



Les objets connectés

(Partie 2 : connexion Wi-Fi)



LE CREN Anthony

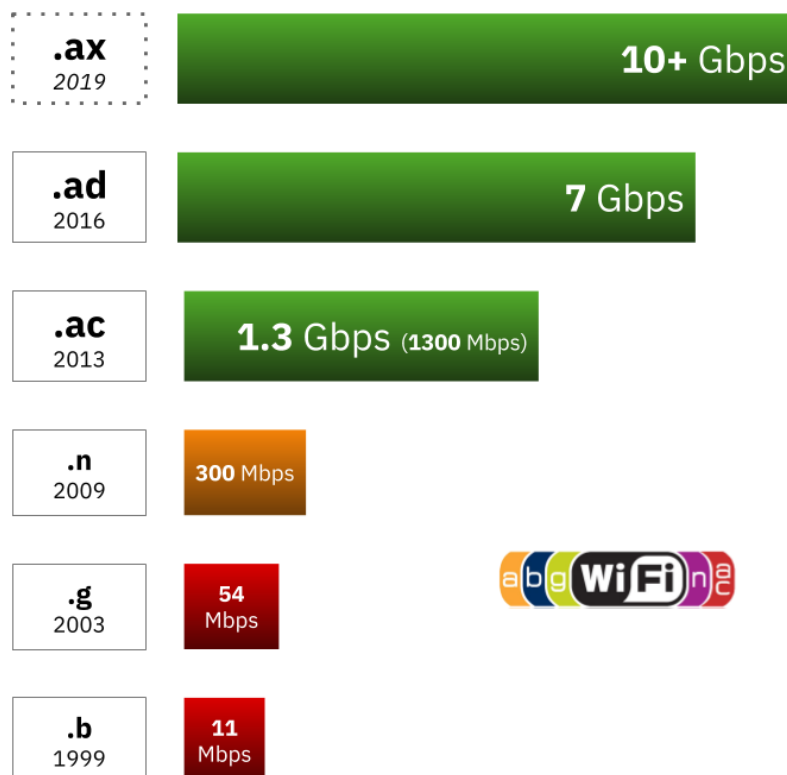
Le Wi-Fi, est un ensemble de protocoles de communication sans fil régis par les normes du groupe **IEEE 802.11**

On estime qu'il y a aujourd'hui 10 milliards de périphériques sans fils dans le monde. Si on les reporte à la population mondiale humaine estimée à 7 milliards d'individus cela nous donne 1.43 périphériques par personnes. En 2020 l'augmentation des clients Wi-Fi fera que nous aurons en moyenne 3.75 périphériques dans les mains.

WIFI (Wireless Fidelity) dans un WLAN
- Fréquence : 2.4Ghz/5Ghz
- Puissance 100mW
- 802.11g : 54Mbps/s, 140m max
- 802.11n : 450Mbps/s, 250m max

Avec la norme IEEE 802.11ax, le Wi-Fi s'apprête à investir le champ d'applications relatif à l'Internet des Objets. Les débits sont 4 fois plus élevés que ceux atteints avec la norme IEEE 802.11ac (11 Gbit/s)

Plusieurs utilisateurs vont pouvoir transmettre leurs données à un point d'accès Wi-Fi unique **en même temps**. Pour ce faire, l'idée est d'utiliser un mécanisme présent dans la norme mobile 4G-LTE et de l'adapter au Wi-Fi : l'OFDMA (orthogonal frequency-division multiple access)



1 Connexion au point d'accès Wi-Fi

Voir Annexe 2 : Installer un point d'accès Wifi RaspAP

Tester le programme suivant : wifi_connexion.py

```
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())
```

```
5 import network
6
7 # user data
8 ssid = "raspi-webgui" # wifi router name
9 pw = "touchardNSI" # wifi router password
10
11 # wifi connection station mode
12 wifi = network.WLAN(network.STA_IF)
13 wifi.active(True)
14 wifi.connect(ssid, pw)
15
16 # wait for connection
17 while not wifi.isconnected():
18     pass
19
20 # wifi connected
21 print(wifi.ifconfig())
```

===== RESTART =====

```
>>> %Run wifi_cnx.py
```

```
('192.168.1.104', '255.255.255.0', '192.168.1.1', '192.168.1.1')
```

La configuration est automatiquement mémorisée dans l'esp8266. Si l'on veut savoir si la connexion est effective au prochain démarrage de la carte, exécuter les instructions suivantes :

(wifi_isconnected.py)

```
import network

# wifi connection station mode
wifi = network.WLAN(network.STA_IF)

# wifi is connected ?
if wifi.isconnected():
    print ("connecté au réseau wifi")
    print(wifi.ifconfig())
else:
    print (" NON connecté au réseau wifi")
```

```
import network

# wifi connection station mode
wifi = network.WLAN(network.STA_IF)

# wifi is connected ?
if wifi.isconnected():
    print ("connecté au réseau wifi")
    print(wifi.ifconfig())
else:
    print (" NON connecté au réseau wifi")
```

Q1 Afficher l'adresse IP de l'esp8266 au centre de l'écran OLED. (Q1-wifi_isconnected_oled.py)

```
import network
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

# wifi connection station mode
wifi = network.WLAN(network.STA_IF)

i2c = I2C(scl=Pin(5), sda=Pin(4))
oled = SSD1306_I2C(128, 64, i2c, 0x3c)

# wifi is connected ?
if wifi.isconnected():
    ip=wifi.ifconfig()
    oled.text(ip[0],10, 40)
else:
    oled.text("not connected",10, 40)
oled.show()
```

Comment afficher l'adresse MAC ?

```
import network
import ubinascii

# wifi connection station mode
wifi = network.WLAN(network.STA_IF)

# wifi is connected ?
if wifi.isconnected():
    ip=wifi.ifconfig()
    print(ip[0])
    mac = ubinascii.hexlify(network.WLAN().config('mac'), ':').decode()
    print(mac)
else:
    print(" NON connecté au réseau wifi")
```

2 Obtenir l'heure depuis le serveur worldtimeapi.org

Q2 A partir du lien ci-dessous, Identifier l'URL permettant de récupérer l'heure en fonction du lieu géographique.

<http://worldtimeapi.org/timezones>

<http://worldtimeapi.org/timezone/Europe/Paris>

Q3 Quel est La norme internationale décrivant de format de l'heure

La norme internationale ISO 8601 spécifie la représentation numérique de la date et de l'heure respectivement basées sur le calendrier grégorien et le système horaire de 24 heures.

Q4 A partir du lien ci-dessous, vérifier la réponse du serveur dans un navigateur

<http://worldtimeapi.org/api/timezone/Europe/Paris>

JSON	Données brutes	En-têtes
Enregistrer	Copier	Tout réduire
	Tout développer	Filtrer le JSON
week_number:	31	
utc_offset:	" +02:00 "	
utc_datetime:	" 2019-08-03T08:28:18.933946+00:00 "	
unixtime:	1564820898	
timezone:	"Europe/Paris"	
raw_offset:	3600	
dst_until:	" 2019-10-27T01:00:00+00:00 "	
dst_offset:	3600	
dst_from:	" 2019-03-31T01:00:00+00:00 "	
dst:	true	
day_of_year:	215	
day_of_week:	6	
datetime:	" 2019-08-03T10:28:18.933946+02:00 "	
client_ip:	"86.212.16.247"	
abbreviation:	"CEST"	

Q5 Quel champ de données sera prise en compte pour récupérer l'heure locale ?

datetime "2019-08-03T10:28:18.933946+02:00"

Q6 Tester le programme suivant : Q6-worldtime.py

```
import network
import urequests
import ujson
import utime

# user data
url = "http://worldtimeapi.org/api/timezone/Europe/Paris"

# initialization

response = urequests.get(url)

if response.status_code == 200:
    worldtime = ujson.loads(response.text)
    print(type(worldtime))
    print(worldtime.keys())
    print(worldtime)

else:
    print("Pas de réponse")
```

Q7 Quel est le type de la variable worldtime ?

<class 'dict'>

Q8 Donner l'adresse IP du client

86.212.16.247

Q9 Compléter les indices dans le programme suivant afin de récupérer séparément l'année, le mois, le jour, les heures, les minutes, les secondes : Q9-worldtime_indices.py

2	0	1	9	-	0	8	-	0	3	T	1	0	:	4	0	:	1	5	.	2	7	3	7	4	7	+	0	2	:	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

2019-08-03T10:40:15.273747+02:00

```
import network
import urequests
import ujson
import utime

# user data
url = "http://worldtimeapi.org/api/timezone/Europe/Paris"

# initialization

response = urequests.get(url)

if response.status_code == 200:
    worldtime = ujson.loads(response.text)
    print(type(worldtime))
    print(worldtime.keys())
    print(worldtime)
    horloge = worldtime["datetime"]
    print(horloge)
    annee = int(horloge[0:4])
    mois = int(horloge[5:7])
    jour = int(horloge[8:10])
    heure = int(horloge[11:13])
    minute = int(horloge[14:16])
    seconde = int(horloge[17:19])
    subsecond = int(horloge[20:23])
    print(annee, mois, jour, heure, minute, seconde, subsecond)
else:
    print("Pas de réponse")
```

Q10 Précédemment dans la 1ere partie, une horloge avait été réalisée. Ajouter des instructions permettant de mettre à jour l'horloge dès de démarrage du programme. Q10-worldtime_oled.py

```
#rappel de l'horloge sur écran OLED
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= s = "%02d" % horloge[4]+':'+"%02d" % horloge[5]+':'+"%02d" % horloge[6]
    oled.fill(0)
    oled.text(heure,30, 40)
    oled.show()
    sleep(1)
```

```

import network
import urequests
import ujson
import utime
from machine import Pin,I2C,RTC
from time import sleep
from ssd1306 import SSD1306_I2C

# user data
url = "http://worldtimeapi.org/api/timezone/Europe/Paris"

# initialization
i2c = I2C(scl=Pin(5), sda=Pin(4))
oled = SSD1306_I2C(128, 64, i2c, 0x3c)
rtc = RTC()

response = urequests.get(url)

if response.status_code == 200:
    worldtime = ujson.loads(response.text)
    print(type(worldtime))
    print(worldtime.keys())
    print(worldtime)
    horloge = worldtime["datetime"]
    print(horloge)
    annee = int(horloge[0:4])
    mois = int(horloge[5:7])
    jour = int(horloge[8:10])
    heure = int(horloge[11:13])
    minute = int(horloge[14:16])
    seconde = int(horloge[17:19])
    subsecond = int(horloge[20:23])
    print(annee, mois, jour, heure, minute, seconde, subsecond)
    rtc.datetime((annee, mois, jour, 0,heure, minute, seconde, subsecond))
else:
    print("Pas de réponse")
    rtc.datetime((2019, 8, 2, 0,13,55, 0,0))

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

```

3 Protocole UDP pour commander une led

Tester le programme suivant : udp_led.py

```
from machine import Pin
#import usocket as socket
import socket

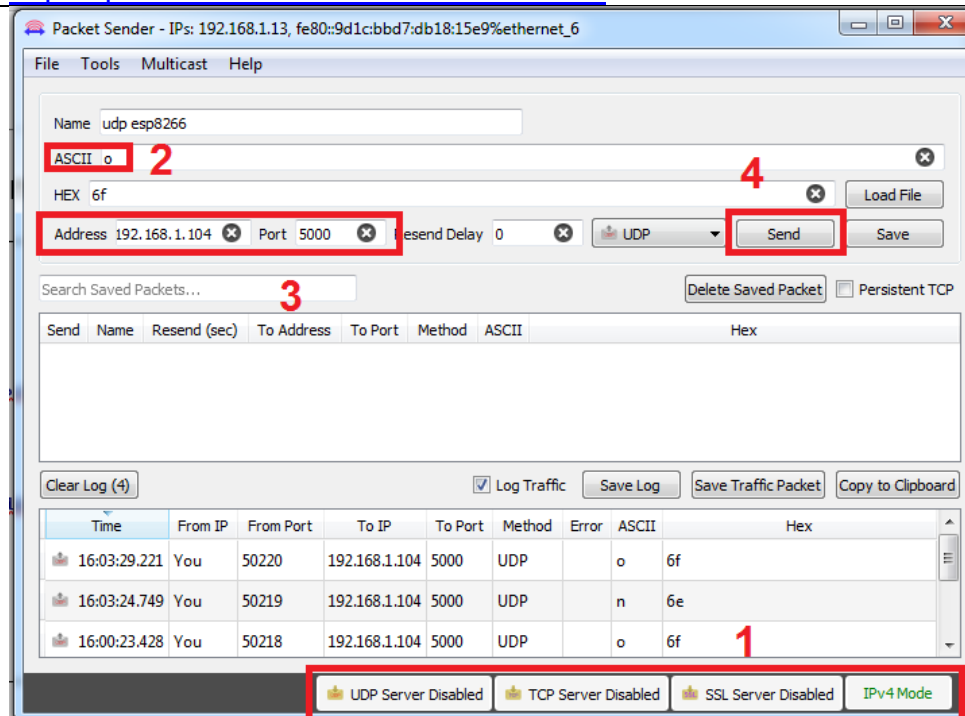
led = Pin(0,Pin.OUT)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('', 5000))
print("A l'écoute du port 5000")
while True:
    data, addr = s.recvfrom(1024)
    print('received:',data,'from',addr)
    if (data.decode()=='o'):
        led.on()
    elif (data.decode()=='n'):
        led.off()
```

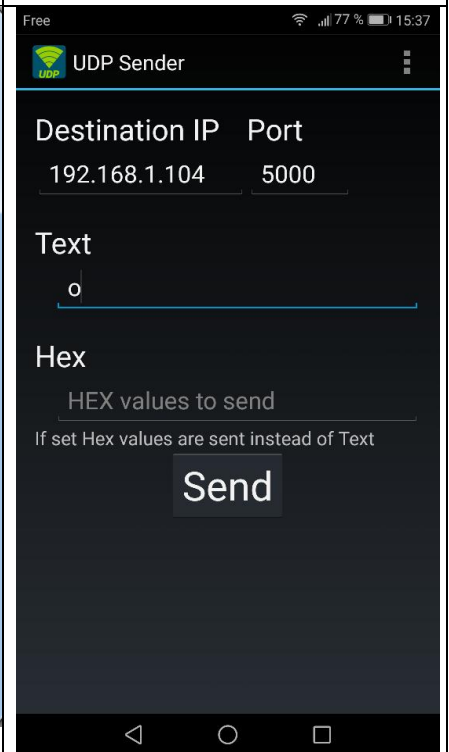
```
1 from machine import Pin
2 #import usocket as socket
3 import socket
4
5 led = Pin(0,Pin.OUT)
6
7 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8 s.bind(('', 5000))
9 print("A l'écoute du port 5000")
10 while True:
11     data, addr = s.recvfrom(1024)
12     print('received:',data,'from',addr)
13     if (data.decode()=='o'):
14         led.on()
15     elif (data.decode()=='n'):
16         led.off()
```

Packet Sender sous Windows

<https://packetsender.com/download#show>



UDP Sender sous Android




Q11 Dans quel matériel se trouve le serveur UDP et le client. Quel est le port d'écoute du serveur ?

Le serveur UDP est sur la carte ESP8266

Le client est le logiciel packet sender sous windows/linux ou UDP Sender sous Android/iOS

Le port d'écoute du serveur est 5000

Q12. Utiliser le logiciel Wireshark  afin de capturer une trame de communication entre le client et le serveur WEB de l'ESP8266 pour allumer la led (coller la trame dans le cadre suivant). Analyser cette trame Ethernet et compléter le tableau ci-dessous.

```

0000  a0 20 a6 21 9d fa 9c 5c 8e 00 b2 d9 08 00 45 00  . . ! . . . \ . . . . . E .
0010  00 1d 40 aa 00 00 80 11 76 60 c0 a8 01 0d c0 a8  . . @ . . . . v ` . . . . .
0020  01 68 eb e3 13 88 00 09 0d aa 6f                . h . . . . .  o

```

Adresse MAC destination (serveur)		A0:20:a6:21:9d:fa
Adresse MAC source (client)		9c:5c:8e:00:b2:d9
IP	Adresse IP source (serveur)	192.168.1.13
	Adresse IP destination (client)	192.168.1.104
UDP	Port source	60387
	Port destination	5000
	Données transmises	o (0x6f)

IP	Longueur en-tête	20
	Longueur totale du datagramme	29

UDP	Longueur en-tête	8
	Longueur des données	1

Q13 A partir du document « Les reseaux NSI_cours », rappeler le principal inconvénient du protocole UDP

Rien ne permet de savoir si le paquet est bien arrivé (il n'y a aucun accusé de réception).

Correction :

92	29.951362	192.168.1.13	192.168.1.104	UDP	43	60386 → 5000	Len=1
▶	Frame 94: 43 bytes on wire (344 bits), 43 bytes captured (344 bits) on interface 0						
▲	Ethernet II, Src: AsustekC_00:b2:d9 (9c:5c:8e:00:b2:d9), Dst: Espressi_21:9d:fa (a0:20:a6:21:9d:fa)						
	▶ Destination: Espressi_21:9d:fa (a0:20:a6:21:9d:fa)						
	▶ Source: AsustekC_00:b2:d9 (9c:5c:8e:00:b2:d9)						
	Type: IPv4 (0x0800)						
▲	Internet Protocol Version 4, Src: 192.168.1.13, Dst: 192.168.1.104						
	0100 = Version: 4						
 0101 = Header Length: 20 bytes (5)						
▶	Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)						
	Total Length: 29						
	Identification: 0x40aa (16554)						
▶	Flags: 0x0000						
	Time to live: 128						
	Protocol: UDP (17)						
	Header checksum: 0x7660 [validation disabled]						
	[Header checksum status: Unverified]						
	Source: 192.168.1.13						
	Destination: 192.168.1.104						
▲	User Datagram Protocol, Src Port: 60387, Dst Port: 5000						
	Source Port: 60387						
	Destination Port: 5000						
	Length: 9						
	Checksum: 0x0daa [unverified]						
	[Checksum Status: Unverified]						
	[Stream index: 39]						
▲	Data (1 byte)						
	Data: 6f						
	[Length: 1]						
0000	a0 20 a6 21 9d fa 9c 5c 8e 00 b2 d9 08 00 45 00						
0010	00 1d 40 aa 00 00 80 11 76 60 c0 a8 01 0d c0 a8						
0020	01 68 eb e3 13 88 00 09 0d aa 6f						

3 Commande d'une led avec une page web

Tester le programme suivant : web_led.py

```
from machine import Pin
import usocket as socket

led = Pin(0, Pin.OUT)

def web_page():
    if led.value() == 1:
        gpio_state="ON"
    else:
        gpio_state="OFF"

    html = """<html>
        <head>
            <title>ESP LED Web Server</title>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
        </head>
        <body>
            <h1>ESP Web Server</h1>
            <p>GPIO state: "" + gpio_state + ""</p>
            <form>
                <p><button name="LED" value="ON" type="submit">LED ON</button></p>
                <p><button name="LED" value="OFF" type="submit">LED OFF</button></p>
            </form>
        </body>
    </html>"""

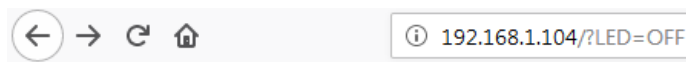
    return html

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Crée une socket
s.bind(('', 80)) #liez le socket à une adresse et un port
s.listen(5) #ecoute le port 80

while True:
    connexion, adresse = s.accept() #le serveur accepte la connexion entrante sur le port 80
    print('Connection du client', adresse)
    requete = str(connexion.recv(1024)) #récupère la requete client
    print('Content = ', requete)
    led_on = requete.find('/?LED=ON') #recherche l'indice de /?LED=ON dans la requete client
    led_off = requete.find('/?LED=OFF')
    print(led_on, led_off)
    if led_on == 6: #l'indice est en position 6 alors allume la led
        print('LED ON')
        led.on()
    if led_off == 6:
        print('LED OFF')
        led.off()
    reponse = web_page() #envoi de la page html
    connexion.send('HTTP/1.1 200 OK\n')
    connexion.send('Content-Type: text/html\n')
    connexion.send('Connection: close\n\n')
    connexion.sendall(reponse)
    connexion.close()
```

Q14 A l'aide de l'annexe 1, analyser le déroulement du programme.

Avec un ordinateur portable ou un smartphone connecté au point d'accès WIFI, saisir L'URL de l'ESP8266 afin de commander l'allumage de la led.



ESP Web Server

GPIO state: OFF

LED ON

LED OFF

Q15 Ajouter une feuille de style dans le programme précédent (Q15-web_led_css.py)

Ne pas oublier d'ajouter les classes dans le code HTML

```
<p><button class="button1" name="LED" value="ON" type="submit">LED ON</button></p>
```

Exemple à modifier suivant vos goûts :

<pre><style> html { font-family: Helvetica; display:inline-block; margin: 0px auto; text-align: center; } h1{ color: #0F3376; padding: 2vh; } p{ font-size: 1.5rem; } </pre>	<pre>.button1{ display: inline-block; background-color: #e7bd3b; border: none; border-radius: 4px; color: white; padding: 16px 40px; text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer; } .button2{ background-color: #4286f4; } </style></pre>
<pre></head> <body> <h1>ESP Web Server</h1> <p>GPIO state: "" + gpio_state + ""</p> <p><button class="button1">ON</button></p> <p><button class="button1 button2">OFF</button></p> </body></html>""</pre>	

ESP Web Server

GPIO state: OFF

ON

OFF

4 Commande d'une led avec putty

Tester le programme suivant : led_putty.py

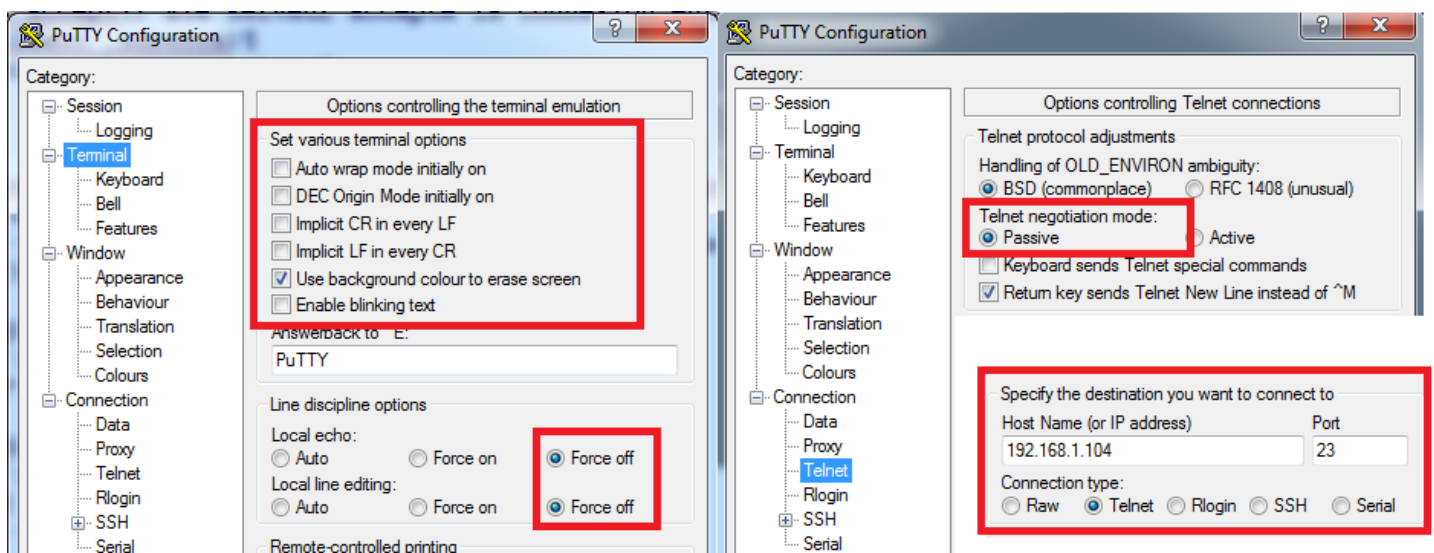
```
from machine import Pin
import usocket as socket

led = Pin(0, Pin.OUT)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Crée une socket
s.bind(('', 23)) #liez le socket à une adresse et un port
s.listen(5) #ecoute le port 80

while True:
    connexion, adresse = s.accept() #le serveur accepte la connexion entrante sur le port 80
    print('Connexion du client', adresse)
    connexion.send('connexion au serveur\n\r')
    print(connexion)
    cnx=True
    while cnx==True:
        requete = connexion.recv(1024).decode('ascii') #récupère la requete client
        print('Requete du client = ', requete)
        if requete == 'a':
            print('LED ON')
            led.on()
            connexion.send('led allumée\n\r')
        elif requete == 'e':
            print('LED OFF')
            led.off()
            connexion.send('led éteinte\n\r')
        elif requete == 'q':
            print('Quit')
            led.off()
            cnx=False
        connexion.close()
```

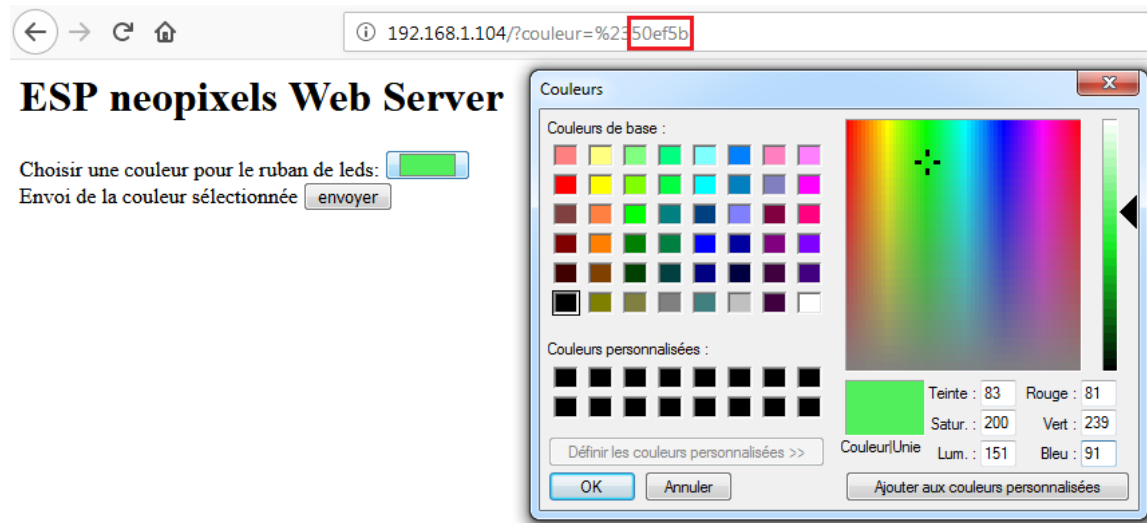
Configuration de putty en mode Telnet (Attention à votre adresse IP)



Q16 A l'aide de Wireshark, Analyser les trames entre le client et le serveur comme la question Q12

4 Commande du ruban de led avec une page web

Cahier des charges : On désire créer une page web qui va modifier la couleur du ruban de leds lorsque l'on appuie sur le bouton envoyer.



On donne la page html suivante

```
html = """<html>
  <head>
    <title>ESP neopixels Web Server</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <h1>ESP neopixels Web Server</h1>
    <form>
      Choisir une couleur pour le ruban de leds:
      <input type="color" name="couleur" id="couleur"/>
      <br>
      Envoi de la couleur sélectionnée
      <input type="submit" value="envoyer" />
    </form>
  </body>
</html>"""
```

Q17 En utilisant le code python de la question **Q14**, établir un programme qui remplit le cahier des charges. (Q17- web_led_ruban.py)

Aide :

Convertir une chaine hexadécimale en entier :

rouge = int("ef",16) # la variable rouge contient 239

Correction :

```
from machine import Pin
import usocket as socket
from neopixel import NeoPixel
from machine import Pin, ADC
```

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

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

```
def web_page():
    html = """<html>
        <head>
        <title>ESP neopixels Web Server</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        </head>
        <body>
        <h1>ESP neopixels Web Server</h1>
        <form>
            Choisir une couleur pour le ruban de leds:
            <input type="color" name="couleur" id="couleur"/>
            <br>
            Envoi de la couleur sélectionnée
            <input type="submit" value="envoyer" />
        </form>
        </body>
        </html>"""

    return html
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.listen(5)
```

```
while True:
    conn, addr = s.accept()
    print('Got a connection from %s' % str(addr))
    request = conn.recv(1024)
    request = str(request)
    print('Content = %s' % request)
```

```
position_couleur = request.find('%')
print(position_couleur)
if position_couleur==16:
    rouge = int(request[19:21],16)
    vert = int(request[21:23],16)
    bleu = int(request[23:25],16)
    print(rouge,vert,bleu)
    couleur=(rouge,vert,bleu)
    for n in range(0,NBPIXELS):
        np[n] = couleur
    np.write()
    response = web_page()
    conn.send('HTTP/1.1 200 OK\n')
    conn.send('Content-Type: text/html\n')
    conn.send('Connection: close\n\n')
    conn.sendall(response)
    conn.close()
```

5 Lecture de la température avec une page HTML

Q18 Ajouter les instructions permettant de lire la température et la stocker dans la variable `temp_celsius`

Q18-web_temperature.py

```
from machine import Pin
import usocket as socket
from machine import Pin
from onewire import OneWire
from ds18x20 import DS18X20
from time import sleep_ms

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

temp_celsius=0

def web_page():
    html = """<html>
        <head>
            <title>ESP neopixels Web Server</title>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
        </head>
        <body>
            <h1>Température de la pièce: """+str(temp_celsius)+""" °C
        </h1>
        </body>
    </html>"""

    return html

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.listen(5)

while True:
    conn, addr = s.accept()
    print('Got a connection from %s' % str(addr))
    request = conn.recv(1024)
    request = str(request)
    print('Content = %s' % request)
    ds.convert_temp()
    sleep_ms(750)
    temp_celsius = ds.read_temp(capteur_temperature[0])
    print("Température : ",temp_celsius )
    response = web_page()
    conn.send('HTTP/1.1 200 OK\n')
    conn.send('Content-Type: text/html\n')
    conn.send('Connection: close\n\n')
    conn.sendall(response)
    conn.close()
```

Q19 Quel est l'inconvénient de ce programme ?

Pour connaître la température, on est obligé de rafraîchir la page manuellement.

Tester le programme suivant : web_temperature_ajax.py (facultatif)

```

from machine import Pin
import usocket as socket
from machine import Pin
from onewire import OneWire
from ds18x20 import DS18X20
from time import sleep_ms

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

temp_celsius=0

def web_page():
    html = """<html>
        <head>
            <title>ESP neopixels Web Server</title>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <script>
                setInterval(function() {
                    getData();
                }, 2000); //mise à jour toutes les 2000 milli Secondes
                function getData() {
                    var xhttp = new XMLHttpRequest();
                    xhttp.onreadystatechange = function() {
                        if (this.readyState == 4 && this.status == 200) {
                            document.getElementById("TEMPValue").innerHTML = this.responseText;
                        }
                    };
                    xhttp.open("GET", "TEMPValue", true);
                    xhttp.send();
                }
            </script>
        </head>
        <body>
            <h1>Température de la pièce: <span id="TEMPValue">"""+str(temp_celsius)+"""</span> °C
        </h1>
        </body>
    </html>"""

    return html

def conversion_temerature():
    global temp_celsius
    ds.convert_temp()
    temp_celsius = ds.read_temp(capteur_temperature[0])
    print("Température : ",temp_celsius )

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.listen(5)

while True:
    conn, addr = s.accept()
    print('Got a connection from %s' % str(addr))
    request = conn.recv(1024)
    request = str(request)
    print('Content = %s' % request)
    root = request.find('/')
    temp = request.find('/TEMPValue')
    conversion_temerature()
    if root==6:
        conn.send('HTTP/1.1 200 OK\n')
        conn.send('Content-Type: text/html\n') #html
        conn.send('Connection: close\n\n')
        response = web_page()
        conn.sendall(response)
        conn.close()
    elif temp==6:
        conn.send('HTTP/1.1 200 OK\n')
        conn.send('Content-Type: text/plane\n') #que du texte
        conn.send('Connection: close\n\n')
        conn.sendall(str(temp_celsius))
        conn.close()

```

Mini projets

Q20 Utiliser le protocole UDP pour commander le ruban de Led avec des couleurs préprogrammées

Exemple :

- Touche a : couleur rouge,
- Touche b : couleur vert,
- Etc...

Q21 Utiliser l'utilitaire putty en TCP afin de changer le texte affiché sur l'écran OLED

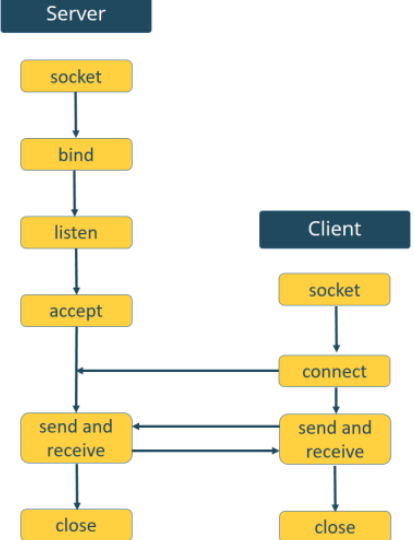
Q22 Réaliser une commande complète du ruban de leds à l'aide d'une page html

- Gestion des couleurs
- Gestion marche/arrêt avec 2 boutons

Q23 Votre projet personnel (le programme de votre choix)

Annexe 1 : Les sockets

On peut le traduire le mot socket par « connecteur réseau » ou « interface de connexion ». Il s'agit d'une interface logicielle avec les services du système d'exploitation, grâce à laquelle un développeur exploitera facilement et de manière uniforme les services d'un protocole réseau. Il lui sera ainsi par exemple aisé d'établir une session TCP, puis de recevoir et d'expédier des données grâce à elle.

 <pre> graph TD subgraph Server S_socket[socket] --> S_bind[bind] S_bind --> S_listen[listen] S_listen --> S_accept[accept] S_accept --> S_send[send and receive] S_send --> S_close[close] end subgraph Client C_socket[socket] --> C_connect[connect] C_connect --> C_send[send and receive] C_send --> C_close[close] end S_listen --> C_connect S_send <--> C_send </pre>	<p>Crée une socket à l'aide de <code>socket.socket ()</code> et spécifie le type de socket. C'est un socket TCP:</p> <pre>s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)</pre> <p>Ensuite, liez le socket à une adresse (interface réseau et numéro de port) à l'aide de la méthode <code>bind ()</code></p> <pre>s.bind('', 80)</pre> <p>La chaîne vide fait référence à l'adresse IP localhost (c'est l'adresse IP ESP32 ou ESP8266).</p> <p>La ligne suivante permet au serveur d'accepter les connexions (écoute). L'argument spécifie le nombre maximal de connexions en file d'attente.</p> <pre>s.listen(5)</pre>
<p>Dans la boucle while, il faut écouter les demandes et envoyer les réponses. Lorsqu'un client se connecte, le serveur appelle la méthode <code>accept ()</code> pour accepter la connexion. Il enregistre un nouvel objet « connexion » afin d'envoyer des données, ainsi que l'adresse du client « adresse »</p> <pre> connexion, adresse = s.accept() print('Connection du client', addr) </pre>	
<p>La méthode <code>recv ()</code> reçoit les données du socket client. L'argument de la méthode <code>recv ()</code> spécifie le maximum de données pouvant être reçues simultanément.</p> <pre>requete = str(connexion.recv(1024))</pre>	
<pre>Content = b'GET /?LED=OFF HTTP/1.1\r\nHost: 192.168.1.104\r\nUser-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:68.0) Gecko/20100101 Firefox/68.0\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\nAccept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3\r\nAccept-Encoding: gzip, deflate\r\nReferer: http://192.168.1.104/?LED=ON\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n'</pre> <p>led_on = requete.find('/?LED=ON') #/ ?LED=ON est présent à l'indice 338 (rouge)</p> <p>led_off = requete.find('/?LED=OFF') #/ ?LED=OFF est présent à l'indice 6 (vert)</p> <p>En fonction du numéro d'indice on allume ou éteint la LED (ici : LED OFF, indice 6)</p>	
<p>La variable <code>reponse</code> contient le texte HTML renvoyé par la fonction <code>web_page ()</code>. Terminer par l'envoi des données vers le client en respectant le protocole http.</p> <pre> reponse = web_page() connexion.send('HTTP/1.1 200 OK\n') connexion.send('Content-Type: text/html\n') connexion.send('Connection: close\n\n') connexion.sendall(reponse) connexion.close() </pre>	

Annexe 2 : 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";
```

```

pi@raspberrypi: ~
File Edit Tabs Help
GNU nano 2.7.4 File: /etc/sudoers.tmp Modified
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:$
Defaults        env_keep+="http_proxy https_proxy no_proxy"

# Host alias specification

# User alias specification

# Cmnd alias specification

^G Get Help  ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line

```