau de bord

Interface is up. X

Informations d'interface

Nom de l'interface	wlan0
IPv4 Address	10.3.141.1
Masque de sous-réseau	255.255.255.0
IPv6 Address	
Array	
Adresse Mac	b8:27:eb:07:2a:81

Informations sans fil

Connecté à	Not connected
AP Mac Adresse	
Bitrate	
Niveau du signal	
Puissance de transmission	31.00 dBm
La fréquence	
MHz	
Qualité de lien	<div></div>

Statistiques d'interface

Paquets reçus	1597
Octets reçus	222792 (217,57 KB)
Paquets transférés	2140
Octets transférés	1903992 (1,82 MB)

Appareils connectés

Nom d'hôte	Adresse IP	Adresse Mac
Honor_6X	10.3.141.237	44:c3:46:b8:d2

Comment configurer le proxy (utile dans un lycée avec Raspbian)

<https://www.raspberrypi.org/documentation/configuration/use-a-proxy.md>

Configuration de raspbian via le terminal

```
sudo nano /etc/environment
```

```
export http_proxy="http://username:password@proxyipaddress:proxyport"
export https_proxy="http://username:password@proxyipaddress:proxyport"
export no_proxy="localhost, 127.0.0.1"
```

Pour que vos commandes via sudo gardent ces paramètres,

```
sudo visudo
```

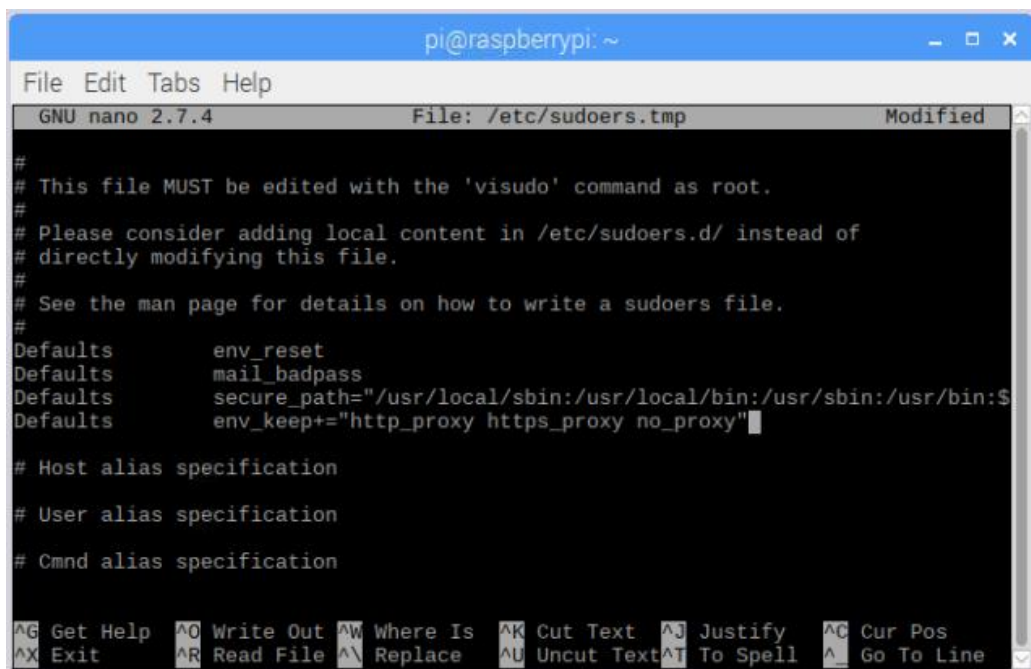
```
Defaults    env_keep+="http_proxy https_proxy no_proxy"
```

Pour que le système de package APT utilise le proxy, créer un fichier 10proxy dans /etc/apt/apt.conf.d/ :

```
cd /etc/apt/apt.conf.d
sudo nano 10proxy
```

```
Acquire::http::proxy "http://username:password@proxyipaddress:proxyport";
```

```
Acquire::https::proxy "http://username:password@proxyipaddress:proxyport";
```



Annexe 5 : Exemples de programmes Micropython

Timers en micropython

La précision du signal généré sur la broche 13 n'a rien de comparable avec le même programme en langage C.

```
"""
timer irq
the ESP32 has 4 hardware timers. For this example, we will use timer 0.
"""

import machine

led = machine.Pin(16, machine.Pin.OUT)
timer = machine.Timer(0)

def handleInterrupt(timer):
    global ledState
    ledState ^= 1
    led.value(ledState)

ledState = 0

timer.init(freq=1000, mode=machine.Timer.PERIODIC, callback=handleInterrupt)
# timer.init(period=1000, mode=machine.Timer.PERIODIC, callback=handleInterrupt)

while True:
    pass
```

Scanner I2C

```
"""
scan i2c esp32
Scan i2c bus...
i2c devices found: 1
Decimal address: 60 | Hexa address: 0x3c
"""

import machine
i2c = machine.I2C(scl=machine.Pin(4), sda=machine.Pin(5))

print('Scan i2c bus...')
devices = i2c.scan()

if len(devices) == 0:
    print("No i2c device !")
else:
    print('i2c devices found:', len(devices))

for device in devices:
    print("Decimal address: ", device, " | Hexa address: ", hex(device))
```

Afficheur OLED SSD1306

```
"""
oled test
"""

import machine, ssd1306

i2c = machine.I2C(scl=machine.Pin(4), sda=machine.Pin(5))
oled = ssd1306.SSD1306_I2C(128, 64, i2c, 0x3c)
oled.fill(0)
oled.text("Hello f4goh", 0, 0)
oled.show()
```

Emission et réception sur l'UART 2 avec asyncio

```

"""
esp32 pinout
gpio16 rx
gpio17 tx
"""
import uasyncio as asyncio
from machine import UART
uart = UART(2, 9600)

async def sender():
    swriter = asyncio.StreamWriter(uart, {})
    while True:
        await swriter.awrite('Hello uart. \n')
        print('Wrote')
        await asyncio.sleep(2)

async def receiver():
    sreader = asyncio.StreamReader(uart)
    while True:
        res = await sreader.readline()
        print('Recieved', res)

loop = asyncio.get_event_loop()
loop.create_task(sender())
loop.create_task(receiver())
loop.run_forever()

```

Réception GPS sur l'UART 2 avec asyncio (connecteur J10)

A tester : https://github.com/alexmrgt/micropython-gps/blob/master/adafruit_gps.py

```

"""
esp32 pinout
gpio16 rx relié à la sortie GPS
gpio17 tx
"""
import uasyncio as asyncio
from machine import UART
uart = UART(2, 9600)

async def receiver():
    sreader = asyncio.StreamReader(uart)
    while True:
        res = await sreader.readline()
        nmea=res.split(b',')
        if nmea[0]==b'$GPGGA':
            #print('Recieved', nmea)
            print(int(float(nmea[1].decode())) ,end=',')
            print('latitude :',float(nmea[2].decode()) ,end=',')
            print(' longitude :',float(nmea[4].decode()))

loop = asyncio.get_event_loop()
loop.create_task(receiver())
loop.run_forever()

```

133405,latitude : 4753.415, longitude : 16.6092

133406,latitude : 4753.415, longitude : 16.6092

133407,latitude : 4753.415, longitude : 16.6092

Détecter un code infra-rouge RC5

https://www.st.com/content/ccc/resource/technical/document/application_note/c7/d1/63/f7/80/06/41/a4/CD00267896.pdf/file/CD00267896.pdf/jcr:content/translations/en.CD00267896.pdf

```
# ir 13

from machine import Pin
from time import sleep_us, ticks_us, ticks_diff
from neopixel import NeoPixel

ir = Pin(13, Pin.IN)
led = Pin(0, Pin.OUT)
np = NeoPixel(Pin(14), 4)

time_start = ticks_us()
code = 2
actualBit = 0
lastBit = 1
bit = 1
trigger = 0

def callback(p):
    global time_start, code, actualBit, lastBit, bit, trigger
    edge = ir.value()
    time_stop = ticks_us()
    delta = ticks_diff(time_stop, time_start)
    time_start = time_stop
    if delta > 2000:
        if bit == 13:
            print('-> ', code & 0x3f)
            trigger = code & 0x3f
            code = 2
            lastBit = 1
            actualBit = 0
            bit = 0
            led.value(led.value() ^ 1)
        else:
            long = 0
            valid = 0
            if delta > 1100:
                long = 1

    if edge == 1:
        if lastBit == 1:
            if long == 1:
                actualBit = 0
                lastBit = 1
                valid = 1
            if lastBit == 0:
                if long == 0:
                    actualBit = 0
                    lastBit = 0
                    valid = 1
        if edge == 0:
            if lastBit == 1:
                if long == 0:
                    actualBit = 1
                    lastBit = 1
                    valid = 1
            if lastBit == 0:
                if long == 1:
                    actualBit = 1
                    lastBit = 0
                    valid = 1
    if valid == 1:
        bit += 1
        code |= actualBit
        if bit < 13:
            code <= 1
        lastBit = actualBit

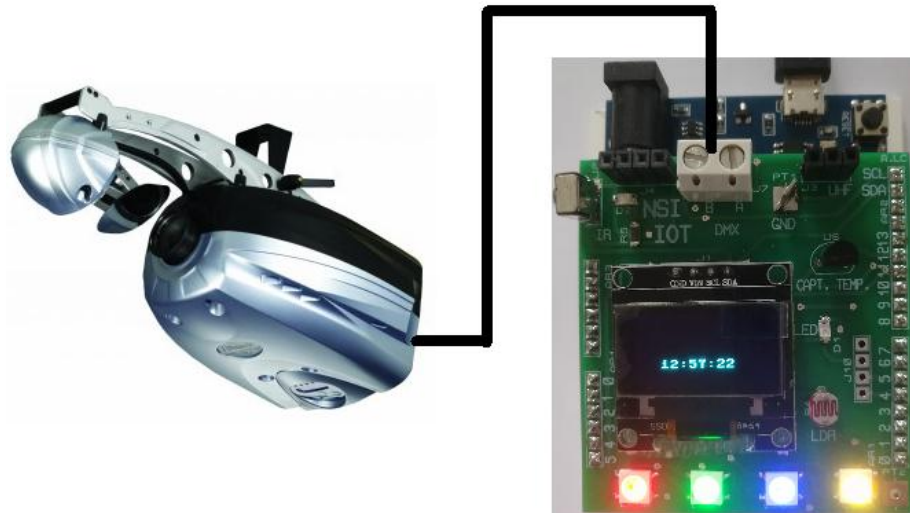
ir.irq(trigger=Pin.IRQ_RISING | Pin.IRQ_FALLING, handler=callback)

couleur = (0, 0, 0)

while (True):
    if trigger == 1:
        couleur = (255, 0, 0)
    elif trigger == 2:
        couleur = (0, 255, 0)
    elif trigger == 3:
        couleur = (0, 0, 255)

    for n in range(0, 4):
        np[n] = couleur
    np.write()
```

Contrôler un périphérique DMX



```

"""
routines DMX configuré ici pour l'esp32
esp 8266 uart 1
esp 32    uart 2
          esp8266    esp32
broche    2(9)      17(4)
"""

from machine import UART
import machine, time
from array import array

dmx_message = array('B', [0] * 16) # 16 channels

def set_channels(message):
    for ch in message:
        dmx_message[ch] = message[ch]

def write_frame():
    #dmx_uart = machine.Pin(2,machine.Pin.OUT)
    dmx_uart = machine.Pin(17, machine.Pin.OUT)
    dmx_uart.value(0)
    time.sleep_us(74)
    dmx_uart.value(1)
    # Now turn into a UART port and send DMX data
    # dmx_uart = UART(1)
    dmx_uart = UART(2)
    dmx_uart.init(250000, bits=8, parity=None, stop=2)
    #send bytes
    dmx_uart.write(dmx_message)
    #Delete as its going to change anyway
    del (dmx_uart)

set_channels({1:80})
set_channels({2:255})
set_channels({3:174})
set_channels({4:255})

print(dmx_message)

write_frame()

```

Contrôler un périphérique DMX avec une librairie

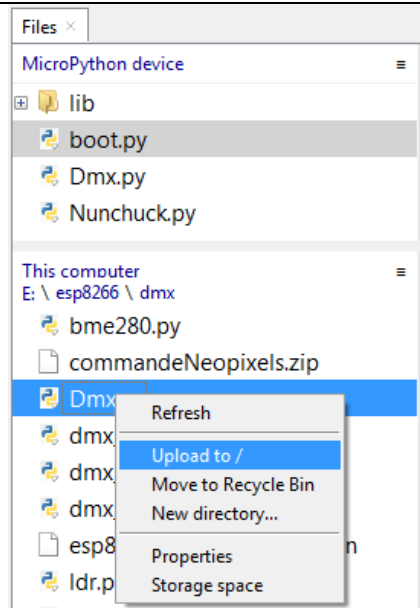
Charger le fichier Dmx.py dans l'ESP : Menu View, Files, clic droit sur dmx.py puis upload to /

```
from machine import UART, Pin
from time import sleep_us
from array import array

class Dmx():
    def __init__(self, pin, uart, nbChannels):
        self.pin = pin
        self.uart = uart
        self.buffer=array('B', [0] * nbChannels)

    def setChannel(self, channel, value):
        self.buffer[channel]=value

    def write(self):
        dmx_uart = Pin(self.pin, Pin.OUT)
        dmx_uart.value(0)
        sleep_us(74)
        dmx_uart.value(1)
        dmx_uart = UART(self.uart)
        dmx_uart.init(250000, bits=8, parity=None, stop=2)
        dmx_uart.write(self.buffer)
        del(dmx_uart)
```



Choisir l'esp32 ou esp8266, puis exécuter le script. Le programme effectue un fondu de couleur sur un spot RGB

```
"""
esp 8266 uart 1
esp 32  uart 2
          esp8266   esp32
broche   2(9)      17(4)
"""
from Dmx import Dmx
from time import sleep_ms

#dmx=Dmx(17,2,16) #ESP32 pin17,UART 2, 16 canaux
dmx=Dmx(2,1,16)  #ESP8266 pin2,UART 1, 16 canaux

dmx.setChannel(1,0)
dmx.setChannel(2,0)
dmx.setChannel(3,0)
dmx.setChannel(4,255)
dmx.write()

for n in range(0,255):
    dmx.setChannel(1,n)
    dmx.setChannel(2,255-n)
    dmx.setChannel(3,255-n)
    sleep_ms(10)
    dmx.write()
```



Lire des données d'une manette nunchuk de Wii

Charger le fichier Nunchuk.py dans l'ESP : Menu View, Files, clic droit sur dmx.py puis upload to /

```
class Nunchuck():
    def __init__(self, i2c, addr=0x52):
        self.i2c = i2c
        self.addr = addr
        self.buffer = bytearray(6)
        self.i2c.start()
        self.i2c.writeto(self.addr, b'\xF0\x55')
        self.i2c.stop()
        self.i2c.start()
        self.i2c.writeto(self.addr, b'\xFB\x00')
        self.i2c.stop()

    def rescale(self, valeur, in_min, in_max, out_min, out_max):
        return int((valeur - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)

    def read(self, scale=0):
        self.i2c.start()
        self.i2c.writeto(self.addr, b'\x00')
        self.i2c.stop()
        self.i2c.start()
        self.buffer=self.i2c.readfrom(self.addr, 6)
        self.i2c.stop()
        joyX=self.buffer[0]
        joyY=self.buffer[1]
        accX = (self.buffer[2] * 4) + ((self.buffer[5] // 4) & 3)
        accY = (self.buffer[3] * 4) + ((self.buffer[5] // 16) & 3)
        accZ = (self.buffer[4] * 4) + ((self.buffer[5] // 64) & 3)
        btC = (self.buffer[5] // 2 ) & 1
        btZ = self.buffer[5] & 1
        if scale==1:
            accX=self.rescale (accX, 256, 768,0, 255)
            accY=self.rescale (accY, 256, 768,0, 255)
            accZ=self.rescale (accZ, 256, 768,0, 255)
            if accX>255:
                accX=255
            if accX<0:
                accX=0
            if accY>255:
                accY=255
            if accY<0:
                accY=0
            if accZ>255:
                accZ=255
            if accZ<0:
                accZ=0
        return (accX,accY,accZ,joyX,joyY,btC,btZ)
```

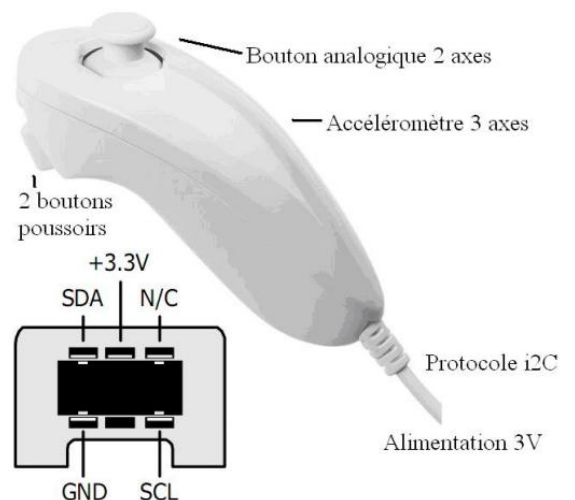
Pour relier la manette à la carte ESP, couper le connecteur en identifiant le nom des broches avec les 4 fils de couleur.

```
"""
Exemple de programme
Nunchuck I2C
"""
from machine import Pin,I2C
from time import sleep_ms
from Nunchuck import Nunchuck

#i2c = I2C(scl=Pin(22), sda=Pin(21))
i2c = I2C(scl=Pin(5), sda=Pin(4))
nun=Nunchuck(i2c)

while(1):
    print(nun.read(0))
    sleep_ms(500)
```

Caractéristiques de la manette nunchuk



Annexe 6 : Programmation asynchrone avec librairie uasyncio

<https://github.com/peterhinch/micropython-async/blob/master/TUTORIAL.md>

Attention la dernière version d'asyncio n'est pas dans le firmware de micropython. Surveillez les mises à jour.

```
import machine
import uasyncio as asyncio

led1 = machine.Pin(2, machine.Pin.OUT)
led2 = machine.Pin(15, machine.Pin.OUT)

async def blink(led, delay): # coroutine
    while True:
        print(led, "on")
        led.on()
        await asyncio.sleep_ms(delay)
        print(led, "off")
        led.off()
        await asyncio.sleep_ms(delay)

# boucle d'événements
loop = asyncio.get_event_loop()
loop.create_task(blink(led1, 500)) # Schedule ASAP
loop.create_task(blink(led2, 1000)) # Schedule ASAP
loop.run_forever()
```

#ctrl+c pour stopper

Pin(2) on
Pin(15) on
Pin(2) off
Pin(15) off
Pin(2) on
Pin(2) off
Pin(15) on
Pin(2) on
Pin(2) off
Pin(15) off

Traceback (most recent call last):

File "C:\Users\anthony\esp8266\test.py", line 21, in <module>

File "uasyncio/core.py", line 173, in run_forever

File "uasyncio/__init__.py", line 69, in wait

KeyboardInterrupt:

>>>

>>> import uasyncio

>>> dir(uasyncio)

['_class_', '__name__', '__path__', 'DEBUG', 'log', 'select', 'sleep', 'sleep_ms', 'time', 'ucollections', 'uerrno', 'utimeq', 'type_gen', 'set_debug', 'CancelledError', 'TimeoutError', 'EventLoop', 'SysCall', 'SysCall1', 'StopLoop', 'IORead', 'IOWrite', 'IOReadDone', 'IOWriteDone', 'get_event_loop', 'SleepMs', 'cancel', 'TimeoutObj', 'wait_for_ms', 'wait_for', 'coroutine', 'ensure_future', 'Task', '_socket', 'PollEventLoop', 'StreamReader', 'StreamWriter', 'open_connection', 'start_server', 'uasyncio', 'core']

Remarque :

Par défaut il n'y a pas le module uasyncio dans l'esp32, il faut l'installer manuellement après être connecté sur votre point d'accès wifi.

```
>>> import upip
```

```
>>> upip.install('micropython-uasyncio')
```

Installing to: /lib/

Warning: micropython.org SSL certificate is not validated

Installing micropython-uasyncio 2.0 from <https://micropython.org/pi/uasyncio/uasyncio-2.0.tar.gz>

Installing micropython-uasyncio.core 2.0 from <https://micropython.org/pi/uasyncio/core/uasyncio.core-2.0.tar.gz>


upip.install('micropython-ssd1306') # pour l'afficheur oled ssd1306

Annexe 7 : Utilisation du capteur BME280

Câbler le capteur sur la carte IOT et vérifier son adresse 119

<pre> """ scan i2c esp32 Scan i2c bus... """ from machine import Pin, I2C #i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000) #esp8266 i2c = I2C(scl=Pin(22), sda=Pin(21), freq=100000) #esp32 print(i2c.scan()) # scan for slave devices </pre>	<pre> >>> >>> %Run -c \$EDITOR_CONTENT [60, 119] >>> 60 : adresse ssd1306 oled 119 : adresse du bme280 </pre>
--	---

Vérifier la connexion WIFI sur le réseau internet

<pre> """ wifi connexion """ import network # user data ssid = "raspi-webgui" # wifi router name pw = "touchardNSI" # wifi router password # wifi connection station mode wifi = network.WLAN(network.STA_IF) wifi.active(True) wifi.connect(ssid, pw) # wait for connection while not wifi.isconnected(): pass # wifi connected print(wifi.ifconfig()) </pre>	<p>Résultat dans la console :</p> <pre> >>> %Run -c \$EDITOR_CONTENT I (352825) phy: phy_version: 4102, 2fa7a43, Jul 15 2019, 13:06:06, 0, 2 ('192.168.1.126', '255.255.255.0', '192.168.1.1', '192.168.1.1') </pre> 
--	---

Installer la bibliothèque 'micropython-bme280'

```

>> import upip
>>> upip.install('micropython-bme280')
Installing to: /lib/
Warning: micropython.org SSL certificate is not validated
Installing micropython-bme280 2.1.3 from
https://files.pythonhosted.org/packages/e4/dc/549e2aaafb3ba9a38f304b809c4bdaa12b1d76979698c038db757740d31b/micropython-bme280-2.1.3.tar.gz

```

Vérifier le fonctionnement du capteur

<pre> """ bme280 test """ from machine import Pin, I2C import bme280 #i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000) #esp8266 i2c = I2C(scl=Pin(22), sda=Pin(21), freq=100000) #esp32 bme = bme280.BME280(mode=bme280.BME280_OSAMPLE_2, address=0x77, i2c=i2c) print(bme.values) </pre>	<pre> >>> %Run -c \$EDITOR_CONTENT ('22.25C', '1004.76hPa', '70.90%') Retourne un tuple avec les valeurs </pre>
--	---

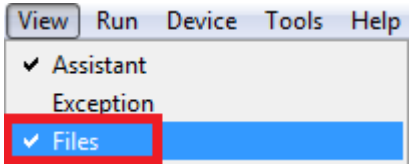
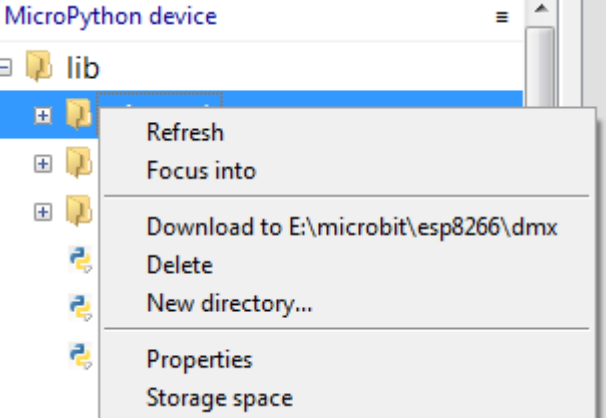
Annexe 8 : Utilisation de la bibliothèque Picoweb (ESP32 seulement)

Installation de la bibliothèque (Vérifier la connexion WIFI sur le réseau internet)

```
>>> import upip
>>> upip.install('picoweb')
Installing to: /lib/
Installing picoweb 1.8.1 from
https://files.pythonhosted.org/packages/ba/e4/68983f6150f7fbd98b6b937bf7f76aeb4eb9ba4abb7c93dc05b2ccdbd25c/picoweb-1.8.1.tar.gz
Installing pycopy-uasyncio 3.6 from
https://files.pythonhosted.org/packages/f9/83/e92363d7d19e6733c4aaa0dcd388f684ae46b024d5bffa92a192efced702/pycopy-uasyncio-3.6.tar.gz
Installing pycopy-pkg_resources 0.2.1 from
https://files.pythonhosted.org/packages/05/4a/5481a3225d43195361695645d78f4439527278088c0822fadaaf2e93378c/pycopy-pkg_resources-0.2.1.tar.gz
Installing pycopy-uasyncio.core 2.3.2 from
https://files.pythonhosted.org/packages/ca/b2/c5bba0bde7022b6d927a6144c026d7bf310d3f8a20e031571fbf1a08a433/pycopy-uasyncio.core-2.3.2.tar.gz
>>>

>>> upip.install('utemplate')
Installing to: /lib/
Installing utemplate 1.3.1 from
https://files.pythonhosted.org/packages/75/11/59ac69a862232afc9fffc1cadcc95395cfb7b28c17610edc061039d4f03f8/utemplate-1.3.1.tar.gz

>>> upip.install('pycopy-logging')
Installing to: /lib/
Installing pycopy-logging 0.3 from
https://files.pythonhosted.org/packages/56/85/47a6790260c85f0dad460124d1f9a6dbdaa0b0ac33b0ac89194f6f106276/pycopy-logging-0.3.tar.gz
>>>
```

<p>Menu View, Files</p>  <p>Supprimer la bibliothèque pycopy-uasyncio (clic droit delete) et la remplacer par une autre version, voir ci-dessous.</p>	<p>MicroPython device</p> 
---	---

```
>>> import upip
>>> upip.install("uasyncio")
Installing to: /lib/
Installing uasyncio 2.0 from https://micropython.org/pi/uasyncio/uasyncio-2.0.tar.gz
Installing micropython-uasyncio.core 2.0 from https://micropython.org/pi/uasyncio.core/uasyncio.core-2.0.tar.gz
```

Programme de test de la page web

```

import picoweb
import network

# user data
ssid = "raspi-webgui" # wifi router name
pw = "touchardNSI" # wifi router password
# wifi connection station mode
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(ssid, pw)
while not wifi.isconnected():
    pass
print(wifi.ifconfig())

app = picoweb.WebApp(__name__)

def web_page():
    html = """<html>
        <head>
            <title>ESP Web Server</title>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <style>
                html {
                    font-family: Helvetica;
                    display:inline-block;
                    margin: 0px auto;
                    text-align: center;
                }
                h1{
                    color: #0F3376;
                    padding: 2vh;
                }
            </style>
            </head>
            <body>
                <h1>ESP Web Server</h1>
            </body></html>"""
    return html

@app.route("/")
def index(req, resp):
    yield from picoweb.start_response(resp)
    yield from resp.awrite(web_page())

app.run(debug=True, port=80, host = wifi.ifconfig()[0])
>>> %Run -c $EDITOR_CONTENT
I (8097) phy: phy_version: 4102, 2fa7a43, Jul 15 2019, 13:06:06, 0, 0
('192.168.1.126', '255.255.255.0', '192.168.1.1', '192.168.1.1')
* Running on http://192.168.1.126:80/

```


Annexe 9 : Commande REPL avec pyQT






REPL signifie « Read Evaluate Print Loop ». C'est le nom donné à l'invite interactive MicroPython à laquelle on accède avec l'ESP8266 ou l'ESP32. L'utilisation de REPL est de loin le moyen le plus simple de tester votre code et d'exécuter des commandes.

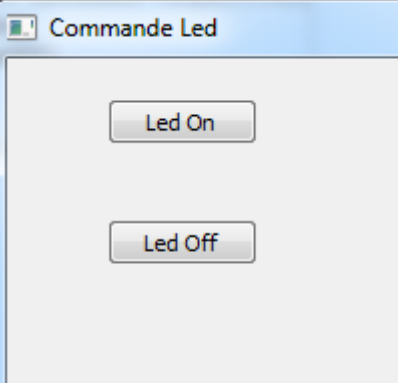
Il y a deux façons d'accéder au REPL : Soit via une connexion filaire via le port série UART, soit via WiFi.

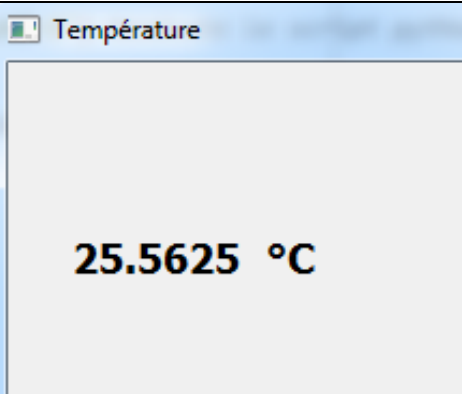
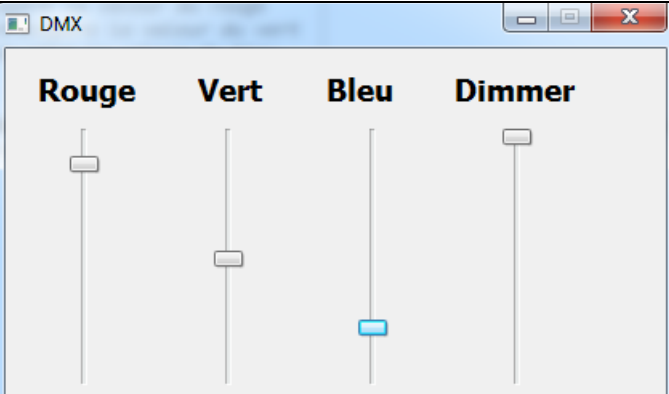
Le REPL est toujours disponible sur le périphérique série UART0, (GPIO1 pour TX et GPIO3 pour RX). Le débit est de 115200 bauds. C'est le moyen le plus simple d'utiliser micropython sur la carte Shield IOT.

Commencer par lire le tutoriel de pyQT (**pyQT.pdf**) en utilisant Edupython

Exemple N°1 : Commande d'une Led (5 fichiers) : `commandeLed.py`

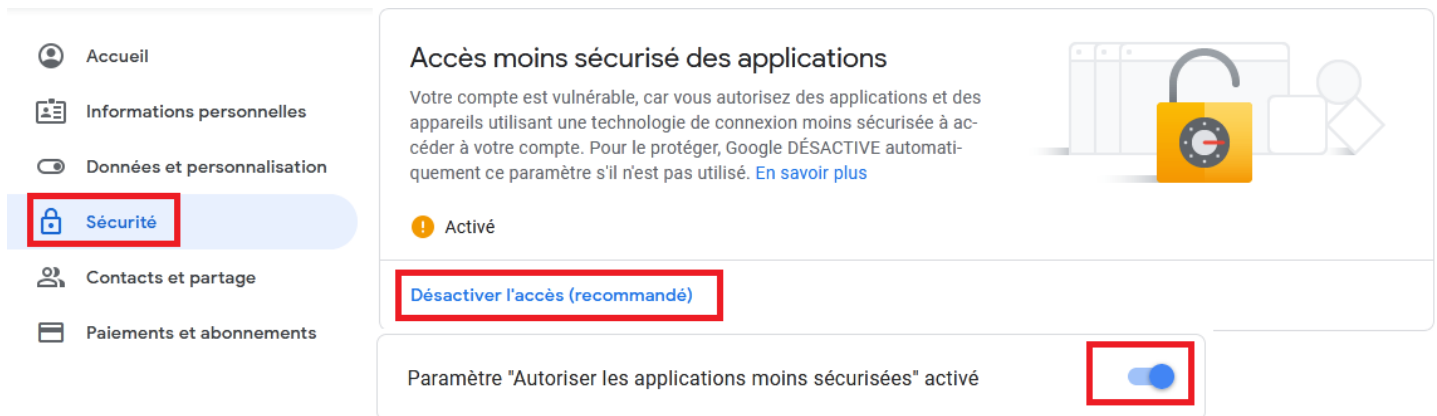
 <code>commandeLed.py</code>	Programme principal exécuté dans Edupython
 <code>commandeLed.ui</code>	Fichier XML (description de la fenêtre graphique)
 <code>led_off.py</code>	Script ESP pour éteindre la led
 <code>led_on.py</code>	Script ESP pour allumer la led
 <code>pyboard.py</code>	Bibliothèque pyboard (REPL)

	Quand on appuie sur les boutons les scripts correspondants sont envoyés et exécutés dans l'ESP.	
	<pre>from machine import Pin from time import sleep_ms #led = Pin(0,Pin.OUT) led = Pin(12,Pin.OUT) led.on() print("led on")</pre>	<pre>from machine import Pin from time import sleep_ms #led = Pin(0,Pin.OUT) led = Pin(12,Pin.OUT) led.off() print("led off")</pre>

Exemple N°1 : Lecture périodique de température (<code>afficheTemperature.py</code>)	Exemple N°3 : Commande RVB DMX avec la bibliothèque <code>Dmx.py</code> (<code>commandeDmx.py</code>)
	

Annexe 10 : Envoyer des emails

Configurer le compte Gmail afin d'avoir un accès moins sécurisé pour l'envoi des messages.



[Télécharger et installer](#) le fichier `umail.py` dans l'ESP

Vérifier la connexion WIFI sur le réseau internet (page 22)

Envoyer un message en connexion TLS/STARTTLS sur le port 587

```
import umail
smtp = umail.SMTP('smtp.gmail.com', 587,
username='nsi.touchard@gmail.com', password='xxxxxxx')
smtp.to('f4goh@orange.fr')
smtp.send("This is an example.")
smtp.quit()
```

Envoyer un message en connexion SSL sur le port 587

```
import umail
smtp = umail.SMTP('smtp.gmail.com', 465, ssl=True) # Gmail's SSL port
smtp.login('nsi.touchard@gmail.com', 'xxxxxxx')
smtp.to('f4goh@orange.fr')
smtp.write("From: Bob <nsi.touchard@gmail.com>\n")
smtp.write("To: Alice <f4goh@orange.fr>\n")
smtp.write("Subject: Poem\n\n")
smtp.write("Roses are red.\n")
smtp.write("Violets are blue.\n")
smtp.write("...\n")
smtp.send()
smtp.quit()
```