

Les sockets réseaux en Python

Un socket est un point de communication entre deux programmes. On peut le voir comme une prise réseau : un programme attend sur une adresse IP et un port, et un autre s'y connecte pour échanger des données.

Voici un exemple simple en TCP : le client envoie un message et le serveur lui renvoie le même message. TCP établit une connexion entre les deux programmes. En Python, le serveur crée une socket, l'associe à une adresse avec `bind()`, attend une connexion avec `listen()` et `accept()` ; le client se connecte avec `connect()`.

Pour le serveur

fileserveur.py

```
import socket

HOST = "192.168.xxx.xxx" # Insérer la bonne @IP ==> 127.0.0.1 pour le
même PC
PORT = 50007

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as serveur:
    serveur.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    serveur.bind((HOST, PORT))
    serveur.listen()

    print(f"Serveur actif sur {HOST}:{PORT} (Ctrl+C pour arrêter)")

    while True: # accepte les connexions indéfiniment
        connexion, adresse = serveur.accept()
        print("Client connecté :", adresse)

        with connexion:
            # Utilise une ligne terminée par Entrée comme délimiteur de
            message
            with connexion.makefile("rwb") as flux:
                for ligne in flux:
                    message = ligne.decode("utf-8").rstrip("\n")
                    print("Message reçu :", message)

                    if message.lower() == "quit":
                        flux.write("Connexion
fermée\n".encode("utf-8"))
                        flux.flush()
                        break
```

```
        flux.write(f"Serveur reçu :  
{message}\n".encode("utf-8"))  
        flux.flush()  
  
    print("Client déconnecté")
```

pour le client

fileclient.py

```
import socket  
  
HOST = "127.0.0.1"  
PORT = 50007  
  
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:  
    client.connect((HOST, PORT))  
  
    with client.makefile("rwb") as flux:  
        while True:  
            message = input("Votre message (quit pour quitter) : ")  
            flux.write((message + "\n").encode("utf-8"))  
            flux.flush()  
  
            reponse = flux.readline()  
            if not reponse:  
                print("Le serveur a fermé la connexion.")  
                break  
  
            print(reponse.decode("utf-8").rstrip())  
  
            if message.lower() == "quit":  
                break
```

Lance d'abord le serveur dans un terminal avec `python3 serveur.py`, puis le client dans un autre avec `python3 client.py`. Le client se connecte à 127.0.0.1 — la machine locale — sur le port 50007. `sendall()` envoie les données, et `recv(1024)` lit jusqu'à 1024 octets. Les données échangées sont des octets : ici, `.encode()` et `.decode()` servent à convertir entre texte et octets.

Dans cet exemple, `AF_INET` indique IPv4 et `SOCK_STREAM` indique TCP. Une socket TCP est un flux d'octets : pour des échanges plus complexes, il faut définir un protocole pour savoir où chaque message commence et se termine.

Pour Serveur permanent, il faut placer `accept()` dans une boucle infinie. Ainsi, le serveur continue

d'attendre des connexions, y compris après le départ d'un client. Voici une version qui garde aussi la connexion avec un client ouverte pour échanger plusieurs messages.

Lance `python3 fileserveur.py` dans un terminal, puis `python3 fileclient.py` dans un autre terminal ou sur un autre pc . Le serveur continue de tourner après qu'un client a tapé "quit" et peut accepter un nouveau client. Avec cette version, il traite les clients l'un après l'autre : il attend que le client courant se déconnecte avant d'accepter le suivant. Arrête le serveur avec "Ctrl+C".

From:

<https://www.magenealogie.chanterie37.fr/www/fablab37110/> - Castel'Lab le Fablab MJC de Château-Renault

Permanent link:

https://www.magenealogie.chanterie37.fr/www/fablab37110/doku.php?id=debuter_en_python:socket&rev=1791908564

Last update: **2026/10/13 18:22**

