



# Les objets connectés

## (Partie 1 : découverte de la carte électronique)

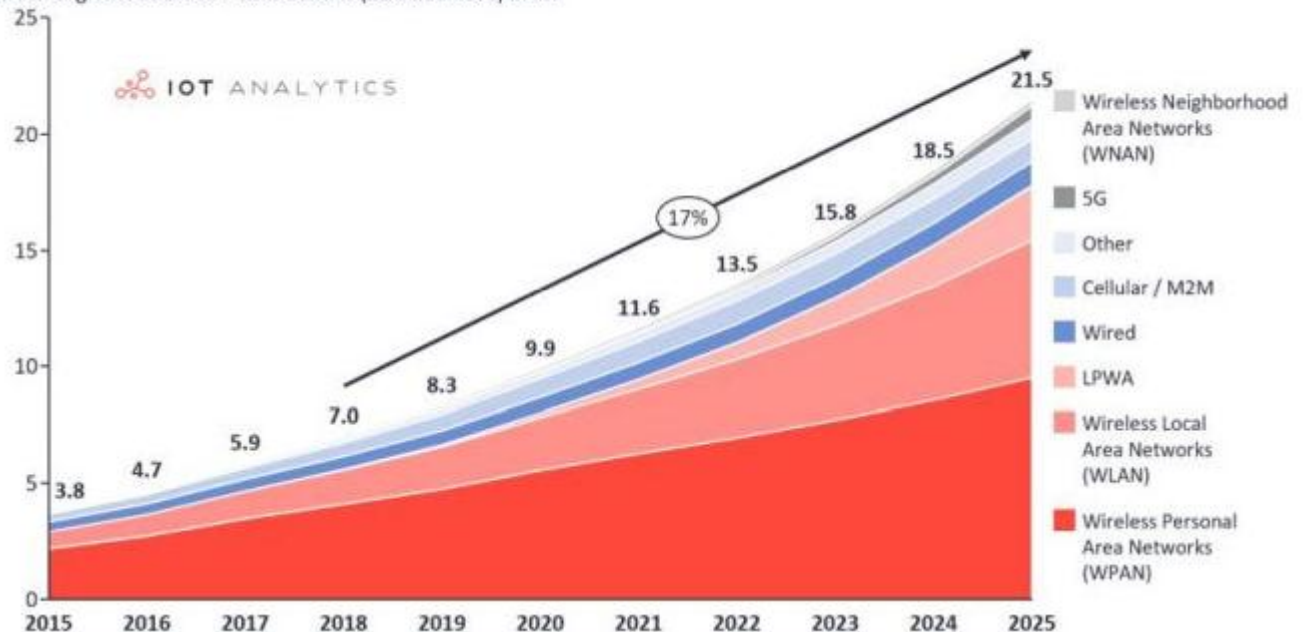


LE CREN Anthony

Bientôt tous les appareils que vous possédez, et pratiquement tous les objets qui existent seront connectés à l'Internet. Que ce soit via votre téléphone portable, des vêtements ou des appareils ménagers, nous serons connectés à l'Internet des objets (IoT). Le nombre d'objets actifs connectés à un réseau IoT était de 7 milliards d'unités dans le monde en 2018 et passera à 21,5 milliards en 2025. Cela est considéré comme la troisième évolution de l'Internet, baptisé Web 3.0 qui fait suite à l'ère du Web social.

## Global Number of Connected IoT Devices

Number of global active IoT Connections (installed base) in Bn

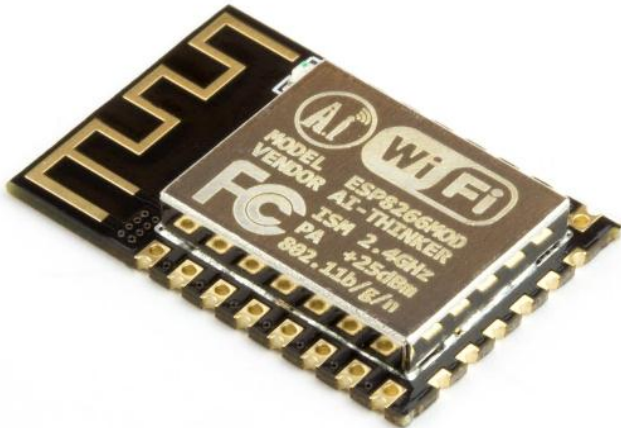


Rappel :

- Le Web 1.0 est le Web constitué de pages web liées entre elles par des hyperliens et qui a été créé au début des années 1990.
- Le Web 2.0 est le Web social, qui s'est généralisé avec le phénomène des blogs, réseaux sociaux et la technologie wiki (Un wiki est une application web qui permet la création, la modification et l'illustration collaboratives de pages à l'intérieur d'un site web).

LPWAN : Low-power wide-area network (Liaisons sans fil à faible consommation énergétique)	WIFI (Wireless Fidelity) dans un WLAN
<ul style="list-style-type: none"> <li>- Fréquence : 868Mhz</li> <li>- Puissance 20mW</li> <li>- Portée 100km max</li> <li>- Débit : 100 bit/s</li> </ul>	<ul style="list-style-type: none"> <li>- Fréquence : 2.4Ghz</li> <li>- Puissance 100mW</li> <li>- 802.11g : 54Mbps/s, 140m max</li> <li>- 802.11n : 450Mbps/s, 250m max</li> </ul>

Dans ce nouveau monde des objets connectés, le module ESP8266 est devenu une solution populaire et abordable. Cette carte électronique est un SOC (System On Chip, Système sur une puce) qui intègre un processeur Xtensa LX106, une interface RF (Radio Fréquence) et une pile TCP/IP qui permet d'implémenter un support WiFi.

Caractéristiques de l'ESP8266 (NodeMCU, ESP-12E)		
Fréquence	80Mhz	
Flash Memory	4 Mbytes	
RAM	96 Kbytes	
Broches E/S	11	
Entrée Analogique	1	
WIFI/PWM/SPI/I2C	oui	
Alimentation	3.3V	

#### Description des périphériques montés sur la carte Shield

Numéro de broche (OUT/IN)		Composant	Rôle
0	Logique (OUT)	Led rouge	Led de test
2	Logique (OUT)	Sortie RS485	Commande de spots DMX
4	Logique	SDA : Afficheur OLED	Afficheur OLED et capteur I2C divers
5	Logique	SCL : Afficheur OLED	
12	Logique	DS18S20	Capteur de température
13	Logique (IN)	VS1838B	Capteur infrarouge pour télécommande
14	Logique (OUT)	WS2812B	Ruban de 4 leds couleurs
15	Logique (OUT)	Emetteur UHF	Passerelle vers le réseau Sigfox
A0	Analogique (IN)	Capteur LDR	Capteur de lumière

Rem : Un Shield est une carte qui se branche sans soudure aux cartes ESP-12E pour augmenter leurs capacités.

## 1 Prise en main de la carte électronique : Clignoter une led

A d'aide de l'annexe 1, configurer le logiciel Thonny

Prendre connaissance des modules (annexe 2)

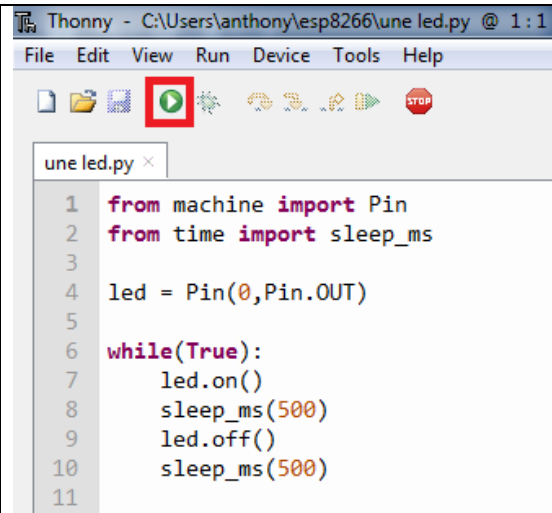
Tester le programme suivant : led.py

```
from machine import Pin
from time import sleep_ms
```

```
led = Pin(0, Pin.OUT)
```

```
while(True):
    led.on()
    sleep_ms(500)
    led.off()
    sleep_ms(500)
```

#ctrl+c pour stopper ou icône stop



```
>>> %Run test.py
```

Traceback (most recent call last):

File "C:\Users\anthony\esp8266\test.py", line 7, in <module>

KeyboardInterrupt:

**Q1** Que faut-il modifier pour allumer et éteindre la led toutes les secondes ? Vérifier expérimentalement votre solution.

```
sleep_ms(1000)
```

## 2 Mesurer la luminosité

Tester le programme suivant : ldr.py

```
from machine import ADC
from time import sleep
```

```
adc = ADC(0)
```

```
while True:
    valeur=adc.read()
    print(valeur)
    sleep(1)
```

#ctrl+c pour stopper ou icône stop

```
5 from machine import ADC
6 from time import sleep
```

```
7
8 adc = ADC(0)
```

```
9
10 while True:
11     valeur=adc.read()
12     print(valeur)
13     sleep(1)
```

**Q2** Compléter le tableau suivant

Valeur moyenne relevée avec la main au-dessous du capteur (Masque la lumière)	Valeur moyenne relevée avec un éclairage nominal (Avec de la lumière)
<b>204</b>	<b>598</b>

**Q3** Calculer la moyenne des deux valeurs précédentes

**(204+598)/2=401**

**Q4** On désire commander la led avec le capteur de lumière de la manière suivante :

- Présence de la lumière sur le capteur : led allumée
- Absence de lumière sur le capteur : led éteinte

Réaliser un programme (Q4-lumiere\_led.py) en s'inspirant des deux programmes précédents

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

```
adc = ADC(0)
led = Pin(0,Pin.OUT)
```

```
while True:
    valeur=adc.read()
    if valeur>401:
        led.on()
    else:
        led.off()
```

## 2 Ruban de 4 leds couleurs

Tester le programme suivant : neopixels.py

```
from neopixel import NeoPixel
from machine import Pin
```

```
np = NeoPixel(Pin(14), 4)
```

```
np[0] = (0, 255, 255)
```

```
np[1] = (0, 128, 0)
```

```
np[2] = (0, 0, 64)
```

```
np[3] = (64, 0, 64)
```

```
np.write()
```

#ctrl+c pour stopper ou icône stop

```
7 from neopixel import NeoPixel
8 from machine import Pin
```

```
9
10 np = NeoPixel(Pin(14), 4)
```

```
11
12 np[0] = (0, 255, 255)
```

```
13 np[1] = (0, 128, 0)
```

```
14 np[2] = (0, 0, 64)
```

```
15 np[3] = (64, 0, 64)
```

```
16
```

```
17 np.write()
```

**Q5** A quoi peuvent correspondre les valeurs (0,255,255) etc... dans le programme ci-dessus ?  
Donner la valeur minimale et maximale.

Les valeurs correspondent dans l'ordre aux composantes de couleurs rouge, vert, bleu. Ces valeurs sont comprises entre 0 et 255

**Q6** Quel est le nombre couleurs affichables sur une led ?

$256 \times 256 \times 256 = 16777216$

**Q7** Modifier le programme précédent conformément au tableau ci-dessous : (Q7-neopixels\_coul.py)

LED0	LED1	LED2	LED3
ROUGE	VERT	BLEU	JAUNE

```
from neopixel import NeoPixel
from machine import Pin
```

```
np = NeoPixel(Pin(14), 4)
```

```
np[0] = (255, 0, 0)
```

```
np[1] = (0, 255, 0)
```

```
np[2] = (0, 0, 255)
```

```
np[3] = (255, 255, 0)
```

```
np.write()
```

**Q8** On désire allumer toutes les leds de la même couleur. Ecrire un programme utilisant une boucle Pour (Q8-neopixels\_une\_coul.py)

```
from neopixel import NeoPixel
from machine import Pin

np = NeoPixel(Pin(14), 4)

for n in range(0,4):
    np[n] = (255, 0, 0)

np.write()
```

**Q9** Compléter la fonction défilement qui permet de faire défiler les leds avec une couleur verte comme un chenillard. (L'intervalle de temps sera de 100ms entre chaque led)

LED0	LED1	LED2	LED3

(Q9-neopixels\_defill.py)

```
NBPIXELS=4

from neopixel import NeoPixel
from machine import Pin
from time import sleep_ms

np = NeoPixel(Pin(14), NBPIXELS)

def defilement(color):
    for p in range(NBPIXELS):
        np[p] = color
        np[(p-1)%NBPIXELS]=(0,0,0)
        np.write()
        sleep_ms(100)

while(True):
    defilement((0,255,0))
```

**Q10** On désire commander la led avec le capteur de lumière de manière progressive

- Présence de la lumière sur le capteur : leds bleu vif
- Absence de lumière sur le capteur : led bleu sombre

Réaliser un programme (Q10-lumiere\_neopixels.py) en s'inspirant des programmes Q2 et Q8

Rappel :

	min	Max	Possibilités
Capteur de lumière	0	1023	1024
Valeur d'une composante	0	255	256

**La valeur d'une composante couleur ne doit pas dépasser 255**

NBPIXELS=4

```
from neopixel import NeoPixel
from machine import Pin,ADC
from time import sleep_ms
```

```
np = NeoPixel(Pin(14), NBPIXELS)
adc = ADC(0)
```

LUMIERE\_MIN = 300

while True:

```
    valeur=adc.read()
    print(valeur)
    .couleur=(0,0,(valeur-LUMIERE_MIN)//8)
    for n in range(0,NBPIXELS):
        np[n] = couleur
    np.write()
```

### 3 Afficheur OLED

La résolution de l'afficheur est de 128x64 pixels

Tester le programme suivant : Q11-oled.py

```
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
i2c = I2C(scl=Pin(5), sda=Pin(4))

oled = SSD1306_I2C(128, 64, i2c, 0x3c)
oled.fill(0)
oled.text("Hello NSI", 0, 0)
oled.show()
```

```
6 from machine import Pin,I2C
7 from ssd1306 import SSD1306_I2C
8 i2c = I2C(scl=Pin(5), sda=Pin(4))
9 |
10 oled = SSD1306_I2C(128, 64, i2c, 0x3c)
11 oled.fill(0)
12 oled.text("Hello NSI", 0, 0)
13 oled.show()
```

**Q11** Déplacer le texte affiché approximativement au centre de l'écran. Où sont les informations de position x et y dans la méthode text ?

```
oled.text("Hello NSI", 30, 30) # dans l'ordre x=30, y=30
```

**Q12** Réaliser un défilement du texte du haut vers le bas. (Q12-oled\_defill.py)

Remarque :

La hauteur de la police est de 7 pixels

Le transfert des données vers l'afficheur est long et le programme ne nécessite pas de temporisation.

HAUTEUR=64-7

```
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
i2c = I2C(scl=Pin(5), sda=Pin(4))
oled = SSD1306_I2C(128, 64, i2c, 0x3c)
```

```
while True:
```

```
    for h in range(0,HAUTEUR):
        oled.fill(0)
        oled.text("Hello NSI", 30, h)
        oled.show()
```



**Q13** Réaliser un défilement du texte du haut vers le bas puis du bas vers le haut sans coupure (Q13-  
oled\_defill\_haut\_bas.py)

HAUTEUR=64-7

```
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
i2c = I2C(scl=Pin(5), sda=Pin(4))
oled = SSD1306_I2C(128, 64, i2c, 0x3c)
```

```
while True:
    for h in range(0,HAUTEUR):
        oled.fill(0)
        oled.text("Hello NSI", 30, h)
        oled.show()
    for h in range(HAUTEUR,0,-1):
        oled.fill(0)
        oled.text("Hello NSI", 30, h)
        oled.show()
```

## 4 Mesure de température

Tester le programme suivant : temperature.py

```
from onewire import OneWire
from ds18x20 import DS18X20
from machine import Pin
from time import sleep_ms
import sys


bus = OneWire(Pin(12))
ds = DS18X20(bus)
capteur_temperature = ds.scan()

while True:
    try:
        ds.convert_temp()
        sleep_ms(750)
        temp_celsius =
ds.read_temp(capteur_temperature[0])
        print("Température : ",temp_celsius )
    except KeyboardInterrupt:
        print('Ctrl-C pressed...exiting')
        sys.exit()
```

```
from onewire import OneWire
from ds18x20 import DS18X20
from machine import Pin
from time import sleep_ms
import sys

bus = OneWire(Pin(12))
ds = DS18X20(bus)
capteur_temperature = ds.scan()

while True:
    try:
        ds.convert_temp()
        sleep_ms(750)
        temp_celsius = ds.read_temp(capteur_temperature[0])
        print("Température : ",temp_celsius )
    except KeyboardInterrupt:
        print('Ctrl-C pressed...exiting')
        sys.exit()
```

Remarque : la gestion d'une exception permet de sortir plus proprement du programme plutôt que de réinitialiser l'ESP8266 avec l'icône STOP 

**Q14** Afficher la température sur l'écran OLED (Q14-oled\_temperature.py)

La méthode text de l'afficheur ne gère que les chaînes de caractères. Pour afficher un nombre, utiliser la fonction str.

Exemple :

```
a=10
oled.text(str(a), 0, 20)
```

```
from onewire import OneWire
from ds18x20 import DS18X20
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
from time import sleep_ms
import sys

i2c = I2C(scl=Pin(5), sda=Pin(4))
oled = SSD1306_I2C(128, 64, i2c, 0x3c)
bus = OneWire(Pin(12))
ds = DS18X20(bus)
capteur_temperature = ds.scan()

while True:
    try:
        ds.convert_temp()
        sleep_ms(750)
        temp_celsius = ds.read_temp(capteur_temperature[0])
        print("Température : ",temp_celsius )
        oled.fill(0)
        oled.text("temperature", 0, 0)
        oled.text(str(temp_celsius), 0, 20)
        oled.show()
    except KeyboardInterrupt:
        print('Ctrl-C pressed...exiting')
        sys.exit()
```

## 5 Horloge en temps réel (RTC : Real Time Clock)

Tester le programme suivant : rtc.py

```
from machine import Pin,I2C,RTC
from time import sleep

rtc = RTC()

#(year, month, day, weekday, hours, minutes,
seconds, subseconds)
rtc.datetime((2019, 8, 2, 4,13,55, 0,0))

while True:
    horloge=rtc.datetime()
    print(horloge)
    sleep(1)
```

```
from machine import Pin,I2C,RTC
from time import sleep

rtc = RTC()

#(year, month, day, weekday, hours, minutes, seconds, subseconds)
rtc.datetime((2019, 8, 2, 4,13,55, 0,0))

while True:
    horloge=rtc.datetime()
    print(horloge)
    sleep(1)
```

Weekday : jour de la semaine : lundi = 0...dimanche = 6

**Q15** A partir du tuple horloge, extraire les heures, minutes, secondes. Afficher l'heure sous la forme heure:minutes:secondes , exemple 15:12:35 (Q15-rtc\_hms.py)

```
print(horloge[4],':',horloge[5],':',horloge[6])
```

**Q16** Afficher l'heure au centre de l'écran OLED (Q16-rtc\_hms\_oled.py)

```
from machine import Pin,I2C,RTC
from time import sleep
from ssd1306 import SSD1306_I2C
i2c = I2C(scl=Pin(5), sda=Pin(4))
oled = SSD1306_I2C(128, 64, i2c, 0x3c)

rtc = RTC()

#(year, month, day, weekday, hours, minutes, seconds, subseconds)
rtc.datetime((2019, 8, 2, 4,13,55, 0,0))

while True:
    horloge=rtc.datetime()
    #heure=str(horloge[4])+':' +str(horloge[5])+':' +str(horloge[6])
    #ou
    heure= s = "%02d" % horloge[4]+':'+"%02d" % horloge[5]+':'+"%02d" % horloge[6]
    oled.fill(0)
    oled.text(heure ,30, 40)
    oled.show()
    sleep(1)
```

## Mini projets

**Q17** Défiler des couleurs différentes sur le ruban de led

**Q18** Déclencher le défilement du ruban de leds en fonction de la luminosité ambiante

**Q19** Les couleurs du ruban changent en fonction de la température ambiante

**Q20** Gérer l’afficheur OLED en ajoutant :

- La température ambiante
- La température maximale
- La température minimale

**Q21** Tracer une courbe image de la température sur l’afficheur OLED (mesure toutes les secondes)

oled.pixel(10 ,20, 1) #x,y,pixel allumé ou éteint 0/1

**Q22** Clignoter la led rouge plus moins rapidement en fonction de la luminosité ambiante

**Q23** Réaliser un dégradé de couleur avec le ruban de leds

**Q24** Réaliser un compteur binaire (0 à 15) avec le ruban de leds

**Q25** tracer un cercle sur l’afficheur OLED et afficher l’information de température au centre du cercle.

**Q26** Reprendre la question Q16 et ajouter la date sous la forme suivante

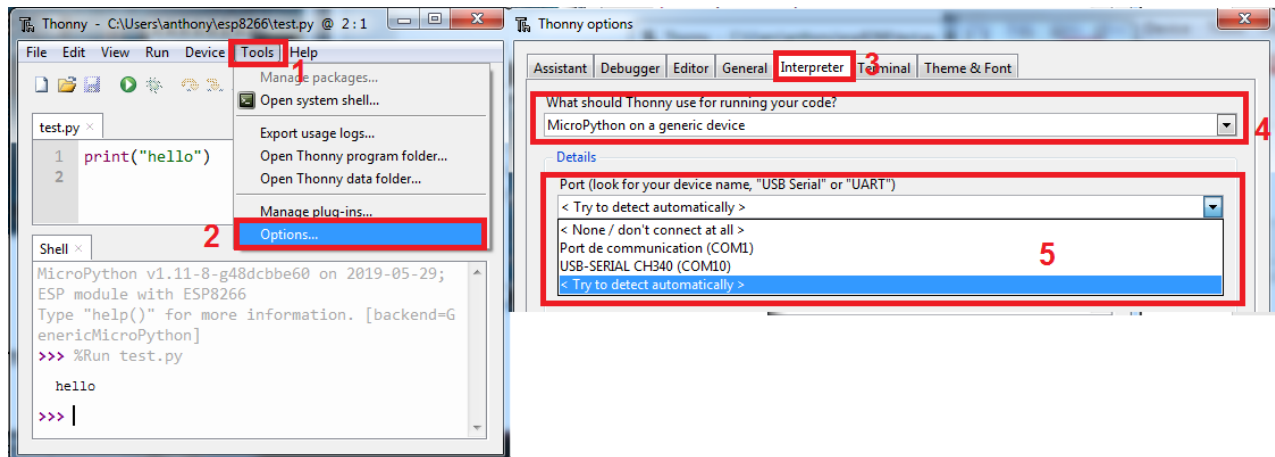
<p><b>Lundi</b> <b>16 septembre</b> <b>2019</b> 15 :45 :32</p>
--

**Q27** Votre projet personnel (le programme de votre choix)

## ANNEXE 1

Lancer le logiciel Thonny

Configurer le logiciel en mode esp8266



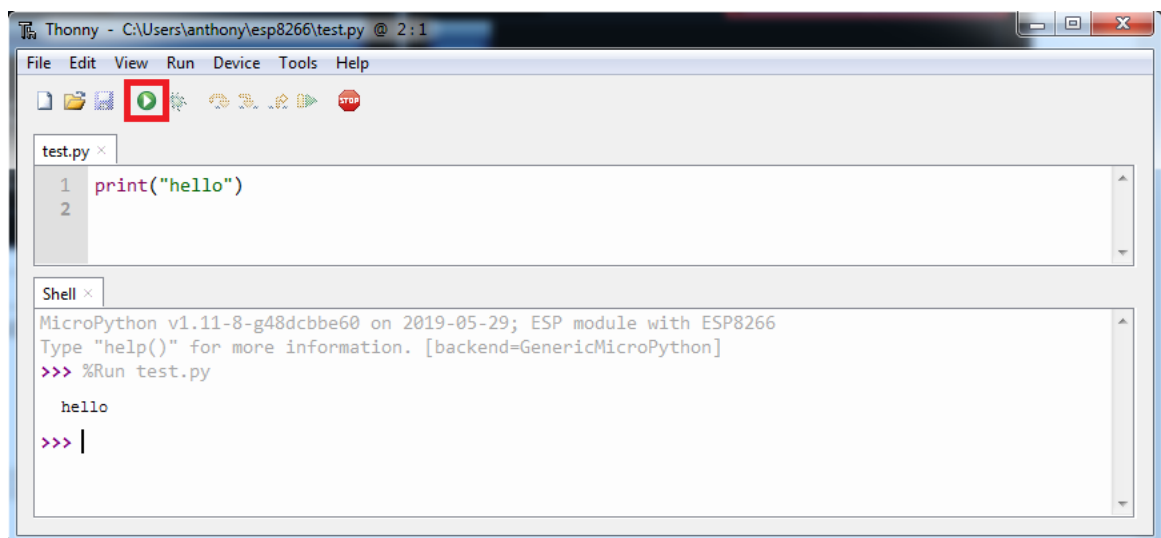
Relier la carte sur un port USB du PC

Ajouter une photo ici

Attendre la reconnaissance des pilotes si nécessaire

Taper CTRL+F2 pour se connecter à la carte depuis le logiciel Thonny

Tester le bon fonctionnement à l'aide d'un print("hello")



<https://docs.micropython.org/en/latest/esp8266/tutorial/intro.html>

## Annexe 2

<http://docs.micropython.org/en/v1.9.3/esp8266/esp8266/tutorial/index.html>

Quels sont les modules déjà intégrés ?

```

Shell x
MicroPython v1.11-8-g48dcbb60 on 2019-05-29; ESP module with ESP8266
Type "help()" for more information. [backend=GenericMicroPython]
>>> help("modules")

__main__      http_client    socket          upip
__boot        http_client_ssl ssd1306 ■       upip_utarfile
__onewire     http_server     ssl             upysh
__webrepl     http_server_ssl struct          urandom
apa102 ■      inisetup        sys            ure
array         io              time           urequests
binascii      json            uasyncio/___init___ urllib/urrequest
btree         lwip            uasyncio/core  uselect
builtins      machine ■       ubinascii      usocket
collections   math            ucollections   ussl
dht ■         micropython    ucryptolib     ustruct
ds18x20 ■     neopixel ■     uctypes        utime
errno         network        uerrno         utimeq
esp           ntptime        uhashlib       uwebsocket
example_pub_button uio            onewire        uheapq          uzlib
example_sub_led  os             ujson          webrepl
flashbdev       port_diag      umqtt/robust ■ webrepl_setup
framebuf        random         umqtt/simple   websocket_helper
gc              re              uos            zlib
hashlib         select
Plus any modules on the filesystem

>>> |

```

En rouge le hardware géré en natif. On remarque le protocole mqtt intéressant pour la gestion des IOT.

En vert (machine). C'est le module qui gère les périphériques.

<http://docs.micropython.org/en/v1.9.3/esp8266/library/machine.html>

class Pin – control I/O pins  
 class Signal – control and sense external I/O devices  
 class UART – duplex serial communication bus  
 class SPI – a Serial Peripheral Interface bus protocol (master side)  
 class I2C – a two-wire serial protocol  
 class RTC – real time clock  
 class Timer – control hardware timers  
 class WDT – watchdog timer

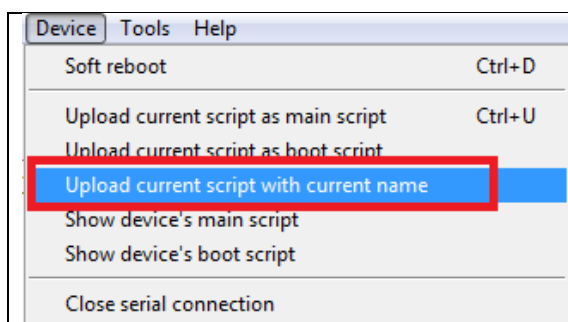
Il est possible d'ajouter des modules en faisant :

Liste des méthodes de la classe machine

```

>>> import machine
>>> dir(machine)
['__class__', '__name__', 'ADC', 'DEEPSLEEP',
'DEEPSLEEP_RESET', 'HARD_RESET', 'I2C', 'PWM',
'PWRON_RESET', 'Pin', 'RTC', 'SOFT_RESET', 'SPI',
'Signal', 'Timer', 'UART', 'WDT', 'WDT_RESET',
'deepsleep', 'disable_irq', 'enable_irq', 'freq',
'idle', 'lightsleep', 'mem16', 'mem32', 'mem8',
'reset', 'reset_cause', 'sleep', 'time_pulse_us',
'unique_id']

```



### A noter :

L'esp8266 en mode micropython contient 2 fichiers importants.

**Boot.py** : est l'équivalent du setup sur Arduino. (Programme exécuté au démarrage)  
 Ensuite si l'esp contient un fichier **main.py**, alors celui-ci sera exécuté juste après le boot.py