

Évaluation Bloc 3 – DIU-EIL

Auteur : Johann Dolivet (Groupe 1)

Thèmes :

- 3.4 : Systèmes embarqués
- 3.5 : Réseaux

Comptage en binaire

Cette séance est fortement inspirée de l'atelier sur les systèmes embarqués (Bloc 3.4) réalisé en compagnie de Yann PAILLOT, Raphaël CHAY et Ivan MAUGIS sous la direction de Xavier REDON.

Description

L'objectif de cette séance sera de travailler sur un programme (en Python) permettant d'obtenir la représentation binaire d'un nombre entier positif donné dans sa représentation décimale.

Ce programme ne sera pas l'objet de cette séance (il aura été fait dans un chapitre précédent ou sera donné en exercice avant de débiter la séance).

À partir de ce programme, l'objectif sera d'effectuer un montage (commandé par un Raspberry Pi) permettant d'allumer des LEDs qui donneront ainsi la représentation binaire d'un nombre entier positif donné en paramètre d'une fonction.

Par la suite, nous commanderons ces allumages à distance en se plaçant sur le réseau local et en faisant jouer le rôle de serveur au Raspberry Pi.

Cette séance se situe à la suite du cours sur le réseau et permettra donc de réinvestir les notions du chapitre tout en abordant la partie systèmes embarqués.

Prérequis

- Écriture d'un entier positif dans une base $b \geq 2$
- Interaction client-serveur

Remarques

- Le choix sera fait de représenter l'entier non signé sur un demi-octet. Cela permet d'éviter de faire un montage trop complexe tout en restant suffisamment expressif.
- On utilise un Raspberry PI car cela permet d'utiliser la partie contrôleur du nano-ordinateur et de programmer en Python (les bibliothèques nécessaires sont déjà installées, par défaut). On profite, également, de la possibilité d'utiliser un SHELL et quelques fonctionnalités réseaux.

Liens programme

- Représentation des données
 - Écriture d'un entier positif dans une base $b \geq 2$
- Langages et programmation
 - Utilisation de bibliothèques
- Architectures matérielles et systèmes d'exploitation
 - Périphériques d'entrée et de sortie (Identifier le rôle des capteurs et actionneurs).
 - Systèmes d'exploitation (Utiliser les commandes de base en ligne de commande).
 - Transmission de données dans un réseau (Simuler ou mettre en œuvre un réseau).
 - Interface Homme-Machine (Réaliser par programmation une IHM répondant à un cahier des charges donné).

I. Préliminaires

On part d'un script python permettant de convertir un entier non signé en un nombre binaire de taille fixe (ici sur 4 bits).

Exemple :

- Script :

```
def conversion(entier):  
    """ entier est un nombre entier positif.  
    Retourne l'écriture de entier en binaire sur un demi octet.  
    Le résultat est une liste de 4 bits.  
    1 bit est un entier valant 0 ou 1.  
    Le 1e est le bit de poids faible.  
    """  
    nibble = []  
    for i in range(4):  
        bit = entier % 2  
        nibble.append(bit)  
        entier = entier // 2  
    return nibble
```

fichier : entToBin.py

- Exécution :

```
>>> conversion(5)  
[1, 0, 1, 0]  
>>> conversion(16)  
[0, 0, 0, 0]  
>>> conversion(251)  
[1, 1, 0, 1]
```

Remarques :

La spécification de la fonction `conversion()` de la fiche élève autorise différentes interprétations.

- Ainsi le retour de fonction ne sera pas nécessairement une liste (on peut espérer un tuple, une chaîne de caractères, ...).
- De la même manière, le choix pourra être fait de retourner une représentation avec le 1^{er} bit qui serait le bit de poids faible ou, au contraire, le bit de poids fort.

Les parties suivantes devront donc tenir compte du choix fait par les élèves pour la valeur de retour de leur fonction. Ce qui nécessitera donc, probablement quelques adaptations.

En ce qui nous concerne, le choix a été fait de retourner une liste, avec le 1^{er} bit qui est le bit de poids faible

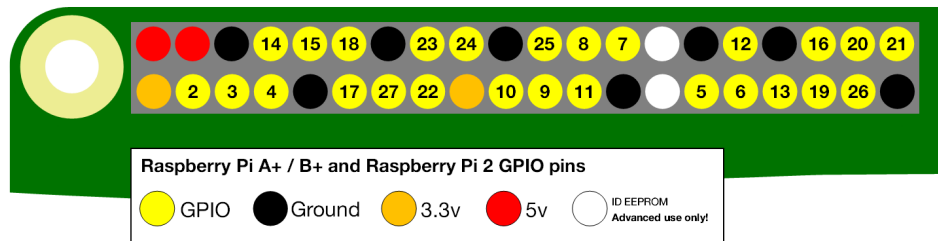
II. Allumage des LEDs

1. Montage

On travaille sur Raspberry Pi et on utilise le script Python pour commander un montage utilisant les broches du Raspberry et permettant de visualiser l'écriture binaire d'un entier en éclairant une suite de LEDs sur un montage.

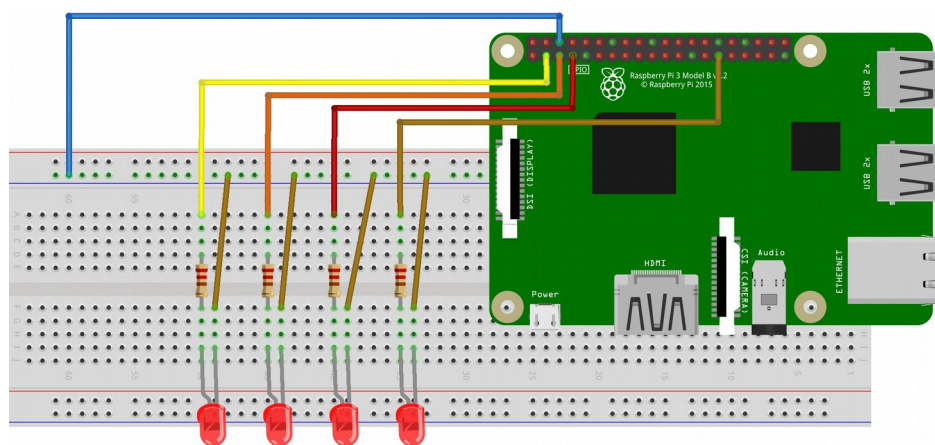
Il faut tout d'abord se familiariser avec le port d'Entrées-Sorties de Raspberry Pi (nous utilisons le modèle RPi3).

Une rapide recherche permet d'obtenir un schéma explicatif sur les ports.

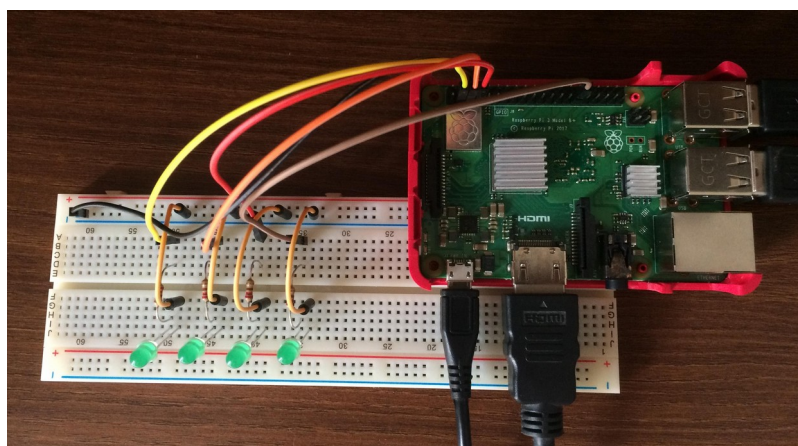


lien : <https://www.raspberrypi.org/documentation/usage/gpio/>

Nous pouvons donc commencer à réaliser le montage avec les LEDs et résistances :



fritzing
fichier : ComptageBinaire.fzz



2. Programmation

Il faut modifier et adapter la fonction `conversion()` afin que celle-ci déclenche l'allumage des LEDS. On utilise la bibliothèque `Rpi.GPIO`.

Liens utiles :

- <https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/>
- <https://deussyss.developpez.com/tutoriels/RaspberryPi/PythonEtLeGpio/>

Exemple :

- Script :

```
import RPi.GPIO as GPIO

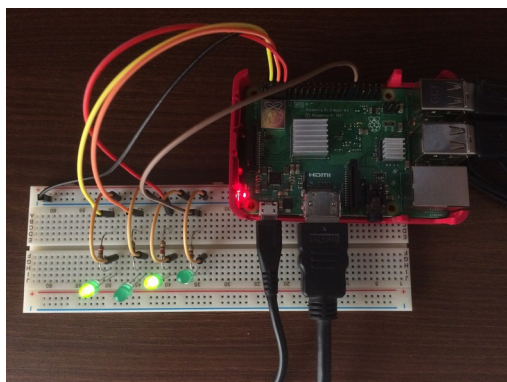
GPIO.setmode(GPIO.BCM) #On choisit le type de numérotation des broches
#Les broches sont des sorties
listeBroches = [2, 3, 4, 5]
for broche in listeBroches:
    GPIO.setup(broche, GPIO.OUT)

def conversionLEDs(entier):
    """ entier est un nombre entier positif.
    Retourne l'écriture de entier en binaire sur un demi octet.
    Le résultat est une liste de 4 bits.
    1 bit est un entier valant 0 ou 1.
    Le 1e est le bit de poids faible.
    Déclenche l'affichage des LEDs correspondantes.
    """
    # On convertit l'entier
    nibble = []
    for i in range(4):
        bit = entier % 2
        nibble.append(bit)
        entier = entier // 2
    # On allume les LEDs correspondantes
    B0, B1, B2, B3 = 2, 3, 4, 5 # On associe les bits aux broches
    listeLEDs = [B0, B1, B2, B3]
    for i in range(4):
        if nibble[i] == 1:
            GPIO.output(listeLEDs[i], GPIO.HIGH)
        else:
            GPIO.output(listeLEDs[i], GPIO.LOW)
    return nibble
```

fichier : entToLeds.py

- Exécution :

```
>>> conversion(5)
[1, 0, 1, 0]
```



III. Client-serveur

1. Essai en local

Avant de passer à la programmation sur le serveur, on peut s'exercer sur un ordinateur et faire des tests sur le localhost. On utilise la bibliothèque flask :

Liens utiles :

- <https://flask.palletsprojects.com/>
- <http://sdz.tdct.org/sdz/creez-os-applications-web-avec-flask.html>

Exemple :

- Script :

```
from flask import Flask

app = Flask(__name__) # On instancie notre objet Flask (une application)

# On définit le chemin principal
@app.route('/')
# Le comportement souhaité lorsqu'on y accède
def index():
    return "Salut! Tu es au bon endroit!"

# On définit le chemin au travers duquel nous pourrions lancer notre convertisseur
@app.route('/binaire/<int:entier>')
# Le paramètre de conversion() correspond au nombre passé dans l'URL
def conversion(entier):
    """ entier est un nombre entier positif.
    Retourne l'écriture de entier en binaire sur un demi octet.
    Le résultat est une liste de 4 bits.
    1 bit est un entier valant 0 ou 1.
    Le 1e est le bit de poids faible.
    """
    nibble = []
    entree = entier
    for i in range(4):
        bit = entier % 2
        nibble.append(bit)
        entier = entier // 2
    return "La représentation binaire de " + str(entree) + \
        " est " + str(nibble)

# On lance notre application
if __name__ == '__main__':
    app.run(host = '0.0.0.0', debug = True)
```

fichier : entToBinServeur.py

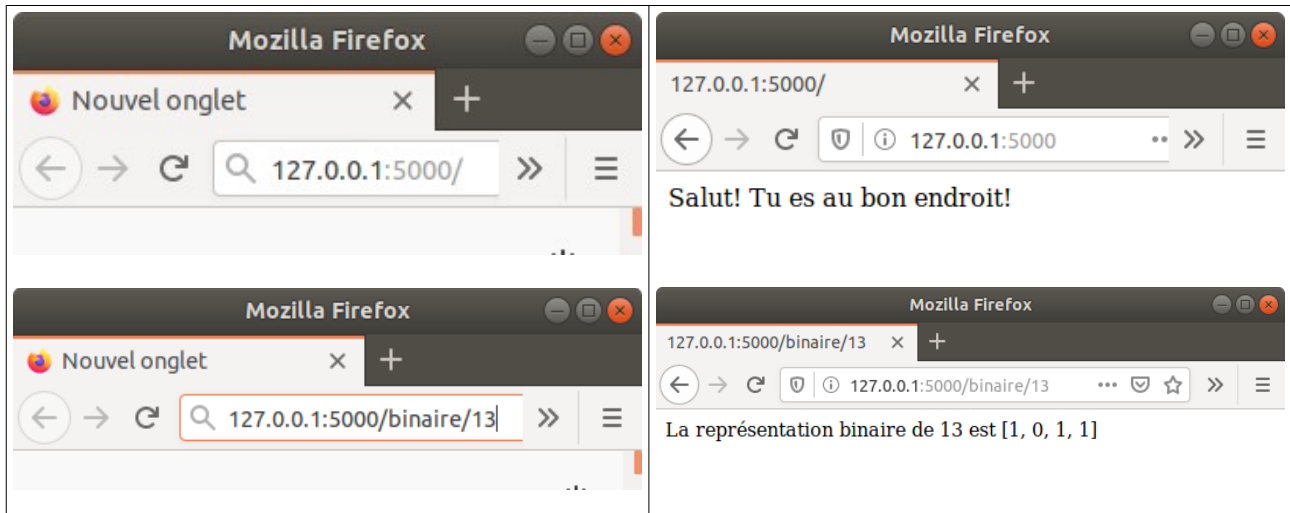
- Exécution :

```
>>> %Run entToBinServeur.py
32* Serving Flask app "entToBinServeur" (lazy loading)
* Environment: production
[31m WARNING: This is a development server. Do not use it in a production deployment.[0m
[2m Use a production WSGI server instead.[0m
* Debug mode: on
```

Remarques :

- Par défaut le serveur est accessible via le port 5000.
- Différentes méthodes peuvent être utilisées : les élèves peuvent utiliser les options de la bibliothèque flask, ils peuvent également utiliser une requête paramétrée pour passer la valeur d'une variable au travers de l'URL, ...

On utilise alors un navigateur pour communiquer avec le serveur localement.



2. Configuration du réseau

Nous pouvons alors étudier le réseau local.

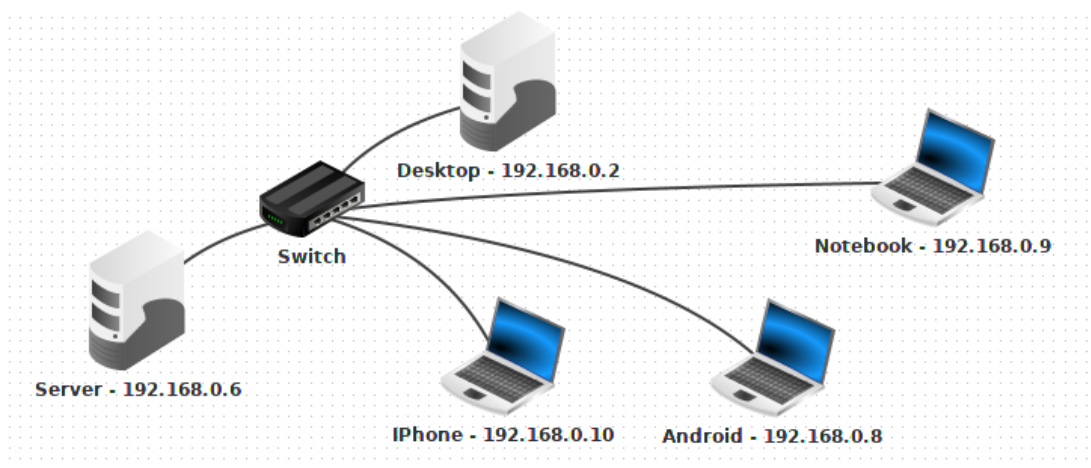
L'objectif principal étant d'obtenir l'IP locale du serveur (le Raspberry Pi).

Exemple :

```
pi@raspberrypi:~ $ ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN group default qlen 1000
   link/ether b8:27:eb:91:09:70 brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
   link/ether b8:27:eb:c4:5c:25 brd ff:ff:ff:ff:ff:ff
   inet 192.168.0.6/24 brd 192.168.0.255 scope global noprefixroute wlan0
       valid_lft forever preferred_lft forever
   inet6 2804:14c:60:8657:5790:49b7:9b1b:a2b1/128 scope global dynamic noprefixroute
       valid_lft 3205sec preferred_lft 3205sec
   inet6 2804:14c:60:8657:1763:905a:c6d1:f72f/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 3599sec preferred_lft 3599sec
   inet6 fe80::39a7:a373:1476:b330/64 scope link
       valid_lft forever preferred_lft forever
```

On vérifie, dans cet exemple, que l'adresse du serveur est 192.168.0.6

Schéma du réseau local :



fichier : ReseauLocal.flvs

3. Configuration du serveur

On utilise maintenant les scripts précédents afin d'obtenir le script du programme qui sera exécuté sur le serveur

Exemple :

- Script :

```
from flask import Flask
import RPi.GPIO as GPIO

# On paramètre les broches
GPIO.setmode(GPIO.BCM)
listeBroches = [2, 3, 4, 5]
for broche in listeBroches:
    GPIO.setup(broche, GPIO.OUT)

# On instancie notre objet Flask (une application)
app = Flask(__name__)

# On définit le chemin principal
@app.route('/')
# Le comportement souhaité lorsqu'on y accède
def index():
    return "Salut! Tu es au bon endroit!"

# On définit le chemin au travers duquel nous pourrions lancer notre convertisseur
@app.route('/binaire/<int:entier>')
# Le paramètre de conversion() correspond au nombre passé dans l'URL
def conversionLEDs(entier):
    """ entier est un nombre entier positif.
    Retourne l'écriture de entier en binaire sur un demi octet.
    Le résultat est une liste de 4 bits.
    1 bit est un entier valant 0 ou 1.
    Le 1e est le bit de poids faible.
    Déclenche l'affichage des LEDs correspondantes.
    """
    entree = entier
    # On convertit l'entier
    nibble = []
    for i in range(4):
        bit = entier % 2
        nibble.append(bit)
        entier = entier // 2
    # On allume les LEDs correspondantes
    B0, B1, B2, B3 = 2, 3, 4, 5 # On associe les bits aux broches
    listeLEDs = [B0, B1, B2, B3]
    for i in range(4):
        if nibble[i] == 1:
            GPIO.output(listeLEDs[i], GPIO.HIGH)
        else:
            GPIO.output(listeLEDs[i], GPIO.LOW)
    return "La représentation binaire de " + str(entree) + " est " + str(nibble)

# On lance notre application
if __name__ == '__main__':
    app.run(host = '0.0.0.0')
```

fichier : entToLedsServeur.py

- Exécution :

```
>>> %Run entToLedsServeur.py

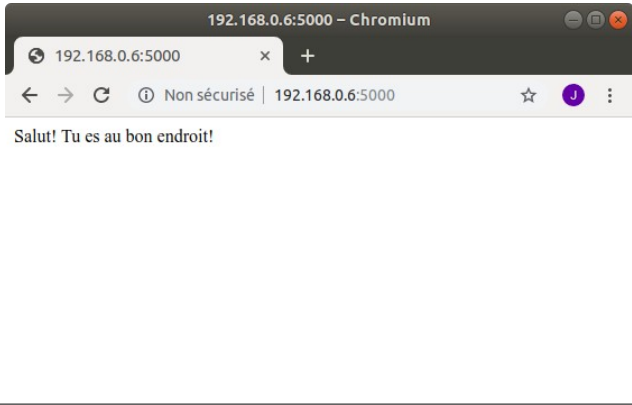
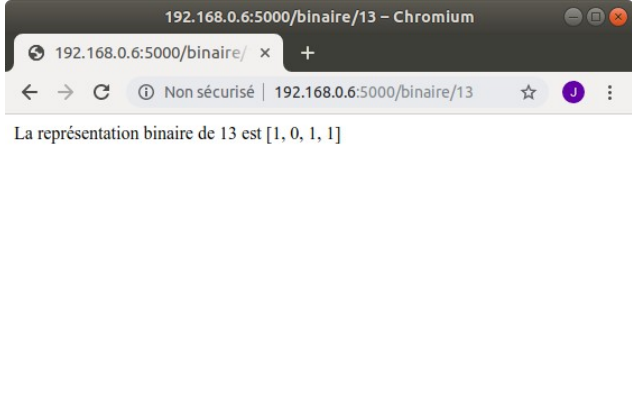
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

On peut désormais communiquer avec le serveur.

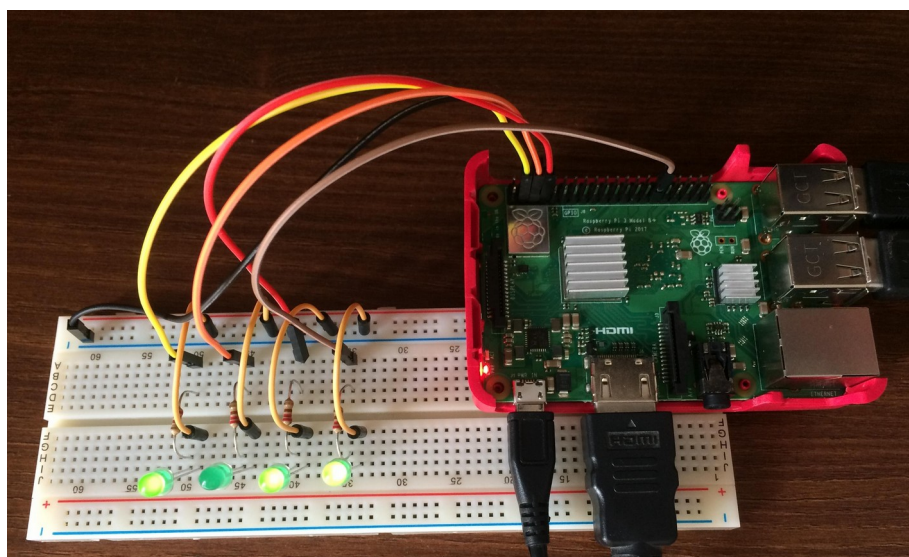
Exemples de clients :

- Notebook (connexion par le WIFI)

Adresse locale : 192.168.0.9

Client	Serveur
	<pre>>>> %Run entToLedsServeur.py * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit) 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET / HTTP/1.1" 200 - 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET /favicon.ico HTTP/1.1" 404 -</pre>
	<pre>>>> %Run entToLedsServeur.py * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit) 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET / HTTP/1.1" 200 - 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.9 - - [07/Nov/2019 10:47:01] "GET /binaire/13 HTTP/1.1" 200 -</pre>

On vérifie bien que l'allumage se déclenche !

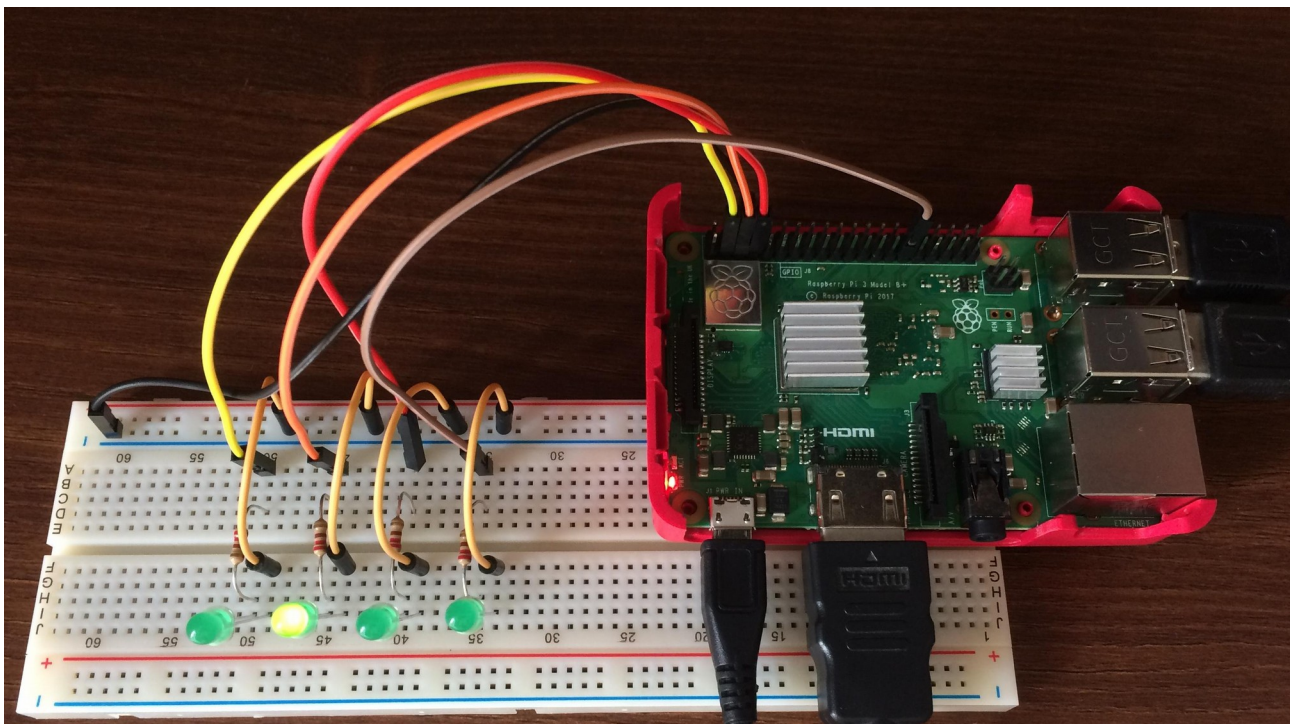


- Iphone (connexion par le WIFI)

Adresse locale : 192.168.0.10

Client	Serveur
<div>192.168.0.6</div> <p>Salut! Tu es au bon endroit!</p>	<pre>>>> %Run entToLedsServeur.py * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit) 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET / HTTP/1.1" 200 - 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.9 - - [07/Nov/2019 10:47:01] "GET /binaire/13 HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET / HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET /favicon.ico HTTP/1.1" 404 -</pre>
<div>192.168.0.6</div> <p>La représentation binaire de 258 est [0, 1, 0, 0]</p>	<pre>>>> %Run entToLedsServeur.py * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit) 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET / HTTP/1.1" 200 - 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.9 - - [07/Nov/2019 10:47:01] "GET /binaire/13 HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET / HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.10 - - [07/Nov/2019 10:51:34] "GET /binaire/258 HTTP/1.1" 200 -</pre>

On vérifie bien que l'allumage se déclenche !

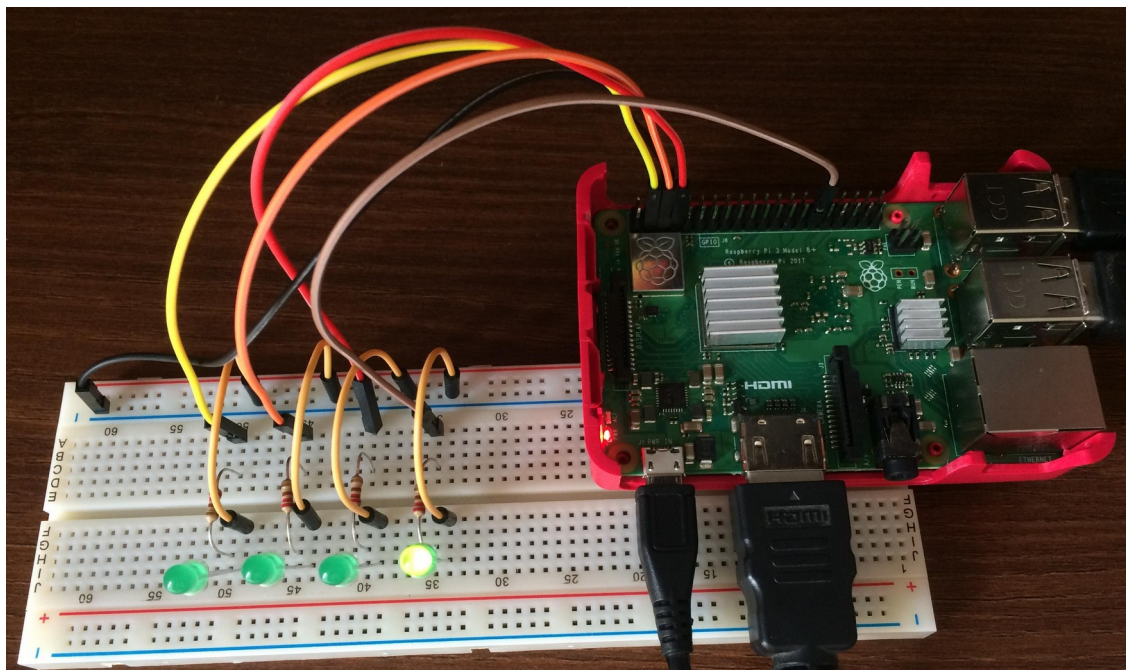


- Android (connexion par le WIFI)

Adresse locale : 192.168.0.8

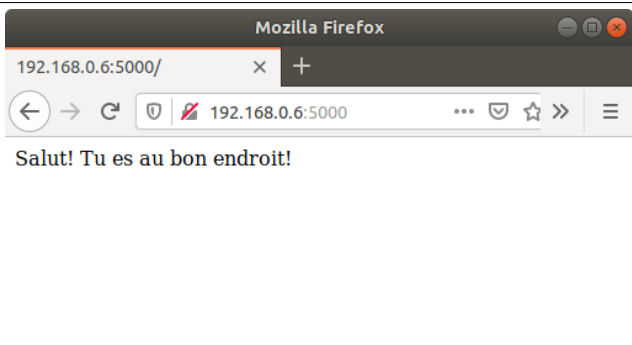
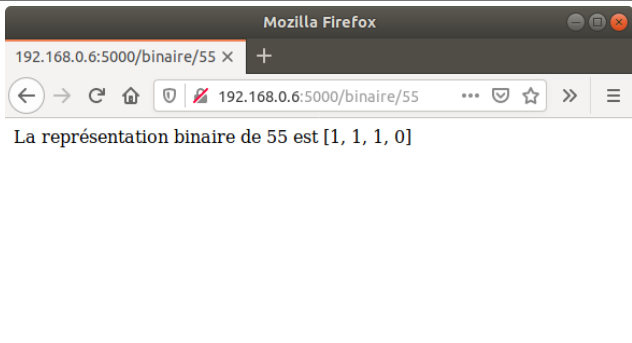
Client	Serveur
<p>192.168.0.6:5000 192.168.0.6:5000</p> <p>Salut! Tu es au bon endroit!</p>	<pre>>>> %Run entToLedsServeur.py * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit) 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET / HTTP/1.1" 200 - 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.9 - - [07/Nov/2019 10:47:01] "GET /binaire/13 HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET / HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.10 - - [07/Nov/2019 10:51:34] "GET /binaire/258 HTTP/1.1" 200 - 192.168.0.8 - - [07/Nov/2019 10:53:19] "GET / HTTP/1.1" 200 -</pre>
<p>192.168.0.6:5000/binaire/24 192.168.0.6:5000</p> <p>La représentation binaire de 24 est [0, 0, 0, 1]</p>	<pre>>>> %Run entToLedsServeur.py * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit) 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET / HTTP/1.1" 200 - 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.9 - - [07/Nov/2019 10:47:01] "GET /binaire/13 HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET / HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.10 - - [07/Nov/2019 10:51:34] "GET /binaire/258 HTTP/1.1" 200 - 192.168.0.8 - - [07/Nov/2019 10:53:19] "GET / HTTP/1.1" 200 - 192.168.0.8 - - [07/Nov/2019 10:53:21] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.8 - - [07/Nov/2019 10:56:20] "GET /binaire/24 HTTP/1.1" 200 -</pre>

On vérifie bien que l'allumage se déclenche !

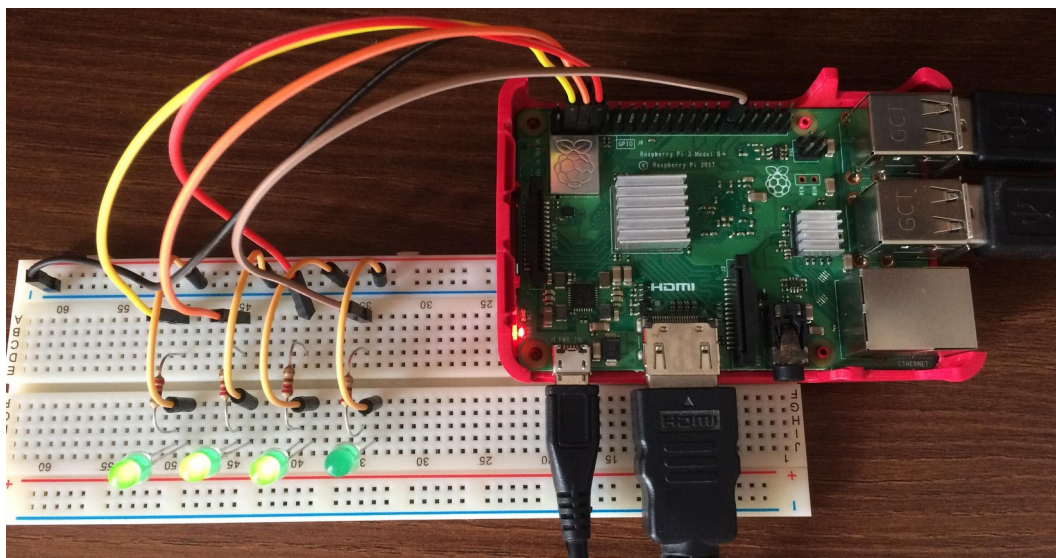


- Ordinateur (connexion filaire)

Adresse locale : 192.168.0.2

Client	Serveur
	<pre>>>> %Run entToLedsServeur.py * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit) 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET / HTTP/1.1" 200 - 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.9 - - [07/Nov/2019 10:47:01] "GET /binaire/13 HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET / HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.10 - - [07/Nov/2019 10:51:34] "GET /binaire/258 HTTP/1.1" 200 - 192.168.0.8 - - [07/Nov/2019 10:53:19] "GET / HTTP/1.1" 200 - 192.168.0.8 - - [07/Nov/2019 10:53:21] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.8 - - [07/Nov/2019 10:56:20] "GET /binaire/24 HTTP/1.1" 200 - 192.168.0.2 - - [07/Nov/2019 10:59:18] "GET / HTTP/1.1" 200 -</pre>
	<pre>>>> %Run entToLedsServeur.py * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit) 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET / HTTP/1.1" 200 - 192.168.0.9 - - [07/Nov/2019 10:44:42] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.9 - - [07/Nov/2019 10:47:01] "GET /binaire/13 HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET / HTTP/1.1" 200 - 192.168.0.10 - - [07/Nov/2019 10:49:56] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.10 - - [07/Nov/2019 10:51:34] "GET /binaire/258 HTTP/1.1" 200 - 192.168.0.8 - - [07/Nov/2019 10:53:19] "GET / HTTP/1.1" 200 - 192.168.0.8 - - [07/Nov/2019 10:53:21] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.8 - - [07/Nov/2019 10:56:20] "GET /binaire/24 HTTP/1.1" 200 - 192.168.0.2 - - [07/Nov/2019 10:59:18] "GET / HTTP/1.1" 200 - 192.168.0.2 - - [07/Nov/2019 10:59:18] "GET /favicon.ico HTTP/1.1" 404 - 192.168.0.2 - - [07/Nov/2019 11:00:11] "GET /binaire/55 HTTP/1.1" 200 -</pre>

On vérifie bien que l'allumage se déclenche !



IV. Prolongements

- On peut effectuer le montage des LEDs et le programmer en utilisant un micro-contrôleur (Arduino, par exemple). Cela permettrait d'aborder la notion sur la diversité des langages.
- On peut travailler sur la mise en forme du site (balisage HTML, CSS).
- Les requêtes peuvent se faire via POST et GET