INTERNSHIP REPORT

ON

CLIENT SERVER MODEL

SUBMITTED IN PARTIAL FULFILLMENT FOR THE

INTERNSHIP OF

INTERNITY FOUNDATION

IN THE FIELD OF

DATA STRUCTURES AND ALGORITHMS

INTERNITY
'TOGETHER WE CAN'

Submitted By:

Akash Sharma

Aman Goyal

Harshit Garg

Lipika Chugh

Under the Guidance of :

SARTHAK AGARWAL

JULY ,2020

ABSTRACT

This report is about the client server network architecture in which each computer or process on the network is either a client which is low end computer which request services and or a server which is high end powerful computer which is highly capable of providing services to many different clients at the same time. The task which servers performs are like managing disk drives which are called file servers, managing printers which are called print servers or managing network data traffic known as network server. Clients are PCs which require or use these facilities from servers which include sharing files, printers, storage and sometimes processing. All the clients and servers in a network communicate by using different protocols which are set of rules or standards which govern the communication between any computers or devices connected in a network.

ACKNOWLEDGEMENT

We would like to express our special gratitude to Internity Foundation for giving us this wonderful opportunity of this internship, under the valuable supervision, timely suggestions and inspiration offered by our most respectable stream head Sarthak Agarwal and our coach, Sangeeta Jha. It was because of their continuous encouragement that this report reached its successful completion. Last but not the least we express our sincere thanks to all our friends who have patiently extended their support for accomplishing this undertaking.

Name:

Akash Sharma

Aman Goyal

Harshit Garg

Lipika Chugh

Contents

Chapter 1

Introduction to Socket Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server. They are the real backbones behind web browsing. In simpler terms there is a server and a client. Socket programming is started by importing the socket library and making a simple socket.

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Here we made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM. AF_INET refers to the address family ipv4. The SOCK_STREAM means connection oriented TCP protocol. Now we can connect to a server using this socket.

1.1 Background

Sockets have a long history. Their use originated with ARPANET in 1971 and later became an API in the Berkeley Software Distribution (BSD) operating system released in 1983 called Berkeley sockets.

When the Internet took off in the 1990s with the World Wide Web, so did network programming. Web servers and browsers weren't the only applications taking advantage of newly connected networks and using sockets. Client-server applications of all types and sizes came into widespread use.

Today, although the underlying protocols used by the socket API have evolved over the years, and we've seen new ones, the low-level API has remained the same.

The most common type of socket applications are client-server applications, where one side acts as the server and waits for connections from clients. This is the type of application that I'll be covering in this tutorial. More specifically, we'll look at the socket API for Internet sockets, sometimes called Berkeley or BSD sockets. There are also Unix domain sockets, which can only be used to communicate between processes on the same host. The primary socket API functions and methods in this module are:

- socket()
- bind()
- listen()
- accept()
- connect()
- connect_ex()
- send()
- recv()
- close()

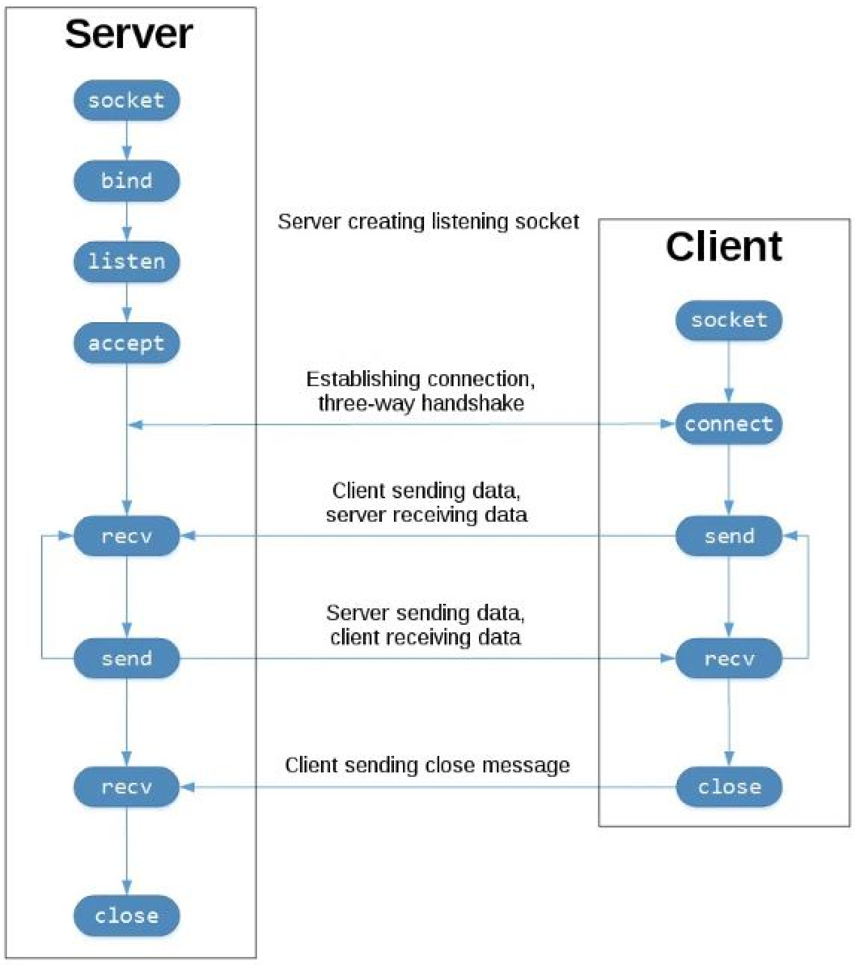

Server
socket
bind
listen
accept
recv
send
recv
close
Server creating listening socket
Establishing connection, three-way handshake
Client sending data, server receiving data
Server sending data, client receiving data
Client sending close message
Client
socket
connect
send
recv
close

Figure 1.1: TCP Socket Flow

The left-hand column represents the server. On the right-hand side is the client.

Starting in the top left-hand column, note the API calls the server makes to setup a "listening" socket:

- socket()
- bind()
- listen()
- accept()

A listening socket does just what it sounds like. It listens for connections from clients. When a client connects, the server calls accept() to accept, or complete, the connection.

The client calls connect() to establish a connection to the server and initiate the three-way handshake. The handshake step is important since it ensures that each side of the connection is reachable in the network, in other words that the client can reach the server and vice-versa. It may be that only one host, client or server, can reach the other.

In the middle is the round-trip section, where data is exchanged between the client and server using calls to send() and recv().

At the bottom, the client and server close() their respective sockets. The main key functions are:

1. socket.socket(): Create a new socket using the given address family, socket type and protocol number.

2. socket.bind(address): Bind the socket to address.

3. socket.listen(backlog): Listen for connections made to the socket. The backlog argument specifies the maximum number of queued connections and should be at least 0; the maximum value is system-dependent (usually 5), the minimum value is forced to 0.

4. socket.accept(): The return value is a pair (conn, address) where conn is a new socket object usable to send and receive data on the connection, and address is the address bound to the socket on the other end of the connection. At accept(), a new socket is created that is distinct from the named socket. This new socket is used solely for communication with this particular client. For TCP servers, the socket object used to receive connections is not the same socket used to perform subsequent communication with the client. In particular, the accept() system call returns a new socket object that's actually used for the connection. This allows a server to manage connections from a large number of clients simultaneously.

5. socket.send(bytes[, flags]): Send data to the socket. The socket must be connected to a remote socket. Returns the number of bytes sent. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data.

6. socket.colse(): Mark the socket closed. all future operations on the socket object will fail. The remote end will receive no more data (after queued data is flushed). Sockets are automatically closed when they are garbage-collected, but it is recommended to close() them explicitly.

Note that the server socket doesn't receive any data. It just produces client sockets. Each clientsocket is created in response to some other client socket doing a connect() to the host and port we're bound to. As soon as we've created that clientsocket, we go back to listening for more connections.

Chapter 2

Multi-Threading In python

Running several threads is similar to running several different programs concurrently, but with the following benefits

- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.

- Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.

• It can be pre-empted (interrupted)

• It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

Starting a New Thread To spawn another thread, you need to call following method available in thread module

thread.start_new_thread (function,args,[kwargs])

This method call enables a fast and efficient way to create new threads in both Linux and Windows. The method call returns immediately and the child thread starts and calls function with the passed list of args. When function returns, the thread terminates.

Chapter 3

Server Side

We will save python socket server program as socket_server.py. To use python socket connection, we need to import socket module.
Then, sequentially we need to perform some task to establish connection between server and client. We can obtain host address by using socket.gethostname() function. It is recommended to user port address above 1024 because port number lesser than 1024 are reserved for standard internet protocol.
See the appendix for python socket server example code, the comments will help you to understand the code.
So our python socket server is running on port 5000 and it will wait for client request. If you want server to not quit when client connection is closed, just remove the if condition and break statement. Python while loop is used to run the server program indefinitely and keep waiting for client request.

Chapter 4

Client Side

We will save python socket client program as socket_client.py. This program is similar to the server program, except binding.

The main difference between server and client program is, in server program, it needs to bind host address and port address together.

See the appendix for python socket client example code, the comment will help you to understand the code.

To see the output, first run the socket server program. Then run the socket client program. After that, write something from client program. Then again write reply from server program. At last, write bye from client program to terminate both program.

Chapter 5

Storing data using MYSQL

Sometimes it is useful to collect data from your website users and store this information in a MySQL database. We will add the practicality of allowing the data to be added through a user-friendly web form.

The first thing we will do is create a page with a form. For our demonstration we will make a very simple one:

Name:

Email:

Password:

As you can see the first thing we do is assign variables to the data from the previous page. We then just query the database to add this new information.

Of course, before we try it we need to make sure the table actually exists. Executing this code should create a table that can be used with our sample files.

CREATE TABLE data (name VARCHAR(30), email VARCHAR(30), password VARCHAR(30));

Now you know how to store user data in MySQL, so let's take it one step further and learn how to upload a file for storage.

The first thing you should notice is a field called id that is set to AUTO_INCREMENT. What this data type means is that it will count up to assign each file a unique file ID starting at 1 and going to 9999 (since we specified 4 digits).

You will also probably notice that our data field is called LONGBLOB. There are many types of BLOB as we have mentioned before. TINYBLOB, BLOB, MEDIUM-BLOB, and LONGBLOB are your options, but we set ours to LONGBLOB to allow for the largest possible files.

Next, we need to actually create upload.php, which will take our users file and store it in our database.

Chapter 6

Encryption/Decryption of Password

Cryptography is the art of communication between two users via coded messages. The science of cryptography emerged with the basic motive of providing security to the confidential messages transferred from one party to another.
Cryptography is defined as the art and science of concealing the message to introduce privacy and secrecy as recognized in information security.

6.1 Terminologies of Cryptography

The frequently used terms in cryptography are explained here

- Plain Text: The plain text message is the text which is readable and can be understood by all users. The plain text is the message which undergoes cryptography.
- Cipher Text: Cipher text is the message obtained after applying cryptography on plain text.
- Encryption: The process of converting plain text to cipher text is called encryption. It is also called as encoding.

- Decryption: The process of converting cipher text to plain text is called decryption. It is also termed as decoding.

6.2 Characteristics of Modern Cryptography

The basic characteristics of modern cryptography are as follows

- It operates on bit sequences.
- It uses mathematical algorithms for securing the information.
- It requires parties interested in secure communication channel to achieve privacy.

Chapter 7

Functionalities in the Project

The following functionalities have been included in the project:

- login : The user after signing up can login.
- signup : this is basically to create an account for the user.
- delete current account : If the user wants to delete his/her account then he/she can use this functionality.
- view current details : If the user wants to view what all are his/her current details, then he/she can use this functionality.
- update current details : If the user wants to update some of his details later on , then he/she can use this functionality.

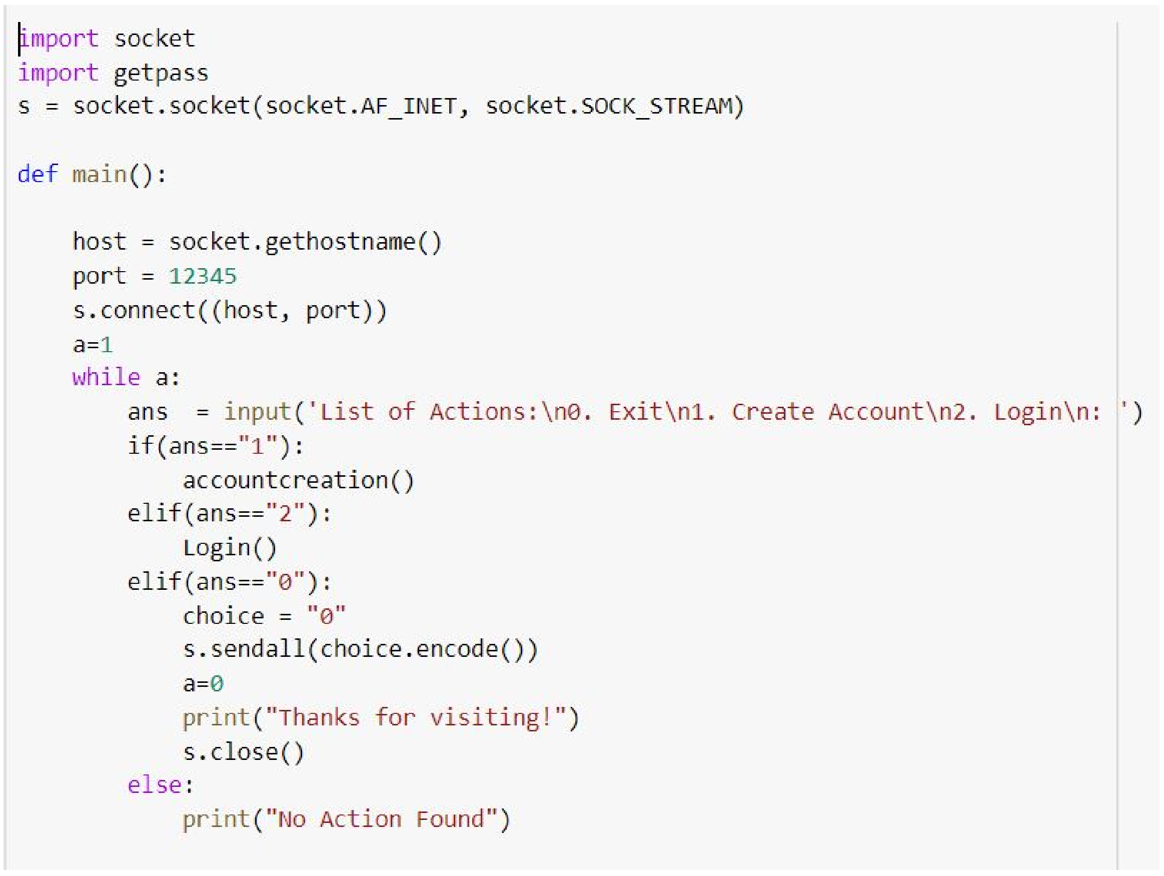
We already learned how to retrieve plain data from our MySQL database. Likewise, storing your files in a MySQL database wouldn't be very practical if there wasn't a way to retrieve them. The way we are going to learn to do this is by assigning each file a URL based on their ID number. If you will recall when we uploaded the files we automatically assigned each of the files an ID number. We will use that here when we call the files back.
Now to retrieve our file, we point our browser to: http://www.yoursite.com/download.php?id=2
(replace the 2 with whatever file ID you want to download/display)
This code is the base for doing a lot of things. With this as a base, you can add in a database query that would list files, and put them in a drop down menu for people

to choose. Or you could set ID to be a randomly created number so that a different graphic from your database is randomly displayed each time a person visits. The possibilities are endless.

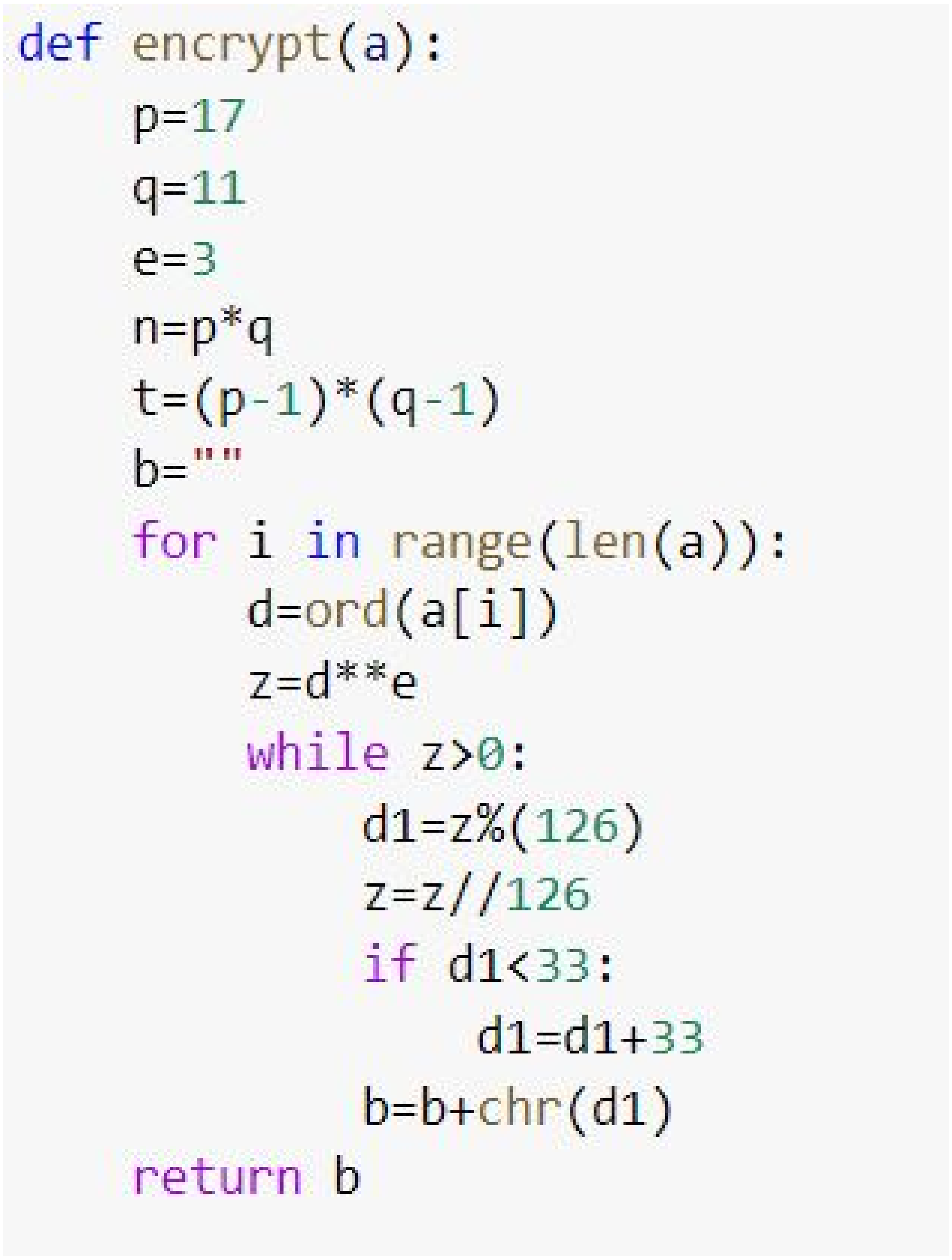
Like our previous code that downloaded files, this script allows files to be removed just by typing in their URL: http://yoursite.com/remove.php?id=2 (replace 2 with the ID you want to remove.) For obvious reasons, you want to be careful with this code. This is of course for demonstration, when we actually build applications we will want to put in safeguards that ask the user if they are sure they want to delete, or perhaps only allow people with a password to remove files. This simple code is the base we will build on to do all of those things.

APPENDIX

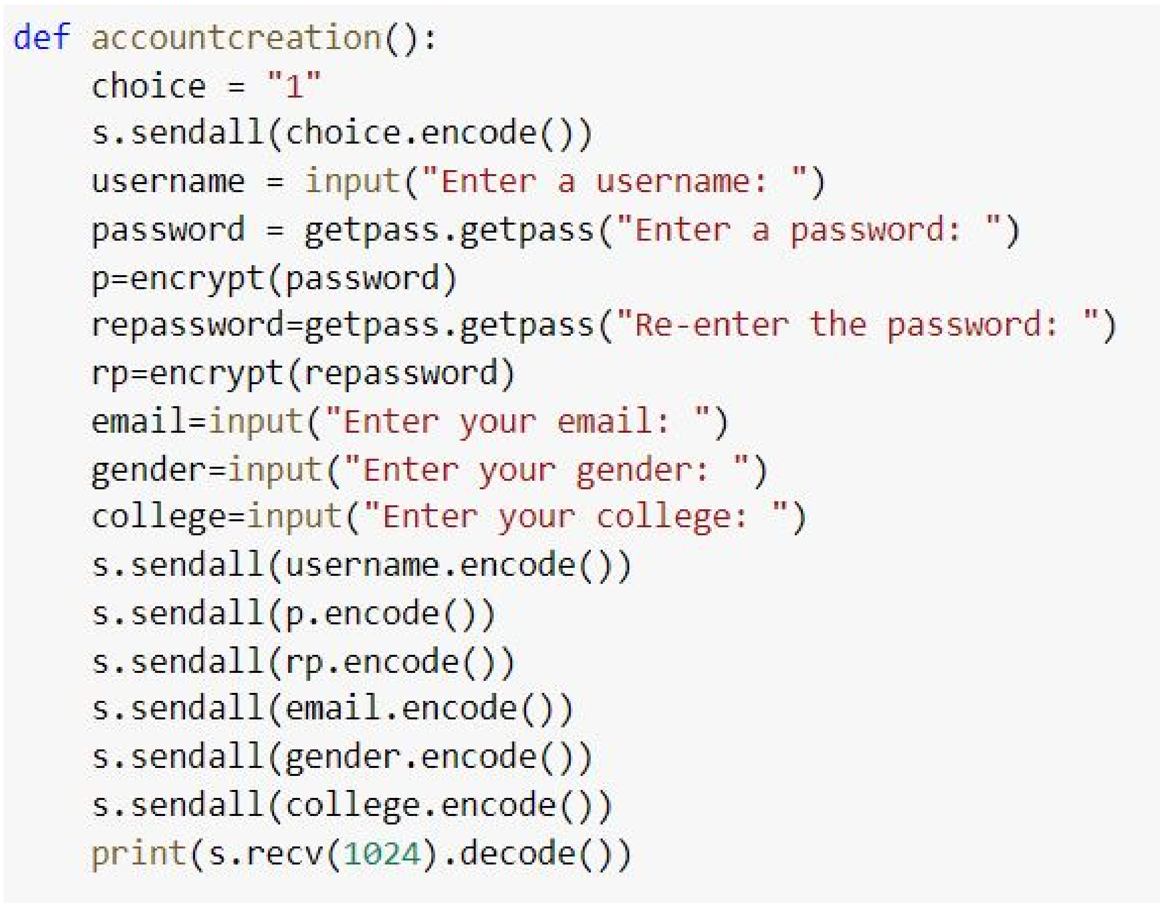
Client Side Code in Python



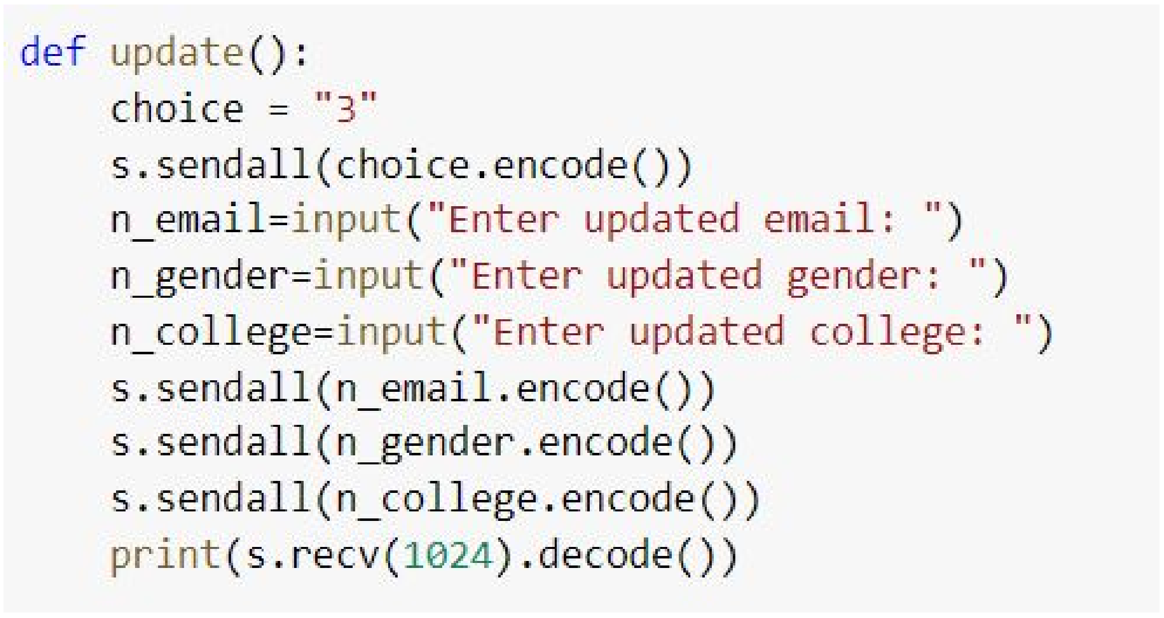
```
import socket
import getpass
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

def main():

    host = socket.gethostname()
    port = 12345
    s.connect((host, port))
    a=1
    while a:
        ans  = input('List of Actions:\n0. Exit\n1. Create Account\n2. Login\n: ')
        if(ans=="1"):
            accountcreation()
        elif(ans=="2"):
            Login()
        elif(ans=="0"):
            choice = "0"
            s.sendall(choice.encode())
            a=0
            print("Thanks for visiting!")
            s.close()
        else:
            print("No Action Found")
```



```python
def encrypt(a):
    p=17
    q=11
    e=3
    n=p*q
    t=(p-1)*(q-1)
    b=""
    for i in range(len(a)):
        d=ord(a[i])
        z=d**e
        while z>0:
            d1=z%(126)
            z=z//126
            if d1<33:
                d1=d1+33
            b=b+chr(d1)
    return b
```

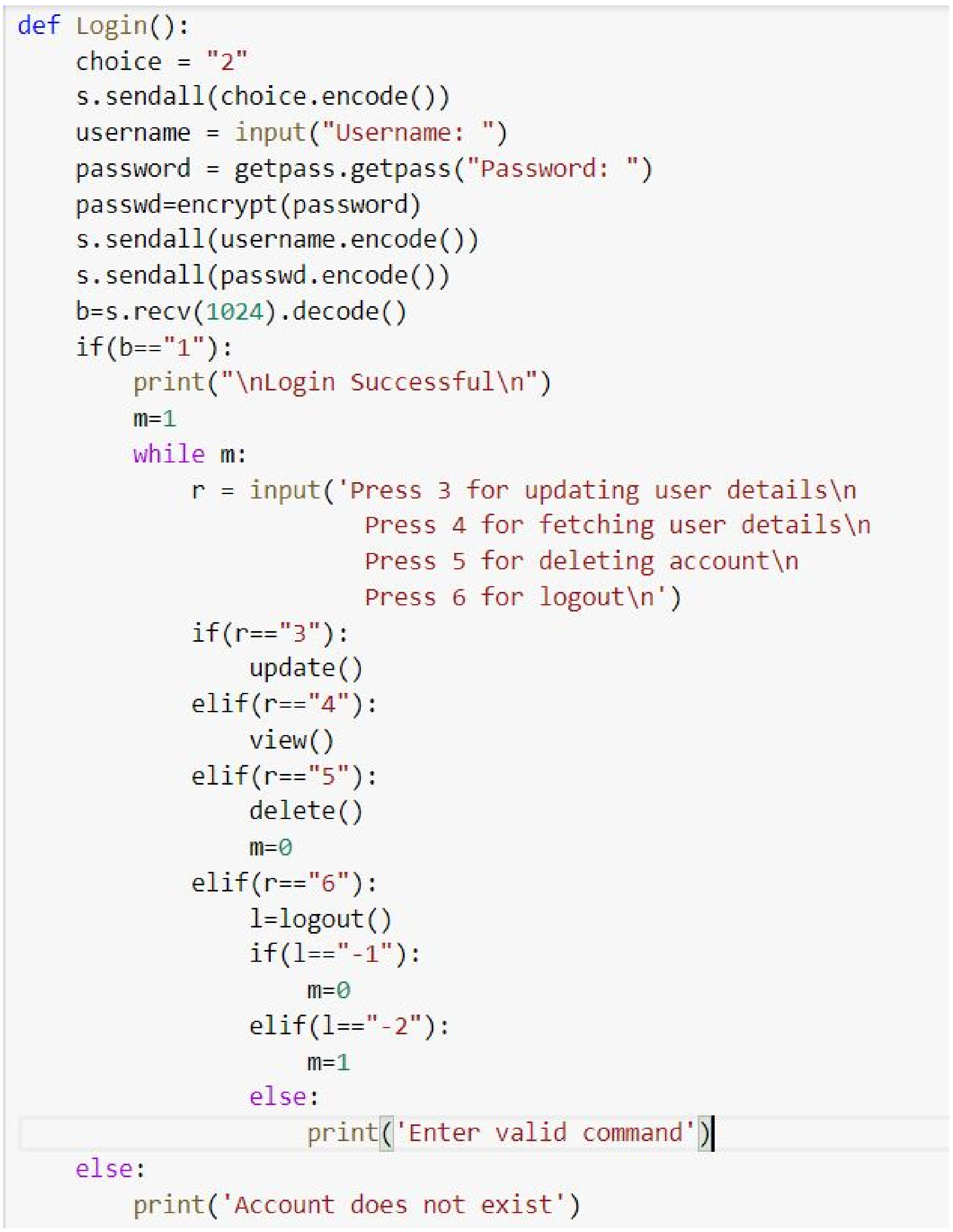


```python
def accountcreation():
    choice = "1"
    s.sendall(choice.encode())
    username = input("Enter a username: ")
    password = getpass.getpass("Enter a password: ")
    p=encrypt(password)
    repassword=getpass.getpass("Re-enter the password: ")
    rp=encrypt(repassword)
    email=input("Enter your email: ")
    gender=input("Enter your gender: ")
    college=input("Enter your college: ")
    s.sendall(username.encode())
    s.sendall(p.encode())
    s.sendall(rp.encode())
    s.sendall(email.encode())
    s.sendall(gender.encode())
    s.sendall(college.encode())
    print(s.recv(1024).decode())
```

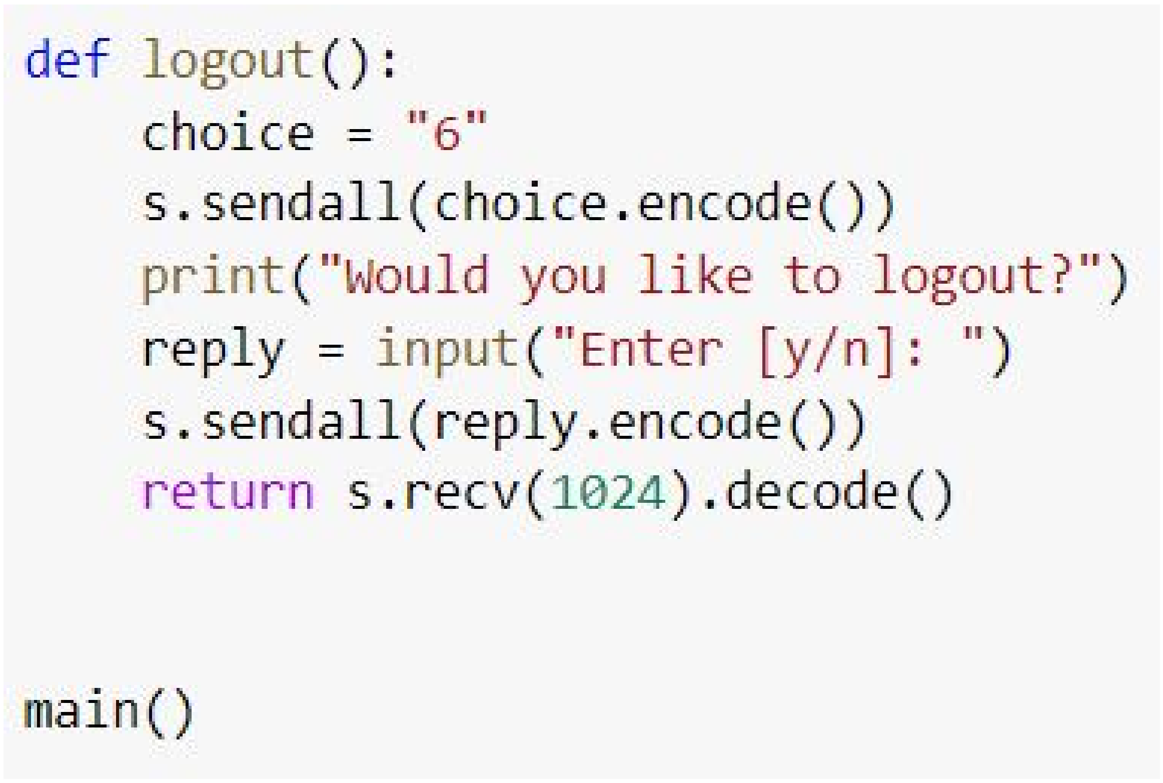


```python
def update():
    choice = "3"
    s.sendall(choice.encode())
    n_email=input("Enter updated email: ")
    n_gender=input("Enter updated gender: ")
    n_college=input("Enter updated college: ")
    s.sendall(n_email.encode())
    s.sendall(n_gender.encode())
    s.sendall(n_college.encode())
    print(s.recv(1024).decode())
```

```
def view():
    choice = "4"
    s.sendall(choice.encode())
    print(s.recv(1024).decode())

def delete():
    choice = "5"
    s.sendall(choice.encode())
    print(s.recv(1024).decode())
```

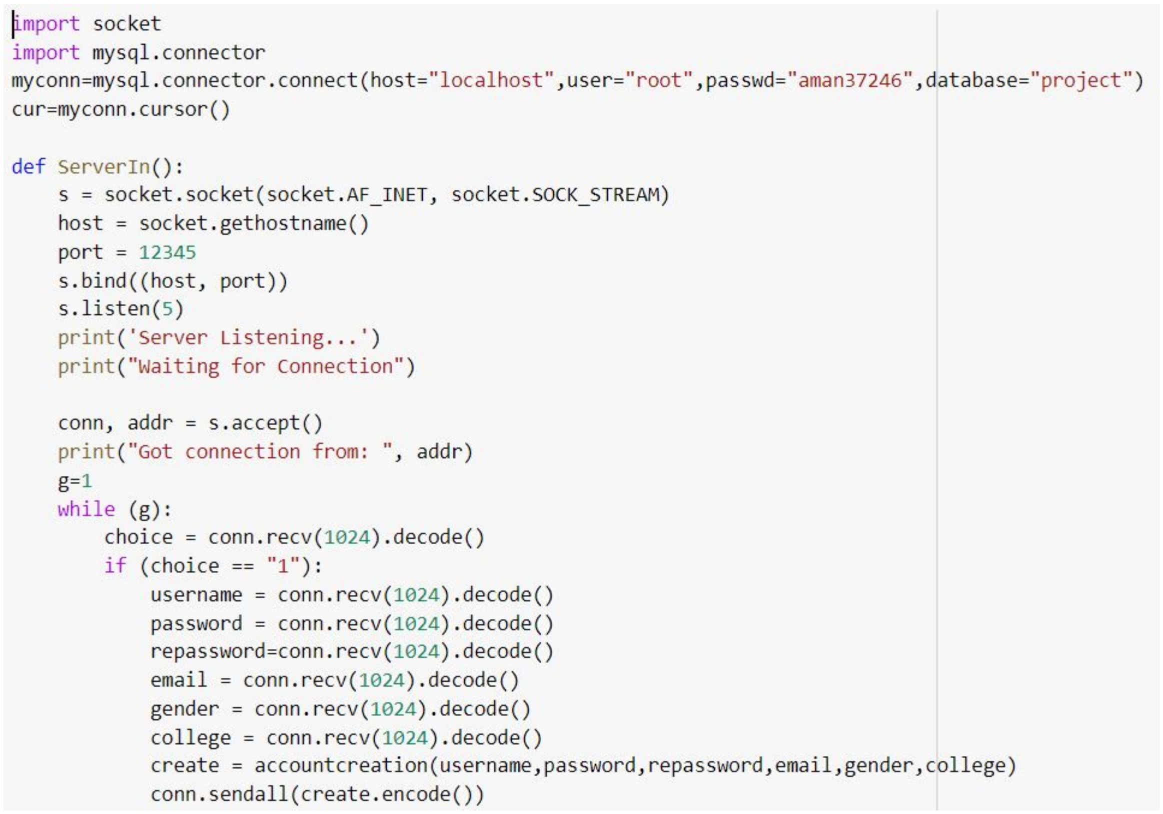


```python
def Login():
    choice = "2"
    s.sendall(choice.encode())
    username = input("Username: ")
    password = getpass.getpass("Password: ")
    passwd=encrypt(password)
    s.sendall(username.encode())
    s.sendall(passwd.encode())
    b=s.recv(1024).decode()
    if(b=="1"):
        print("\nLogin Successful\n")
        m=1
        while m:
            r = input('Press 3 for updating user details\n
                        Press 4 for fetching user details\n
                        Press 5 for deleting account\n
                        Press 6 for logout\n')
            if(r=="3"):
                update()
            elif(r=="4"):
                view()
            elif(r=="5"):
                delete()
                m=0
            elif(r=="6"):
                l=logout()
                if(l=="-1"):
                    m=0
                elif(l=="-2"):
                    m=1
                else:
                    print('Enter valid command')
    else:
        print('Account does not exist')
```

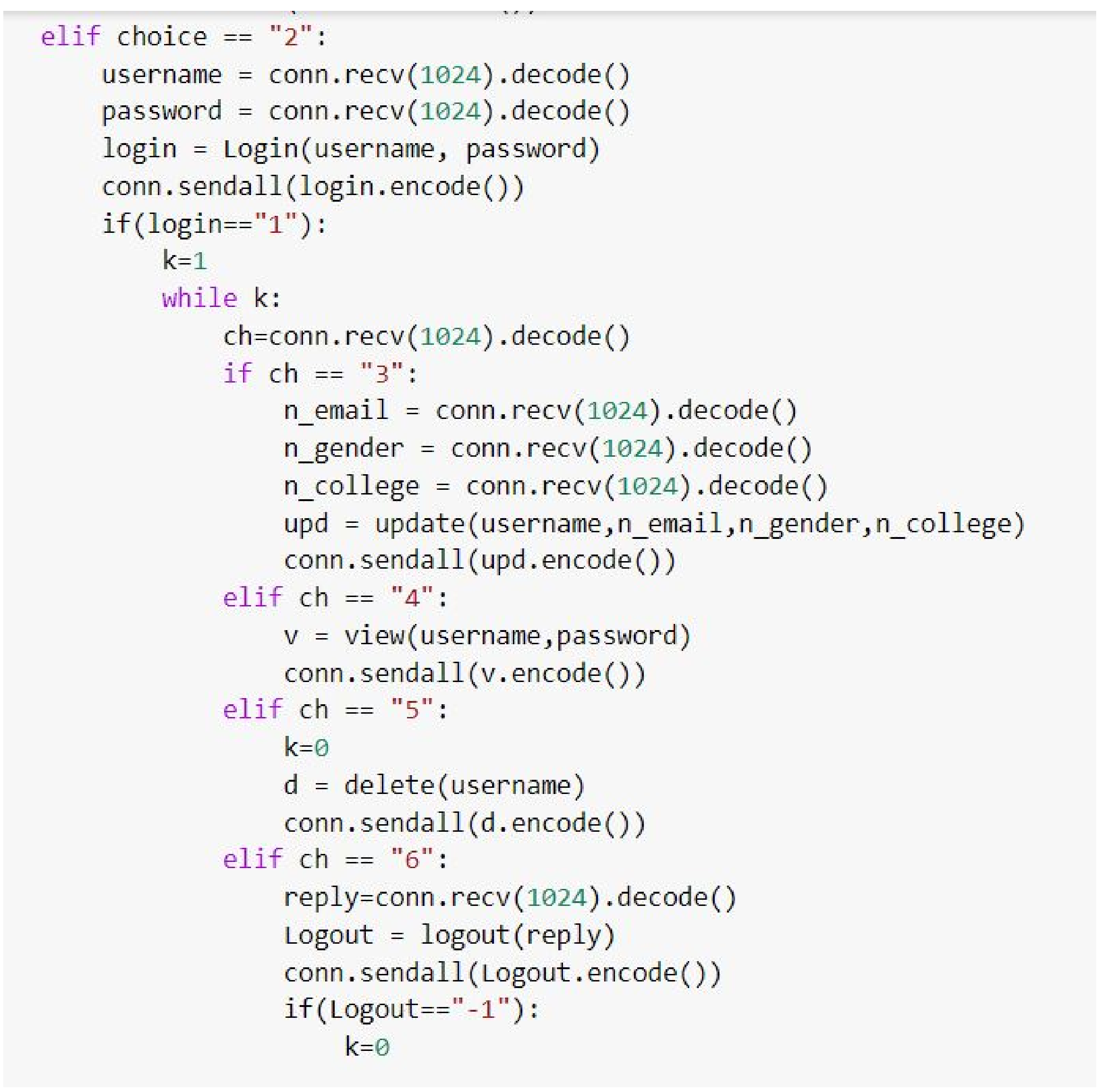


```python
def logout():
    choice = "6"
    s.sendall(choice.encode())
    print("Would you like to logout?")
    reply = input("Enter [y/n]: ")
    s.sendall(reply.encode())
    return s.recv(1024).decode()


main()
```

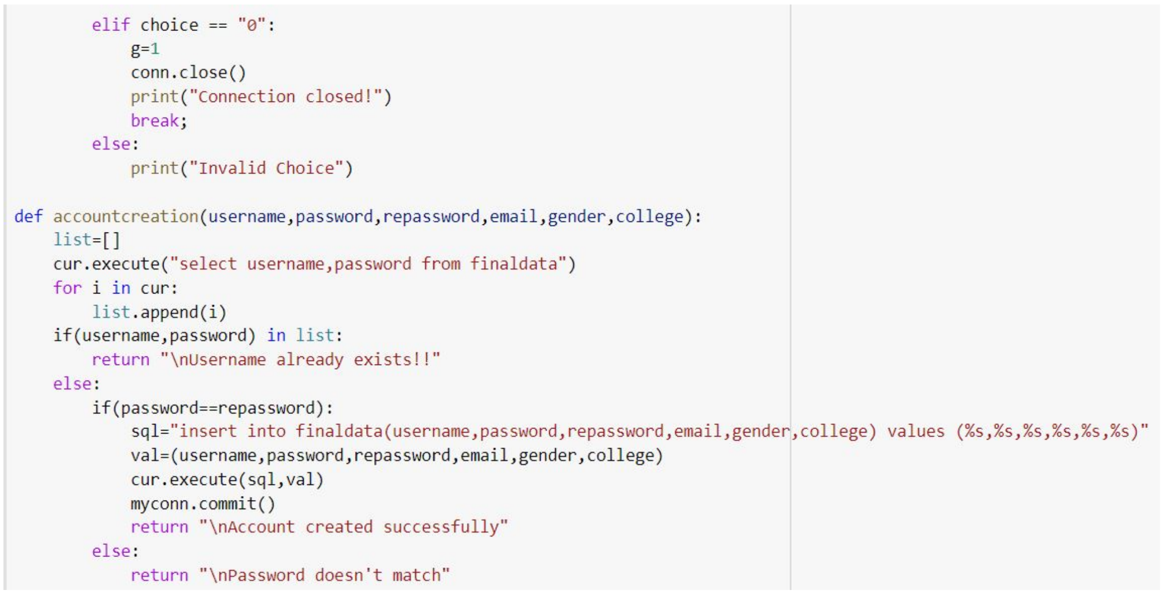
Server Side Code in Python



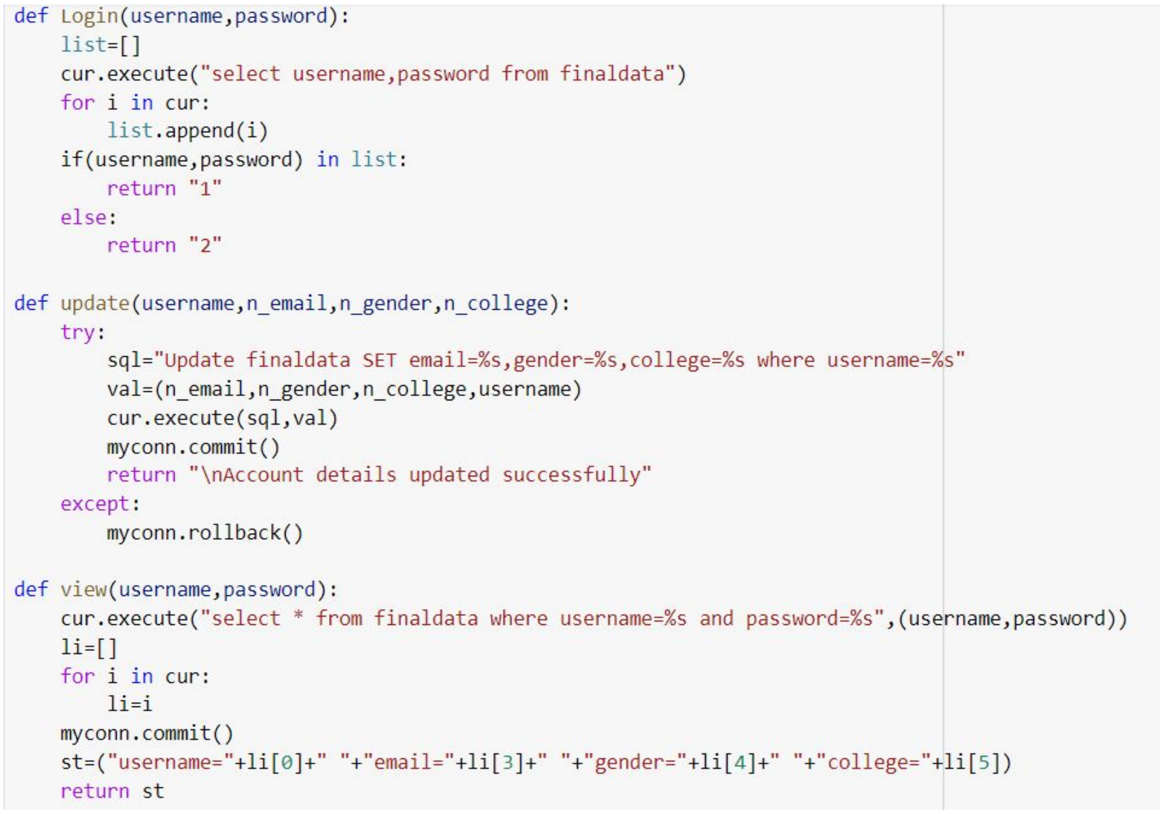
```python
import socket
import mysql.connector
myconn=mysql.connector.connect(host="localhost",user="root",passwd="aman37246",database="project")
cur=myconn.cursor()

def ServerIn():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = socket.gethostname()
    port = 12345
    s.bind((host, port))
    s.listen(5)
    print('Server Listening...')
    print("Waiting for Connection")

    conn, addr = s.accept()
    print("Got connection from: ", addr)
    g=1
    while (g):
        choice = conn.recv(1024).decode()
        if (choice == "1"):
            username = conn.recv(1024).decode()
            password = conn.recv(1024).decode()
            repassword=conn.recv(1024).decode()
            email = conn.recv(1024).decode()
            gender = conn.recv(1024).decode()
            college = conn.recv(1024).decode()
            create = accountcreation(username,password,repassword,email,gender,college)
            conn.sendall(create.encode())
```



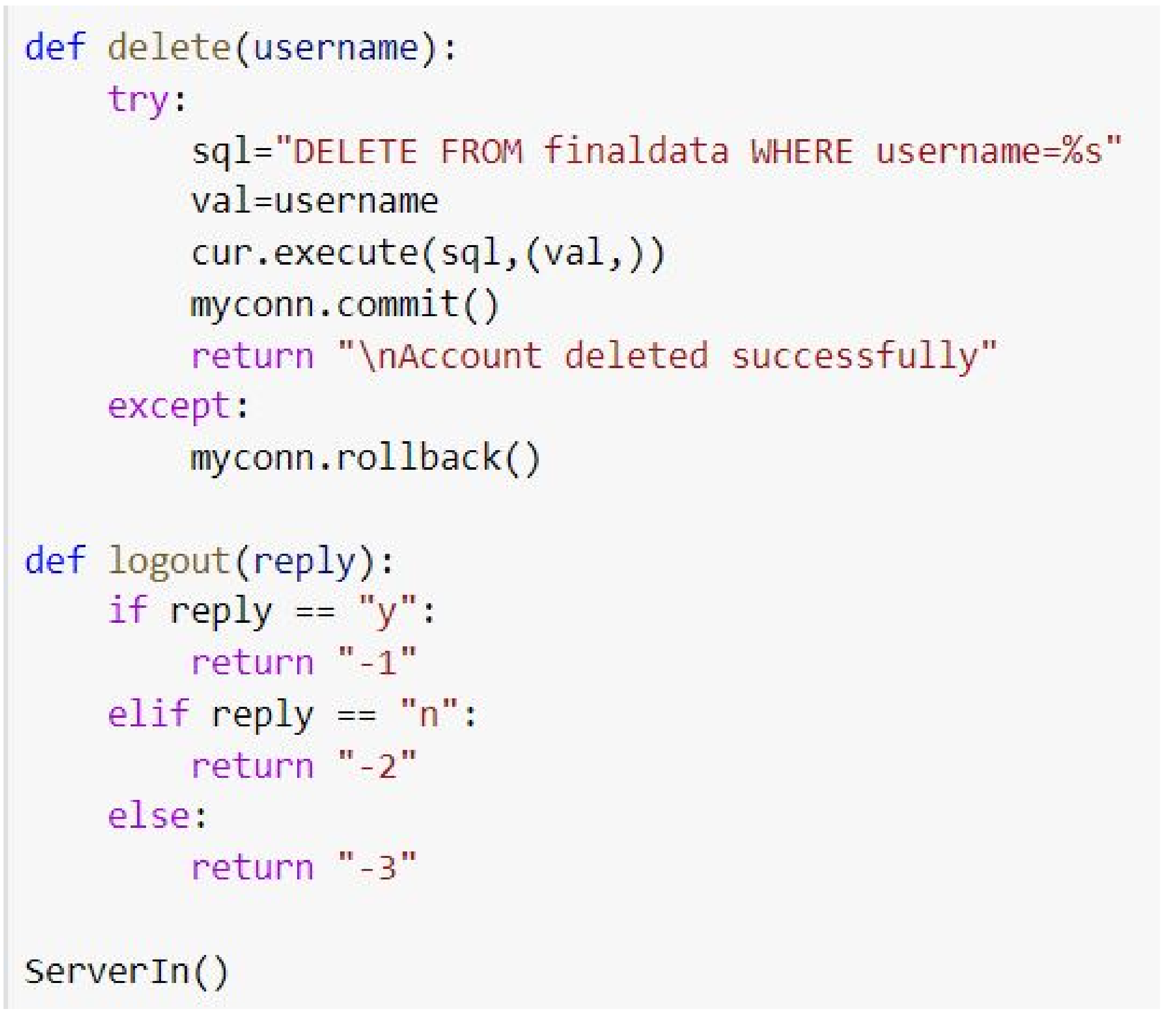
```
        elif choice == "2":
            username = conn.recv(1024).decode()
            password = conn.recv(1024).decode()
            login = Login(username, password)
            conn.sendall(login.encode())
            if(login=="1"):
                k=1
                while k:
                    ch=conn.recv(1024).decode()
                    if ch == "3":
                        n_email = conn.recv(1024).decode()
                        n_gender = conn.recv(1024).decode()
                        n_college = conn.recv(1024).decode()
                        upd = update(username,n_email,n_gender,n_college)
                        conn.sendall(upd.encode())
                    elif ch == "4":
                        v = view(username,password)
                        conn.sendall(v.encode())
                    elif ch == "5":
                        k=0
                        d = delete(username)
                        conn.sendall(d.encode())
                    elif ch == "6":
                        reply=conn.recv(1024).decode()
                        Logout = logout(reply)
                        conn.sendall(Logout.encode())
                        if(Logout=="-1"):
                            k=0
```



```
        elif choice == "0":
            g=1
            conn.close()
            print("Connection closed!")
            break;
        else:
            print("Invalid Choice")

def accountcreation(username,password,repassword,email,gender,college):
    list=[]
    cur.execute("select username,password from finaldata")
    for i in cur:
        list.append(i)
    if(username,password) in list:
        return "\nUsername already exists!!"
    else:
        if(password==repassword):
            sql="insert into finaldata(username,password,repassword,email,gender,college) values (%s,%s,%s,%s,%s,%s)"
            val=(username,password,repassword,email,gender,college)
            cur.execute(sql,val)
            myconn.commit()
            return "\nAccount created successfully"
        else:
            return "\nPassword doesn't match"
```



```python
def Login(username,password):
    list=[]
    cur.execute("select username,password from finaldata")
    for i in cur:
        list.append(i)
    if(username,password) in list:
        return "1"
    else:
        return "2"

def update(username,n_email,n_gender,n_college):
    try:
        sql="Update finaldata SET email=%s,gender=%s,college=%s where username=%s"
        val=(n_email,n_gender,n_college,username)
        cur.execute(sql,val)
        myconn.commit()
        return "\nAccount details updated successfully"
    except:
        myconn.rollback()

def view(username,password):
    cur.execute("select * from finaldata where username=%s and password=%s",(username,password))
    li=[]
    for i in cur:
        li=i
    myconn.commit()
    st=("username="+li[0]+" "+"email="+li[3]+" "+"gender="+li[4]+" "+"college="+li[5])
    return st
```



```python
def delete(username):
    try:
        sql="DELETE FROM finaldata WHERE username=%s"
        val=username
        cur.execute(sql,(val,))
        myconn.commit()
        return "\nAccount deleted successfully"
    except:
        myconn.rollback()

def logout(reply):
    if reply == "y":
        return "-1"
    elif reply == "n":
        return "-2"
    else:
        return "-3"

ServerIn()
```