Réalisation d'un « chat ».

Pour réaliser votre fenêtre de chat il faudra :

Réaliser une interface fonctionnelle (à améliorer graphiquement par la suite)

Zone pour afficher les messages
Zone pour saisir le message
Bouton pour envoyer le message, sauvegarder en locale la conversation et sortir
pour défiler le texte(bibliothèque :import scrolledtext)
#logo et autres apports personnels.
Tester vos scripts (les deux fenêtres) Ils doivent remplir les conditions d'un IHM.
Par exemple s'interroger sur ces points:

- Au contexte
- grand public (proposer une prise en main immédiate)
- loisirs (rendre le produit attrayant)
- Aux caractéristiques de la tâche
- usage occasionnel, régulier, quotidien, tâche répétitive
- Aux contraintes techniques
- plateforme
- *mémoire, bande passante*
- écran
- réutilisation de code ancien

Faire un compte rendu sur ce travail en expliquant les étapes que vous avez suivies pour produire votre IHM. Conseil: réalisez vos fenêtres clients en vous inspirant du tp précédent. et toute innovation n'est pas un luxe…

Partie fonctions:
Etape1 : Ecrire le serveur (il est possible de regarder la doc ici:
https://python.doctor/page-reseaux-sockets-python-port

Configuration du serveur
Créer une socket du serveur
```
    print(f"Serveur en écoute sur {host}:{port}")

    while True:
        # Accepter les connexions des clients
        # Ajouter le client à la liste
        # Démarrer un thread pour gérer la connexion du client
```
Testez le :
```
if __name__ == "__main__":
    # Lancer le serveur dans un thread séparé
    threading.Thread(target=start_server).start()
```

Etape 2 : configurer le socket

Importer les modules : socket et threading.

Déclarer self.host = host .Cette instruction attribue la valeur de « host » à l'attribut « host » de l'instance de la classe. En d'autres termes, elle stocke l'adresse IP passée en argument lors de la création de l'objet de votre classe dans l'attribut « host » de cet objet.

```
        self.port = port
        self.socket =« il faudra chercher comment le remplir »
self.socket.connect(….)
# Lancer le thread pour recevoir des messages
        threading.Thread
```

Etape 3 : a) écrire la fonction envoyer message

```
        # Récupérer le message de la zone de saisie
        # Envoyer le message au socket
        # Afficher le message dans la zone d'affichage
        # Effacer la zone de saisie
```

b) écrire la fonction recevoir un message

```
# Recevoir les messages du socket
# Si aucune donnée n'est reçue, le client est déconnecté
# Afficher le message dans la zone d'affichage
            self.affichae_du_message.insert(tk.END, f"Autre personne: {message}\n")

 # Mise à jour de l'interface Tkinter (sans bloquer la boucle principale)
 # Gérer la déconnexion du client
```

c) ecrire un fonction tartampion(client_socket, address): # Cette fonction est utilisée pour gérer la communication avec un client spécifique.

1. `def tartampion(client_socket):` : Définition de la fonction qui prend en paramètre la socket du client.
2. `while True:` : La boucle infinie qui maintient la communication avec le client.

2.1 Try ….

Essaie de recevoir des données depuis la socket du client avec une taille maximale de 1024 octets et stocke les données dans la variable **`data`**.

3. Si aucune donnée n'est reçue (la socket du client est fermée), la boucle **`while`** est interrompue, indiquant que le client s'est déconnecté.

4 .Parcourt la liste des clients et envoie les données reçues (**`data`**) à tous les autres clients sauf au client d'origine (**`client_socket`**). Cela assure que les messages sont diffusés à tous les clients connectés, sauf à l'émetteur du message.

5. En cas d'erreur de connexion (**`ConnectionResetError`**), la boucle **`while`** est interrompue, indiquant que le client s'est déconnecté

6. Une fois que la boucle **`while`** se termine, cela signifie que le client s'est déconnecté. La socket du client est retirée de la liste **`clients`**, et la socket du client est fermée.