Consignes. Avant toute chose, prenez conscience que les corrections sont effectuées à l'aide d'un automate, par conséquent toute violation des règles conduira inévitablement à la délivrance d'une note nulle.

- Comment livrer mon code ?
  Si mon nom de famille est **yunès** et que je souhaite rendre le travail pour l'exercice 12 du tp 3 alors je dois construire une archive au format ZIP de nom `yunes-tp3-ex12.zip`, le nom ayant été converti en minuscules, sans accent et sans espace (*ie* : **de Gaulle** doit être traduit en `degaulle`). Si vous avez un homonyme rajoutez des initiales, par exemple `yunesjb-tp1-ex1`. Respectez votre nommage pour tout le semestre.
- Que mettre dans l'archive ?
  Le code source (et uniquement le code source) correspondant à l'exercice.

De plus, suivez bien les instructions complémentaires des livrables.

Exercice 1

Créer un code Python permettant d'implémenter un serveur TCP attendant des connexions sur le port 6666. Pour chaque connexion, le serveur:

1. lira un message depuis la socket de service,
2. affichera le message à l'écran (avec les informations de connexion)
3. puis fermera la connexion cliente et se remettra en attente (sauf si le message commence par `quit` ce qui entraînera l'arrêt du serveur).

Note : on pourra tester ce serveur avec le client `telnet`.
Voici donc un exemple de session (vu du côté serveur) :

```
Waiting...
Received bonjour
 from ('127.0.0.1', 53131)
Waiting...
Received salut
 from ('127.0.0.1', 53135)
Waiting...
Received hi fellow
 from ('127.0.0.1', 53138)
Waiting...
Received quit
 from ('127.0.0.1', 53142)
Quitting
```

Livrable : l'archive contenant le code `server.py`.

Exercice 2

Créer un code Python implémentant un client capable de communiquer avec le serveur de l'exercice 1. Il enverra son premier argument en ligne de commande comme message par exemple : `./client.py coucou`.
Livrable : l'archive contenant `client.py`.

Exercice 3

Créer un couple client/serveur implémentant un service de type `echo` (mais sur le port `6666`). La communication entre client et serveur consiste en :

1. l'envoi d'une message par le client,
2. pour toute réception d'un message par le serveur en le renvoi vers le client de ce même message,
3. à reception par le client de son message, vérifier que le message envoyé et celui reçu sont identiques, afficher le diagnostic puis terminer la connexion.

Livrable : l'archive contenant le programme `client_echo.py`, `server_echo.py`.

Exercice 4

Implémenter un mini-serveur HTTP (concurrent d'Apache) à tester avec votre navigateur préféré! Consulter internet (ou l'enseignant) pour avoir des détails sur le protocole HTTP (mais vous pourrez le découvrir en procédant au développement du serveur par itération). Ce mini-serveur devrait permettre de *servir* au moins des fichiers HTML, CSS et JPEG. (Commencer par HTML «pur»).
Indication : le serveur HTTP reçoit des données (texte), s'arrête quand il a lu une ligne vide, interprète ce qu'il a lu puis répond en conséquence au client.
Livrable : l'archive contenant `sioux.py`.