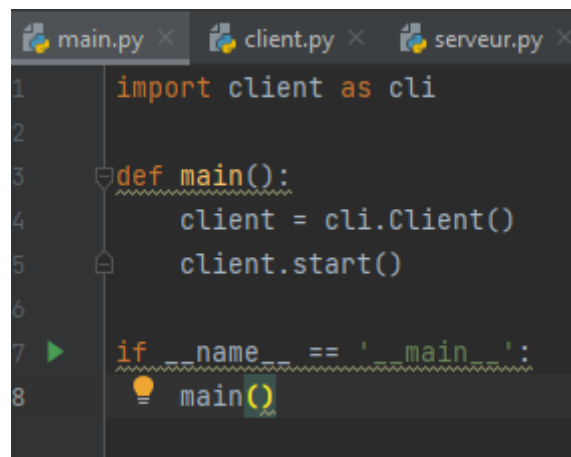


Programmer Documentation (Easy difficulty)

File main.py :

In this file you will find the base of my code, it is thanks to this piece of code that the client will launch.



```
1 import client as cli
2
3 def main():
4     client = cli.Client()
5     client.start()
6
7 if __name__ == '__main__':
8     main()
```

File client.py :

First of all, we import “socket,os,csv” for the good functioning of the code(here os is not used we see it because it is grayed)

Then we see that we define the Client class, then we create a dictionary with the information of the server which will be in a CSV file already created specifically for that.

Then I define “start”, in start I launch the client using the data from the dico which comes from the csv file then I send a message “The client is connected” with the IP address and the port used

```
main.py x client.py x serveur.py x fichier.csv x
1 import socket, os, csv
2
3 class Client:
4
5     def __init__(self):
6         self.__dico = {}
7         f = open('fichier.csv', 'r')
8         obj = csv.reader(f)
9         for lignes in obj:
10             self.__a = lignes[0]
11             self.__dico[lignes[0]] = lignes[1]
12
13
14     def start(self):
15         self.__client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16         self.__client_socket.connect((str(self.__a), int(self.__dico.get(self.__a))))
17         print(f"\nLe client vient de se connecter, Adresse IP : {str(self.__a)} / Port : {int(self.__dico.get(self.__a))}\n")
18         self.message()
19
20
```

I then create "message" which will be the source of the code allowing to take into account the user's message in order to answer his request. For example, if he types 2 he will know his local ip address. If he types 3 he will know the bone... If the user types 7 he will have to put a command that will be executed in command prompt

```
def message(self):
    # ce qui est reçu
    msg = ""
    # ce qui est envoyé
    data = ""
    while msg != "1" and data != "1" and msg != "8" and data != "8":
        msg = input("Voulez vous : ")
        "\n1. Fermer la connexion (Saisir quit avant de taper 1) : "
        "\n2. Avoir l'adresse IP locale : "
        "\n3. Connaître l'os : "
        "\n4. Obtenir l'usage du CPU : "
        "\n5. Avoir le nom du poste : "
        "\n6. Obtenir la quantité de RAM : "
        "\n7. Utiliser l'invite de commande : "
        "\nMerci de Saisir 'quit' si vous souhaitez fermer la connexion lié au serveur puis tapez 1 :")
    if msg == "1":
        print("Vous allez être déconnecté")
        # Ajouter un moyen de se déco
    elif msg == "7":
        message = input('Commande à exécuter : ')
        self.__client_socket.send(message.encode())
    else:
        self.__client_socket.send(msg.encode())
    data = self.__client_socket.recv(1024).decode()
    print(data)
    self.__client_socket.close()
```

File serveur.py:

It's the same as for client.py at the beginning, we import what is necessary, we define the server class then we put a port and an IP address (here 0.0.0.0 and 2047)

We define start which will allow to start the server, we do the same with the connection that is to say that once the server is launched it waits for a client to connect to it

```
import socket, os, sys, platform, psutil, subprocess

class Server:
    def __init__(self):
        self.__host, self.__port = ('0.0.0.0', 2047)
        self.__server_socket = None
        self.__conn = None
        self.__addr = None

    def start(self):
        print("Le serveur se lance")
        self.__server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.__server_socket.bind((self.__host, self.__port))
        self.connexion()

    def connexion(self):
        self.__server_socket.listen(1)
        print(f"Le serveur {self.__host} est en attente de connexion")
        self.__conn, self.__addr = self.__server_socket.accept()
        self.message()
```

I define here the answer to the messages that will come from the client, for example when the person will type 2, the message returned will be his local ip address, if I type 3 I will receive the os and the version of it. If I type 4 I will have the percentage of cpu usage. If I type 5 I'll get the name of the pc, if I type 6 I'll receive the amount of ram. In each "if/elif" the message is define as the thing we will see on the client.

```
def message(self):
    while True:
        msg = ""
        data = ""

        while msg != "1" and data != "1" and msg != "7":
            data = self.__conn.recv(1024).decode()
            print(data)
            if data == "2":
                message = f"\nVoici l'adresse IP du pc : {socket.gethostbyname(socket.gethostname())}\n"
                self.__conn.send(message.encode())
            elif data == "3":
                message = f"\nNom de l'os : {platform.system()} | Version du système : {platform.release()}\n"
                self.__conn.send(message.encode())
            elif data == "4":
                message = f"\nL'usage du CPU est : {psutil.cpu_percent(2)}%\n"
                self.__conn.send(message.encode())
            elif data == "5":
                message = f"\nLe nom du pc est : {socket.gethostname()} \n"
                self.__conn.send(message.encode())
            elif data == "6":
                vm = round(psutil.virtual_memory()[3] / 1000000000, 2)
                message = f"\nMémoire RAM utilisé: {psutil.virtual_memory()[2]} % | RAM Utilisé (GB): {vm}\n"
                self.__conn.send(message.encode())
```

We can see that if the user types "quit" the server will shut down. Then I put an "else" which will answer all the commands that the user will enter manually like for example "ping google.com". The result will be directly displayed and sent back to the client. If the command does not work or an error appears, a message indicating this will be displayed in the client console

```
elif data== "quit":
    message = "\nLe serveur est désormais déconnecté. Afin de déconnecter le client merci de saisir : 1\n"
    self.__conn.send(message.encode())
    self.__conn.close()
    self.__server_socket.close()

else:
    commande = data.split(' ')
    try:
        self.__process = subprocess.Popen(commande, stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                                           text=True, shell=True, encoding="cp850")
        output, error = self.__process.communicate()

    except:
        error = 'Ca ne marche pas'
        output = 'Ca ne marche pas'

    if error == '':
        if output == '':
            message = "La commande marche"
        else:
            message = output
    else:
        message = error
```

At the end of all this, the message is sent back to the client, the connection is closed if he decides to close it and the last commands allow to launch the server.

```
self.__conn.send(message.encode())

self.__conn.close()

serv = Server()
serv.start()
```