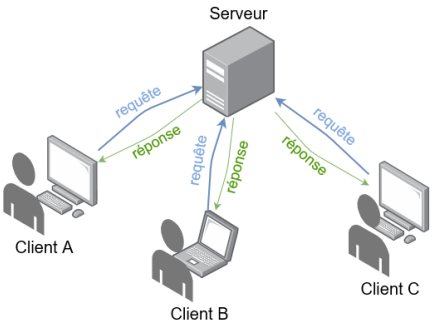
Activité 9 : Interaction Homme – machine sur le web
Et si vous tchattiez !!

1- RAPPEL

1.1 Architecture Client – Serveur :

Dans l'**architecture Client-Serveur**, on trouve en général un serveur et plusieurs clients. Le **serveur** est une machine qui traite les *requêtes* du **client** et lui envoie éventuellement une *réponse*.



Il y a donc deux types d'application installés sur les machines :

- l'application « **serveur** » : *écoute* en attendant des connexions des clients ;
- les applications **« client** » : se *connectent* au serveur.

1.2 Etablissement de la connexion :

Pour que le *client* se connecte au *serveur*, il lui faut deux informations :

- L'**adresse IP** du serveur, ou son **nom d'hôte** (*host name*), qui identifie une machine sur Internet ou sur un réseau local.
- Le **numéro de port** associé à l'application qui doit gérer les requêtes sur le serveur.

Pour que le *serveur* réponde au *client*, il lui faut également deux informations :

- Le **nom d'hôte** (*host name*), du client ;
- Le **numéro de port** associé à l'application qui doit recevoir les réponses sur le client.

2- Réseau simple en python

2.1 Les sockets:

En anglais un **socket** est un "trou" qui laisse passer des choses.
Les **sockets** sont des objets logiciels qui permettent d'ouvrir une connexion avec une machine locale ou distante et d'échanger avec elle.
Ils peuvent mettre en oeuvre deux techniques de communication différentes et complémentaires : celle des paquets (datagrammes), très largement utilisée sur l'internet, et celle de la connexion continue, ou **stream socket**, qui est un peu plus simple.

2.2 Application serveur

Création d'un objet « socket » :

```
socket_ecoute = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Le constructeur de l'objet socket attend deux paramètres :

- la **famille d'adresses** : *socket.AF_INET* = adresses Internet de type IPv4 ;
- le **type du socket** : *socket.SOCK_STREAM* = protocole TCP.

Le script ci-dessous met en place un serveur capable de communiquer avec un seul client.

```
import socket

s = socket.socket()
s.bind(('', 50007))
s.listen(1)
client,laddr = s.accept()

response = client.recv(255).decode('utf8')
while response != 'z':
    print(response)
    response = client.recv(255).decode('utf8')
print ("Close")
#s.close()
client.close()
```

Le *socket* doit à présent être lié à un **port** de la machine avec la méthode .*blind.*
Puis le *socket d'écoute* est mis à l'état d'écoute (*LISTEN*) :

Analyse : Lorsqu'on fait appel à sa méthode **accept()**, le socket attend indéfiniment qu'une requête se présente. Le script est donc interrompu à cet endroit, un peu comme il le serait si nous faisions appel à une fonction **input()** pour attendre une entrée clavier. Si une requête est réceptionnée, la méthode **accept()** renvoie un tuple de deux éléments : le premier est la référence d'un nouvel objet de la classe **socket()**[74], qui sera la véritable interface de communication entre le client et le serveur, et le second un autre tuple contenant les coordonnées de ce client (son adresse IP et le n° de port qu'il utilise lui-même).

2.3 Coté client :

```
import socket
s = socket.socket()
s.connect(('localhost',50007))
print ("Connection on {}".format(50007))
message=input(' ?')

while message!='z':
    s.send(bytes(message,'utf8'))
    message=input('?')
s.send(b'z')
s.close()
```

Les méthodes send() et recv() du socket servent évidemment à l'émission et à la réception des messages, qui doivent être de simples chaînes de caractères. Remarques : la méthode send() renvoie le nombre d'octets expédiés. L'appel de la méthode recv () doit comporter un argument entier indiquant le nombre maximum d'octets à réceptionner en une fois (Les octets surnuméraires sont mis en attente dans un tampon. Ils sont transmis lorsque la même méthode recv() est appelée à nouveau).

2.4 Exécution en local/

Le premier script à lancer sera -comme son nom l'indique- sur le serveur et écoutera les demandes des clients.
Le script client sera donc exécuté sur la machine cliente, c'est lui qui fera la demande du service du serveur distant.

Maintenant , montons d'un cran :
Rechercher les adresses IP des PC de la salle.
Etablir une connexion entre 2 PC

3- Script plus élaboré

COTE SERVEUR

```
1. # Définition d'un serveur réseau rudimentaire
2. # Ce serveur attend la connexion d'un client, pour entamer un dialogue avec lui
3.
4. import socket, sys
5.
```

```
6. HOST = 'à déterminer'
7. PORT = 50000
8.
9. # 1) création du socket :
10.mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11.
12.# 2) liaison du socket à une adresse précise :
13.try:
14.    mySocket.bind((HOST, PORT))
15.except socket.error:
16.    print "La liaison du socket à l'adresse choisie a échoué."
17.    sys.exit()
18.
19.while 1:
20.    # 3) Attente de la requête de connexion d'un client :
21.    print "Serveur prêt, en attente de requêtes ..."
22.    mySocket.listen(5)
23.
24.    # 4) Etablissement de la connexion :
25.    connexion, adresse = mySocket.accept()
26.    print ("Client connecté, adresse IP %s, port %s" % (adresse[0], adresse[1]))
27.
28.    # 5) Dialogue avec le client :
29.    connexion.send("Vous êtes connecté au serveur Marcel. Envoyez vos messages.")
30.    msgClient = connexion.recv(1024)
31.    while 1:
32.        print ("C>", msgClient)
33.        if msgClient.upper() == "FIN" or msgClient =="":
34.            break
35.        msgServeur = input("S> ")
36.        connexion.send(msgServeur)
37.        msgClient = connexion.recv(1024)
38.
39.    # 6) Fermeture de la connexion :
40.    connexion.send("Au revoir !")
41.    print ("Connexion interrompue.")
42.    connexion.close()
43.
44.    ch = input("<R>ecommencer <T>erminer ? ")
45.    if ch.upper() =='T':
46.        break
```

COTE CLIENT :

```
# Définition d'un client réseau rudimentaire
# Ce client dialogue avec un serveur ad hoc

import socket, sys

HOST = '192.168.0.23'
PORT = 50000

# 1) création du socket :
mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 2) envoi d'une requête de connexion au serveur :
try:
    mySocket.connect((HOST, PORT))
except socket.error:
    print ("La connexion a échoué.")
    sys.exit()
print ("Connexion établie avec le serveur.")

# 3) Dialogue avec le serveur :
msgServeur = mySocket.recv(1024)

while 1:
    if msgServeur.upper() == "FIN" or msgServeur =="":
        break
    print ("S>", msgServeur)
    msgClient =input("C> ")
    mySocket.send(bytes(msgClient,'utf8'))
    msgServeur = mySocket.recv(1024)

# 4) Fermeture de la connexion :
print ("Connexion interrompue.")
mySocket.close()
```