CS359 - Assignment 3

client-server calculator

Shreyansh Kumar
2001CS86

Overview

Python socket programming is a way to establish a connection between a client and a server for exchanging data over a network. Sockets are endpoints of a two-way communication link between two processes running on a network. Python provides a socket module that enables us to create socket objects for establishing connections.

Socket programming in Python follows a client-server architecture, where a client requests services from a server, and the server responds to the client. Python socket programming allows us to create sockets with various properties such as type, family, and protocol. It also provides functions for binding, listening, accepting connections, and sending and receiving data between the client and the server.

Python socket programming can be used to create various network applications such as chat applications, file transfer protocols, and web servers. It is widely used for developing network-based applications due to its simplicity and ease of use.

Client

This Python code is a client-side script that establishes a TCP socket connection with a server and exchanges messages with it. The code begins by importing necessary modules such as time, socket, sys, and random. It then defines the localhost and port number using the sys.argv method to take input arguments from the command line. The port number is converted to an integer, and a socket instance is created.

The code then connects to the server using the connect() method of the socket instance, and sets the socket to non-blocking mode using the setblocking() method. The client then waits for the server to send a connection message, and once received, the client sends an arithmetic operation to the server and receives the result.

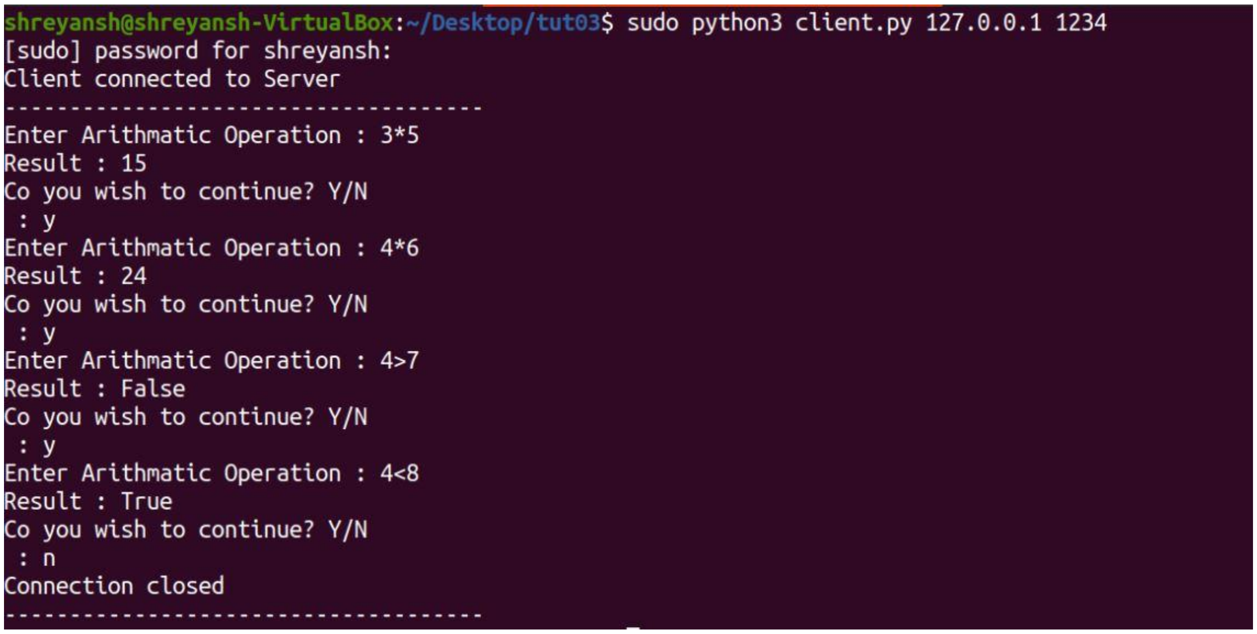
The code runs an infinite loop to send and receive messages with the server until the user decides to end the connection. The client sends an arithmetic operation to the server and receives the result, which is then displayed to the user. The code also prompts the user if they want to continue, and if not, sends a signal to the server to close the connection.

Finally, the client-side code prints a message indicating that the connection has been closed and closes the socket connection. This code can be used to connect to a server and exchange messages in a client-server architecture.

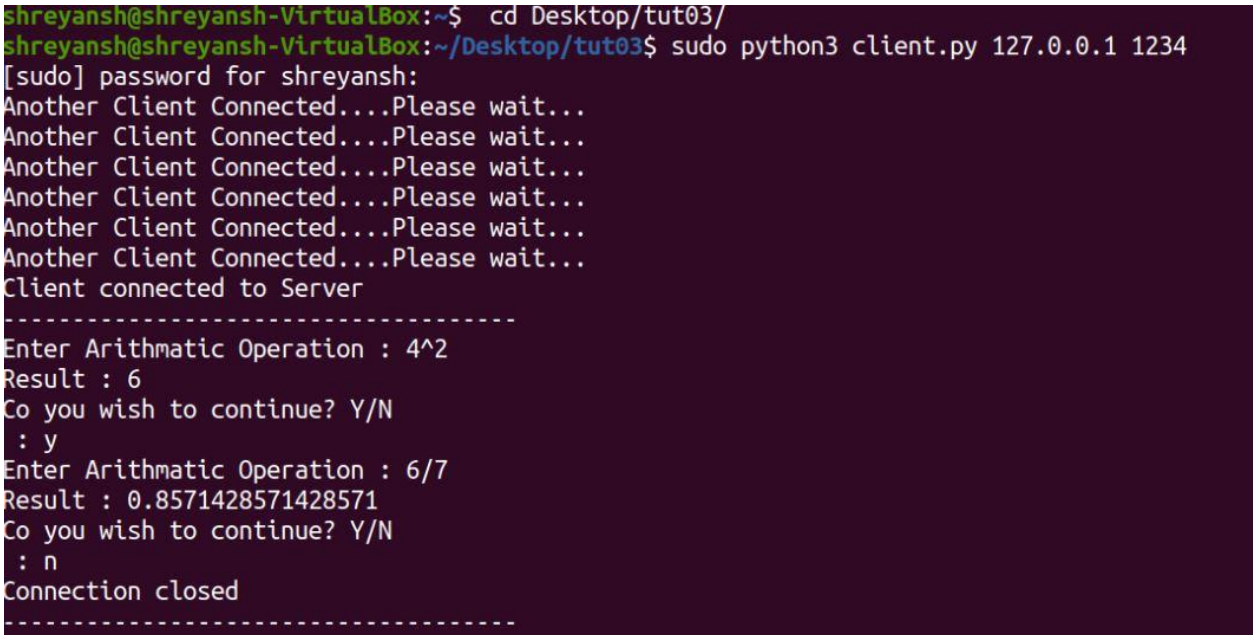
```
# Client Side


# Importing necessary modules

import time

import socket

import sys

from random import randint


# Defining the localhost and port number

LOCALHOST = sys.argv[1]

PORT = sys.argv[2]


# Converting port number to integer

PORT = int(PORT)


# Creating a socket instance
```

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connecting to the server
client.connect((LOCALHOST, PORT))

# Setting the socket to non-blocking mode
client.setblocking(0)

# Waiting for the server to send a connection message
while True:
    try:
        ans = client.recv(1024)
        ans = ans.decode()
        print("Client connected to Server")
        break
    except:
        print("Another client connected. Please wait...")
        time.sleep(5)

print("-------------------------------------")
# Running an infinite loop to send and receive messages with the server
while True:
    # Getting input from the user
    inp = input("Enter arithmetic operation: ")
    # Sending input to the server
    client.send(inp.encode())

    # Receiving output from the server
    while True:
        try:
            answer = client.recv(1024)
            break
        except:
            arr = 3

    # Displaying the result to the user
    print("Result: " + answer.decode())

    # Asking the user if they want to continue
    print("Do you wish to continue? Y/N")
    inp = input(" : ")
    if inp == "n" or inp == "N":
        # Sending a signal to the server to close the connection
        client.send(inp.encode())
        break

# Closing the connection
print("Connection closed")
print("-------------------------------------")
client.close()
```

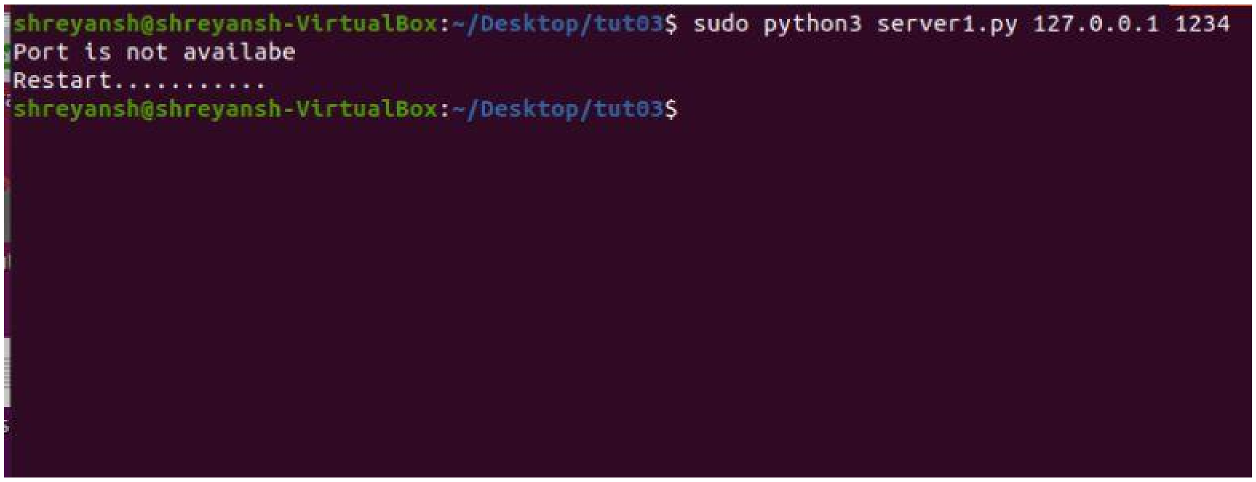
outputs -



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.1 1234
[sudo] password for shreyansh:
Client connected to Server
-------------------------------------
Enter Arithmatic Operation : 3*5
Result : 15
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 4*6
Result : 24
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 4>7
Result : False
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 4<8
Result : True
Co you wish to continue? Y/N
 : n
Connection closed
-------------------------------------
```



```
shreyansh@shreyansh-VirtualBox:~$  cd Desktop/tut03/
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.1 1234
[sudo] password for shreyansh:
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Client connected to Server
-------------------------------------
Enter Arithmatic Operation : 4^2
Result : 6
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 6/7
Result : 0.8571428571428571
Co you wish to continue? Y/N
 : n
Connection closed
-------------------------------------
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 server1.py 127.0.0.1 1234
Port is not availabe
Restart...........
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$
```

Server 1

This is a Python script for creating a server using socket programming. The script creates a server socket, binds it to the specified address and port number, and listens for incoming connections from clients.

The script starts by importing the necessary modules, including the socket and sys modules. It then creates a server socket object using the socket() function, and gets the server address and port number from the command-line arguments.

The script then checks if the specified port is available or not, and binds the socket to the specified address and port number using the bind() method. It starts the server and waits for incoming connections.

Once a client connects, the script accepts the connection, gets the client address and connection object, and sends an initial flag to the client indicating a successful connection.

The script then runs an infinite loop to receive and process client requests. It receives the client request using the recv() method and displays it on the server console. If the client sends 'n' or 'N', the script closes the connection with the client and breaks out of the loop.

If the client request is a valid arithmetic expression, the script evaluates it using the eval() function and sends the result back to the client using the send() method. If the expression is invalid, the script sends a message indicating that the expression is wrong.

Once the connection with the client is closed, the script prints a message indicating that the connection is closed, and closes the client connection object.

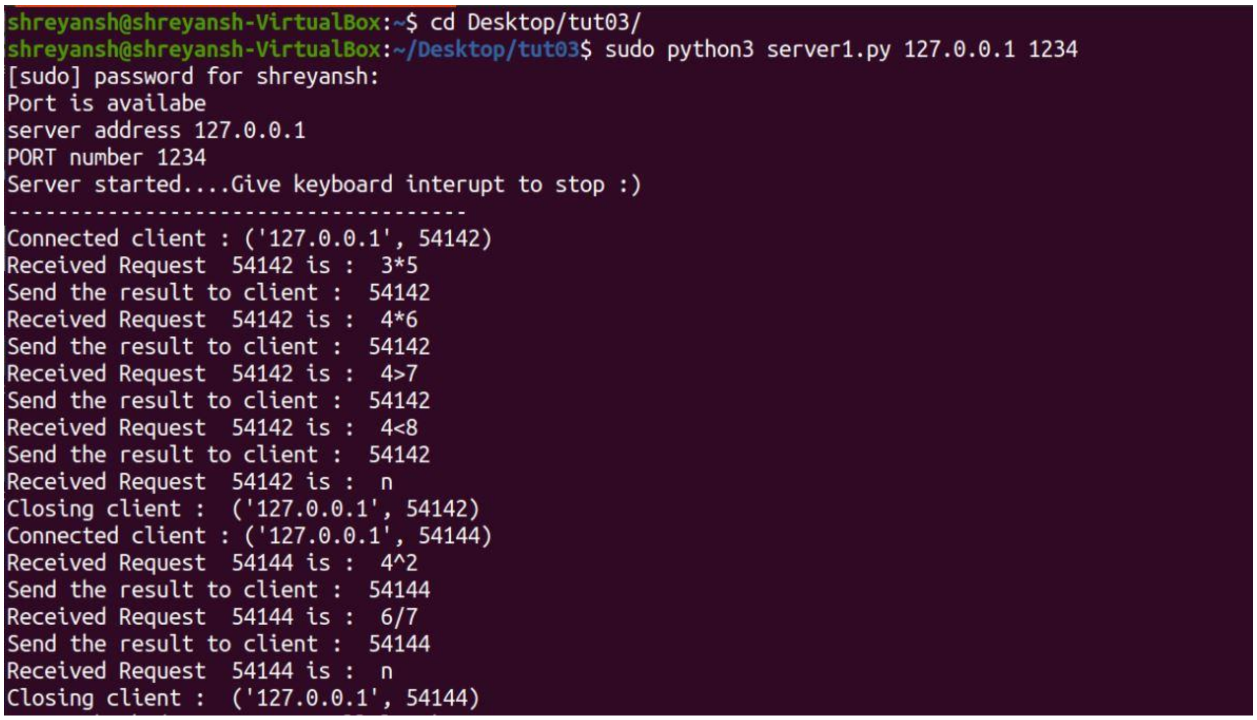
Overall, this script demonstrates how to create a simple server using socket programming in Python that can handle incoming client requests and return the appropriate response.

```
# server side


# Import socket module

import socket

import sys
```

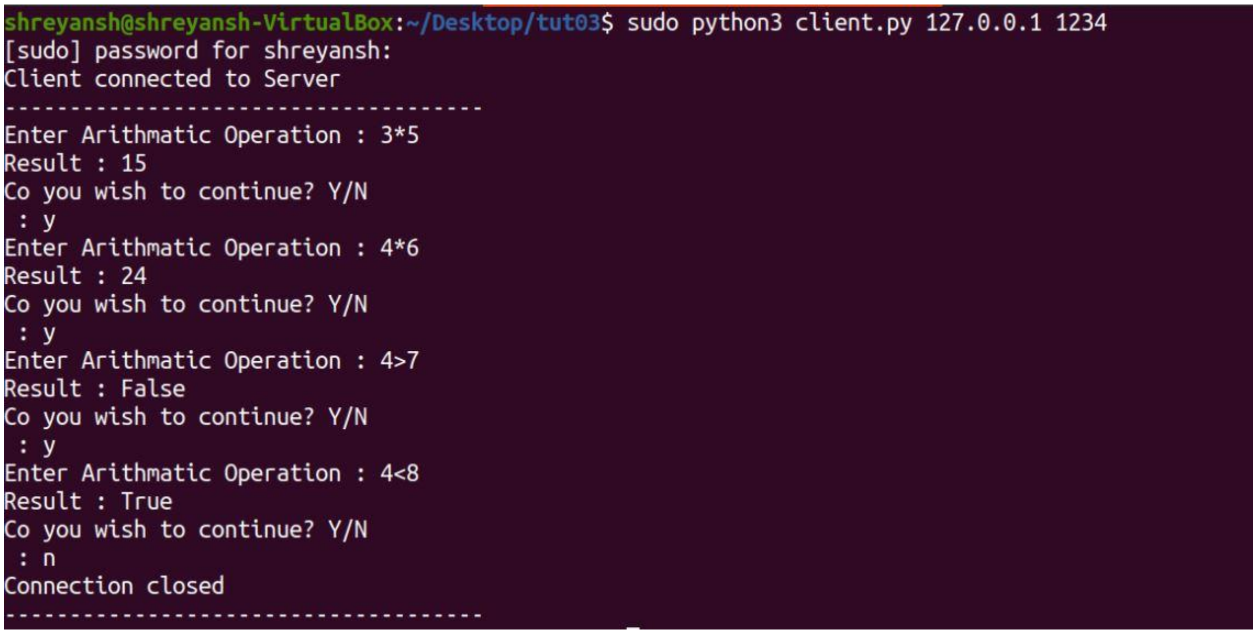
```
from random import randint

# Create server socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Get server address and port number from command line arguments
LOCALHOST = sys.argv[1]
PORT = sys.argv[2]
PORT = int(PORT)

# Check if the specified port is available or not
destination = (LOCALHOST, PORT)
result = server.connect_ex(destination)
if result == 0:
    print("Port is not available")
    print("Restart..........")
    exit()
else:
    print("Port is available")

# Bind the socket to the specified address and port number
print("Server address", LOCALHOST)
print("PORT number", PORT)
server.bind((LOCALHOST, PORT))

# Start the server and wait for incoming connections
print("Server started....Give keyboard interrupt to stop :)")
print("-------------------------------------")

while True:
    # Wait for incoming connections
```

```
server.listen(1)

# Accept the connection and get client address and connection object
clientConnection, clientAddress = server.accept()
print("Connected client :", clientAddress)

# Send initial flag to client indicating successful connection
flag = "1"
clientConnection.send(flag.encode())

# Running infinite loop to receive and process client requests
while True:
    # Receive client request
    msg = ''
    data = clientConnection.recv(1024)
    msg = data.decode()

    # Display received request from client
    print("Received Request ", clientAddress[1] ,"is : " , msg)

    # If client sends 'n' or 'N', close the connection with client
    if msg == "n" or msg == "N":
        print("Closing client : ",clientAddress)
        break

    # Evaluate client request and send the result back to client
    output = ''
    try:
        res = eval(msg)
        print("Send the result to client : ", clientAddress[1])
        output = str(res)
        except(NameError, SyntaxError):
            print("Wrong expression given to server.....")
            output = str("wrong expression.... resend your query")

        clientConnection.send(output.encode())

# Close the connection with client and stop the server
print("Connection closed")
print("-------------------------------------")
clientConnection.close()
```

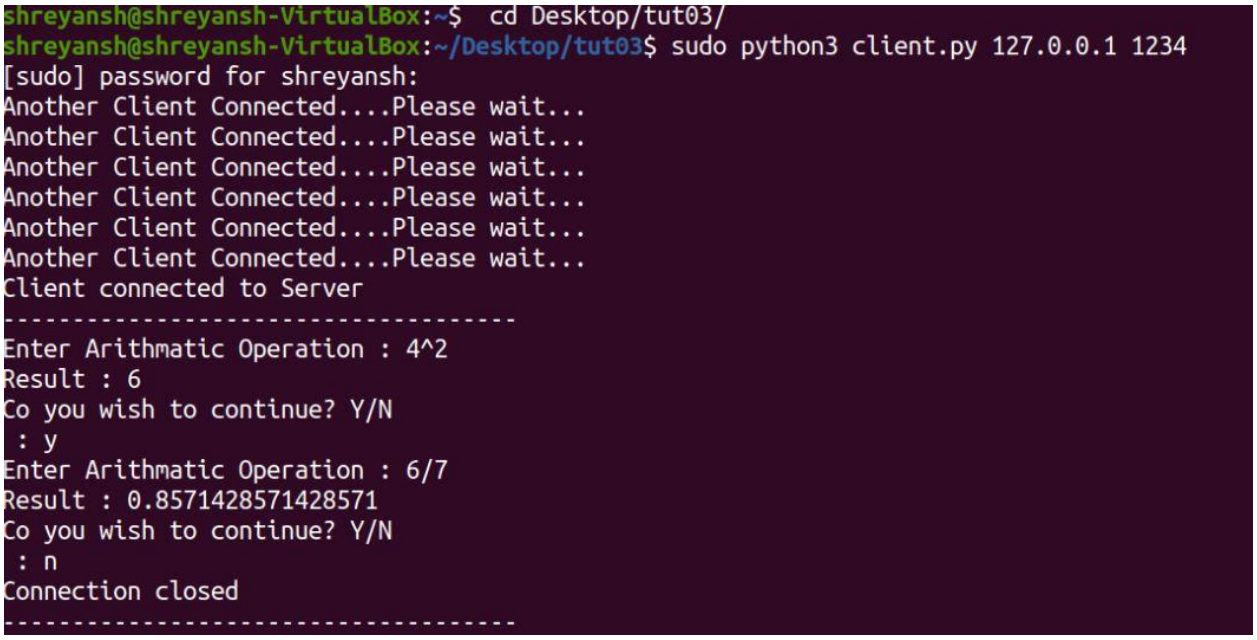
output -



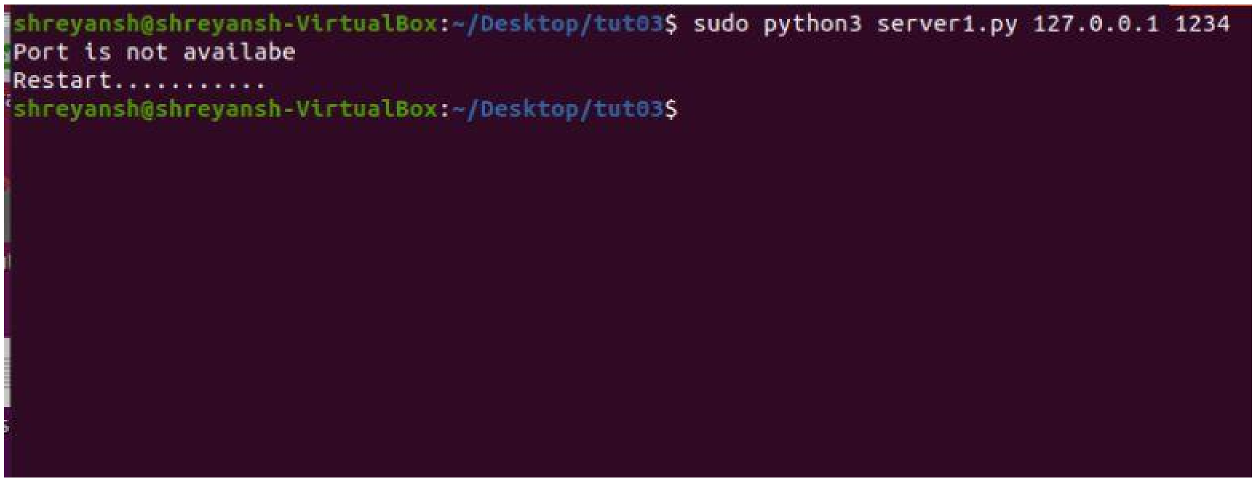
```
shreyansh@shreyansh-VirtualBox:~$ cd Desktop/tut03/
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 server1.py 127.0.0.1 1234
[sudo] password for shreyansh:
Port is availabe
server address 127.0.0.1
PORT number 1234
Server started....Give keyboard interupt to stop :)
-------------------------------------
Connected client : ('127.0.0.1', 54142)
Received Request  54142 is :  3*5
Send the result to client :  54142
Received Request  54142 is :  4*6
Send the result to client :  54142
Received Request  54142 is :  4>7
Send the result to client :  54142
Received Request  54142 is :  4<8
Send the result to client :  54142
Received Request  54142 is :  n
Closing client :  ('127.0.0.1', 54142)
Connected client : ('127.0.0.1', 54144)
Received Request  54144 is :  4^2
Send the result to client :  54144
Received Request  54144 is :  6/7
Send the result to client :  54144
Received Request  54144 is :  n
Closing client :  ('127.0.0.1', 54144)
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.1 1234
[sudo] password for shreyansh:
Client connected to Server
-------------------------------------
Enter Arithmatic Operation : 3*5
Result : 15
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 4*6
Result : 24
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 4>7
Result : False
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 4<8
Result : True
Co you wish to continue? Y/N
 : n
Connection closed
-------------------------------------
```



```
shreyansh@shreyansh-VirtualBox:~$  cd Desktop/tut03/
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.1 1234
[sudo] password for shreyansh:
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Another Client Connected....Please wait...
Client connected to Server
-------------------------------------
Enter Arithmatic Operation : 4^2
Result : 6
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 6/7
Result : 0.8571428571428571
Co you wish to continue? Y/N
 : n
Connection closed
-------------------------------------
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 server1.py 127.0.0.1 1234
Port is not availabe
Restart...........
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$
```

Server 2

The code provided is a Python script that implements a simple server-client model. The server listens on a specified IP address and port for incoming connections from clients, and once a connection is established, it spawns a new thread to handle the client's requests. The server is capable of receiving and evaluating simple Python expressions sent by the client and returns the result of the expression to the client.

The script uses the socket library to create and manage socket connections, the threading library to implement concurrent connections with multiple clients, and the _thread library to create new threads for each new client connection.

The conn_thread class extends the threading.Thread class and represents a new thread that handles a client's connection. The class takes the client's address, port number, and connection object as input and overrides the run() method, which is called when the thread is started. The run() method continuously listens for incoming data from the client, receives the data, evaluates it as a Python expression, and sends the result back to the client.

The script checks if the specified host IP address and port number are available for use and binds the server socket to the IP address and port number. It also checks for errors during the socket creation and binding processes and prints out appropriate error messages.

Finally, the script creates an empty list to hold thread objects and accepts incoming client connections. For each new connection, a new thread is created using the conn_thread class, and the thread is started to handle the client's requests. The script also sends a flag to the client to indicate that the connection has been established. The server continues to listen for new client connections until it is stopped manually by a keyboard interrupt.

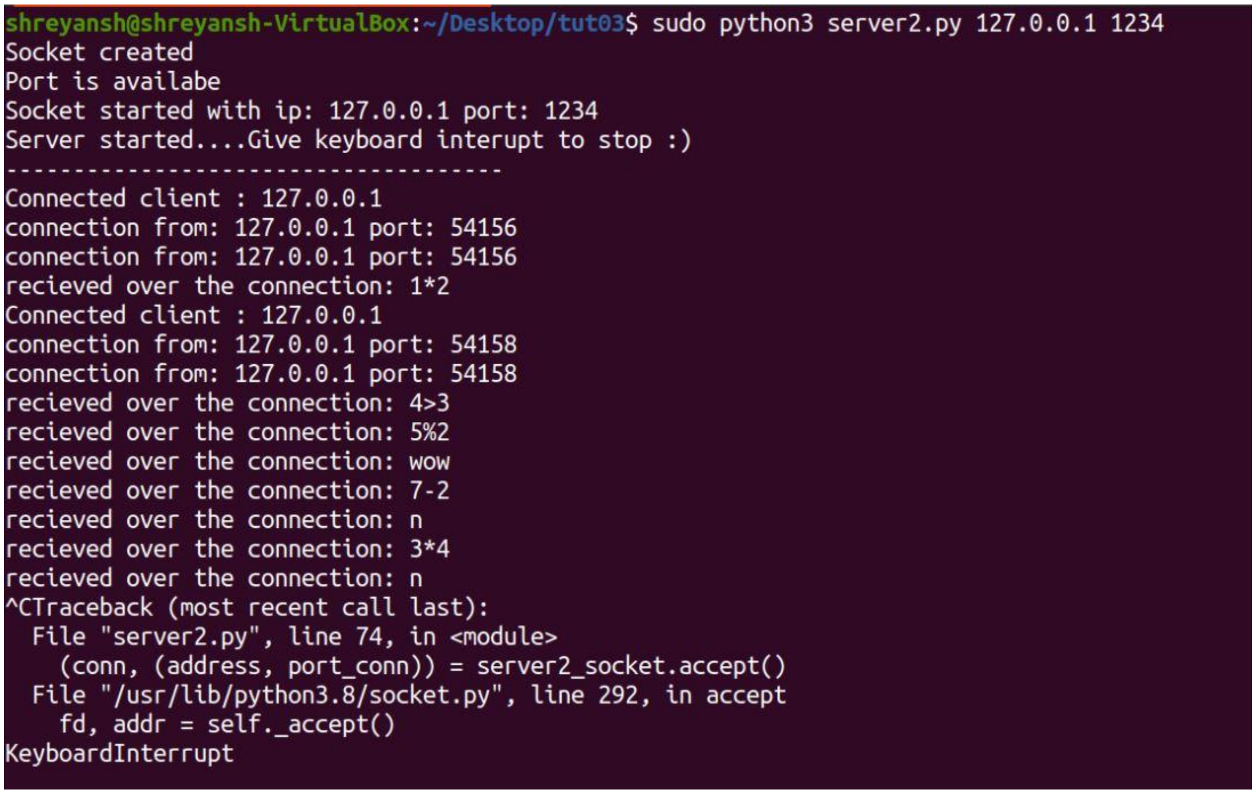
Overall, the script provides a basic example of how to implement a server-client model in Python using sockets and threads.

```python
# import required libraries

import socket as sk

import errno

import sys
```

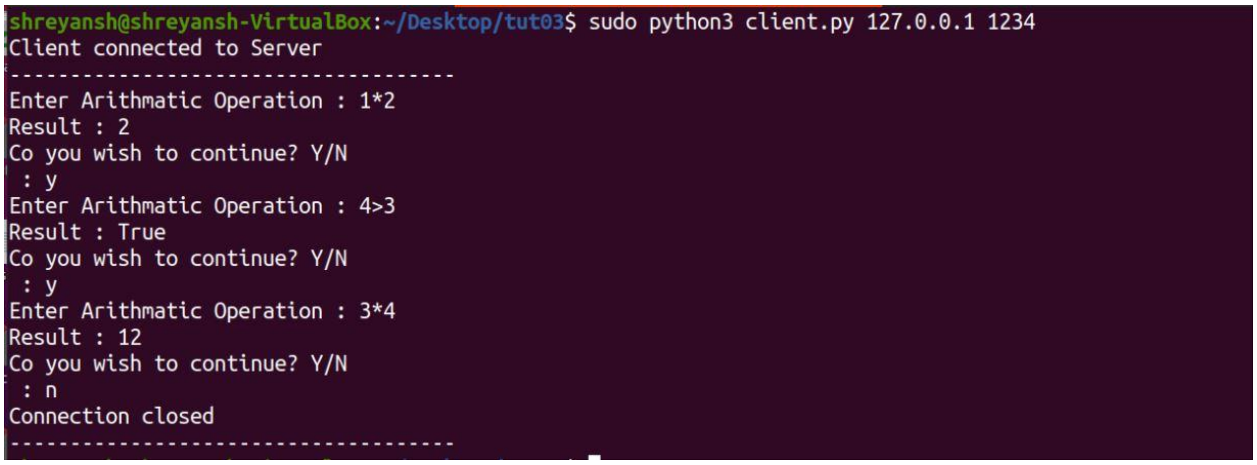
```python
from _thread import *

import threading


# Define a class that extends the threading.Thread class to handle client
connections

class conn_thread(threading.Thread):


    # Constructor that sets the client's address, port number and
connection object

    def __init__(self, address, port_conn, conn):

        threading.Thread.__init__(self)

        self.address = address

        self.port_conn = port_conn

        self.conn = conn

        print("connection from:", str(address), "port:", str(port_conn))


    # The run() method is called when a new thread is started

    def run(self):

        # Continuously listen for data from the client

        while True:

            # Receive data from the client and decode it

            data = self.conn.recv(1024).decode()

            # If the data is empty, break the loop

            if not data:

                break


            # Print the received data

            print("received over the connection:", str(data))


            # Initialize message variable

            msg = "try again"
            try:
                # Try to evaluate the received data as a Python expression
                msg = eval(str(data))
            except SyntaxError as err:
                # If the expression has a syntax error, send an error
message
                msg="Invalid syntax"
            except NameError as err:
                # If the expression contains an undefined variable, send
an error message
                msg="Please write an expression"

            # Send the message back to the client
            self.conn.send(str(msg).encode())



# Get the host IP address and port number from the command-line arguments
host_ip =  sys.argv[1]
port = sys.argv[2]
port = int (port)

try:
    # Create a new socket object
    server2_socket = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
    print("Socket created")
except sk.error as err:
    # If socket creation fails, print an error message and exit
    print("Socket creation failed with error: ", str(err))
    sys.exit()

# Check if the specified port number is available
destination = (host_ip, port)
```

```
result = server2_socket.connect_ex(destination)
if result == 0:
    print("Port is not available")
    print("Restart..........")
    exit()
else:
    print("Port is available")

try:
    # Bind the socket to the host IP address and port number
    server2_socket.bind((host_ip, port))
    print("Socket started with ip:", str(host_ip), "port:", str(port))
    print("Server started....Give keyboard interrupt to stop :)")
    print("-------------------------------------")
except sk.error as err:
    # If socket binding fails, print an error message and exit
    if err.errno == errno.EADDRINUSE:
        print("Port already in use")
        sys.exit()
    else:
        print("Socket binding failed with error: ", str(err))
        sys.exit()

# Create an empty list to hold thread objects
threads = []

while True:
    # Listen for incoming client connections
    server2_socket.listen(5)
    # Accept a new client connection
    (conn, (address, port_conn)) = server2_socket.accept()
    print("Connected client :", address)

    # Send a flag to the client indicating that the connection has been
established

    flag = "1"

    conn.send(flag.encode())


    print("connection from:", str(address), "port:", str(port_conn))


    # Create a new thread to handle the client connection

    # start_new_thread(thread_conn, (conn, ))

    newthread = conn_thread(address, port_conn, conn)

    newthread.start()

    threads.append(newthread)
```

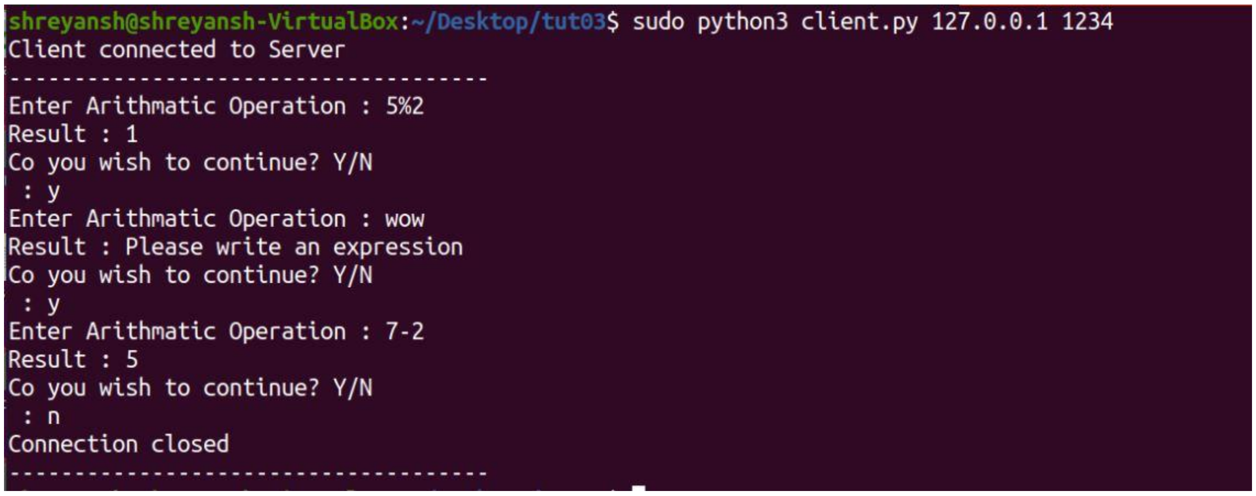
Output -



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 server2.py 127.0.0.1 1234
Socket created
Port is availabe
Socket started with ip: 127.0.0.1 port: 1234
Server started....Give keyboard interupt to stop :)
-------------------------------------
Connected client : 127.0.0.1
connection from: 127.0.0.1 port: 54156
connection from: 127.0.0.1 port: 54156
recieved over the connection: 1*2
Connected client : 127.0.0.1
connection from: 127.0.0.1 port: 54158
connection from: 127.0.0.1 port: 54158
recieved over the connection: 4>3
recieved over the connection: 5%2
recieved over the connection: wow
recieved over the connection: 7-2
recieved over the connection: n
recieved over the connection: 3*4
recieved over the connection: n
^CTraceback (most recent call last):
  File "server2.py", line 74, in <module>
    (conn, (address, port_conn)) = server2_socket.accept()
  File "/usr/lib/python3.8/socket.py", line 292, in accept
    fd, addr = self._accept()
KeyboardInterrupt
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.1 1234
Client connected to Server
-------------------------------------
Enter Arithmatic Operation : 1*2
Result : 2
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 4>3
Result : True
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 3*4
Result : 12
Co you wish to continue? Y/N
 : n
Connection closed
-------------------------------------
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.1 1234
Client connected to Server
-------------------------------------
Enter Arithmatic Operation : 5%2
Result : 1
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : wow
Result : Please write an expression
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 7-2
Result : 5
Co you wish to continue? Y/N
 : n
Connection closed
-------------------------------------
```

Server 3

This is a Python script that implements a basic server using sockets and the selectors module. The server listens for incoming connections and processes client requests. Upon successful connection, the server sends a flag to the client indicating the successful connection.

The server is designed to handle simple mathematical expressions submitted by the client. When the client submits an expression, the server evaluates it using the built-in eval() function in Python. If the expression is valid, the server sends the result back to the client. If the expression is invalid, the server sends an error message back to the client.

The script includes two functions, accept_wrapper() and service_connection(), which handle accepting new connections and serving existing connections, respectively. The accept_wrapper() function accepts a new connection and registers it with the selector object. The service_connection() function serves existing connections by receiving data from the client, processing the client request, and sending a response back to the client.

The script takes two arguments from the command line: the host address and the port number to listen on. The server attempts to bind to the given address and port, and if successful, it sets the server socket to non-blocking and registers it with the selector object.

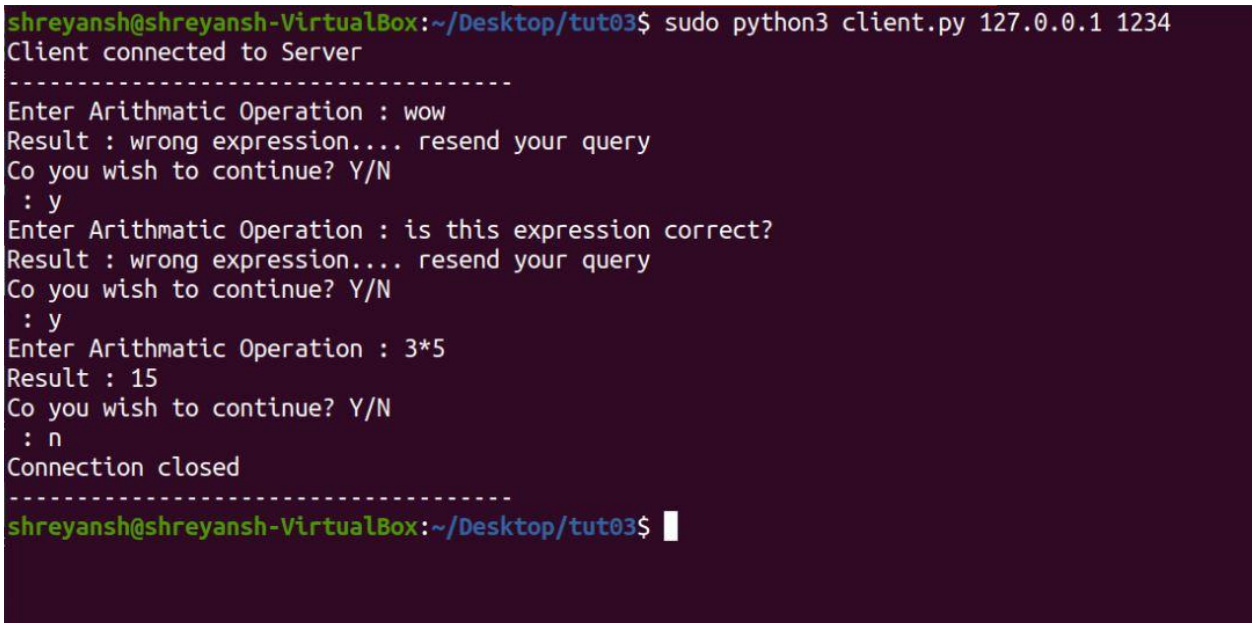
The script includes a try-except block to handle exceptions that may occur while binding the server to the given address and port. The script also includes a try-except block to handle keyboard interrupts, which will close the server.

Overall, this script provides a basic implementation of a server using sockets and the selectors module, which can handle incoming connections and process client requests.

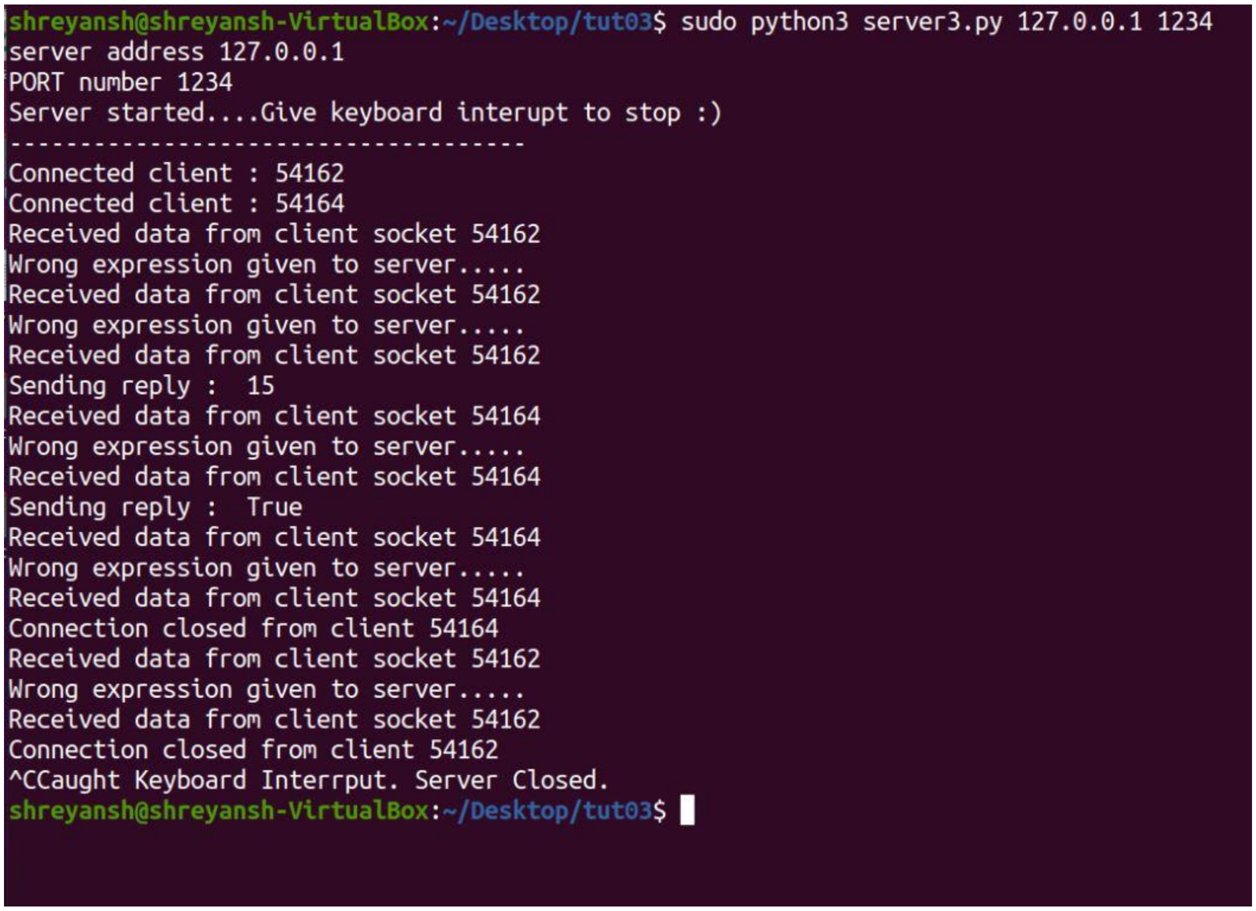
```python
import sys

import socket

import selectors

import types

from operator import pow, truediv, mul, add, sub


# Create a selector object for monitoring I/O events

sel = selectors.DefaultSelector()


# Function to handle accepting new connections

def accept_wrapper(sock):

    # Accept the connection and get client address

    conn, addr = sock.accept()

    print("Connected client :", addr[1])


    # Send a flag to the client indicating successful connection

    flag = "1"

    conn.send(flag.encode())


    # Set the connection to non-blocking and create a data object to store
connection info
```

```
    conn.setblocking(False)
    data = types.SimpleNamespace(addr=addr, inb="", outb="")

    # Register the connection with the selector object
    events = selectors.EVENT_READ | selectors.EVENT_WRITE
    sel.register(conn, events, data=data)

# Function to handle serving existing connections
def service_connection(key, mask):
    sock = key.fileobj
    data = key.data

    # If the socket is ready to read, receive data from the client
    if mask & selectors.EVENT_READ:
        recv_data = sock.recv(1024)
        print("Received data from client socket",data.addr[1])

        # If data was received, process the client request
        if recv_data:
            msg = str(recv_data.decode())
            output = ""

            # Try to evaluate the client request
            try:
                res = eval(msg)
                print("Sending reply : ", res )
                output = str(res)

            # If the client request is invalid, send an error message
            except(NameError, SyntaxError):
                print("Wrong expression given to server.....")
                output = str("wrong expression.... resend your query")

            # Send the response back to the client
            sock.sendall(str(output).encode())

        # If no data was received, the connection was closed by the client
        else:
            print("Connection closed from client", data.addr[1])
            sel.unregister(sock)
            sock.close()

# Get server host and port from command line arguments
host = sys.argv[1]
port = sys.argv[2]
port = int(port)

# Attempt to bind to the given address and port
try:
    lsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    lsock.bind((host, port))
    lsock.listen()
    print("server address",host)
    print("PORT number",port)

    # Set the server socket to non-blocking and register with the selector
object
    lsock.setblocking(False)
    sel.register(lsock, selectors.EVENT_READ, data=None)
    print("Server started....Give keyboard interupt to stop :)")
    print("-------------------------------------")
```

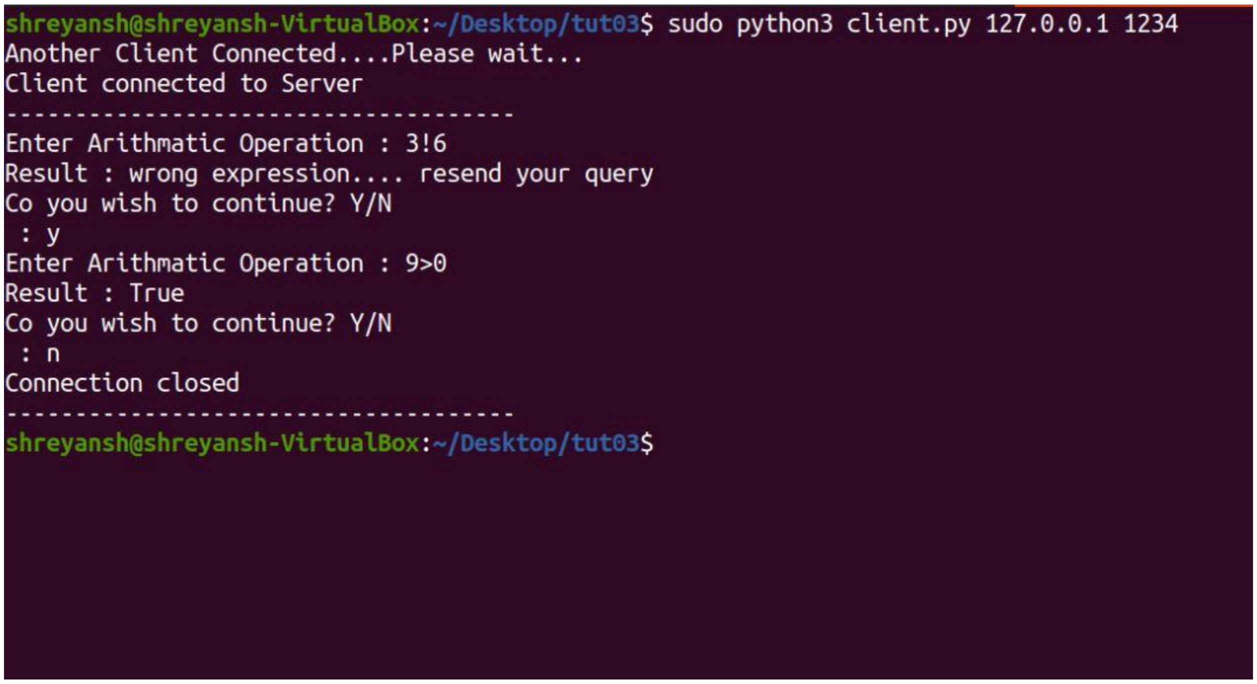
```
# If the address is already in use, print an error message and exit
except:
    print("Address already in use......RESTART")
    exit()

# Wait for connections and process requests until interrupted by keyboard
try:
    while True:
        events = sel.select(timeout=None)
        for key, mask in events:
            if key.data is None:
                accept_wrapper(key.fileobj)
            else:
                service_connection(key, mask)
except KeyboardInterrupt:
    # If keyboard interrupt is detected, close the server
        print("Caught Keyboard Interrput. Server Closed.")
finally:
        sel.close()
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.1 1234
Client connected to Server
------------------------------------
Enter Arithmatic Operation : wow
Result : wrong expression.... resend your query
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : is this expression correct?
Result : wrong expression.... resend your query
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 3*5
Result : 15
Co you wish to continue? Y/N
 : n
Connection closed
------------------------------------
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 server3.py 127.0.0.1 1234
server address 127.0.0.1
PORT number 1234
Server started....Give keyboard interupt to stop :)
------------------------------------
Connected client : 54162
Connected client : 54164
Received data from client socket 54162
Wrong expression given to server.....
Received data from client socket 54162
Wrong expression given to server.....
Received data from client socket 54162
Sending reply :  15
Received data from client socket 54164
Wrong expression given to server.....
Received data from client socket 54164
Sending reply :  True
Received data from client socket 54164
Wrong expression given to server.....
Received data from client socket 54164
Connection closed from client 54164
Received data from client socket 54162
Wrong expression given to server.....
Received data from client socket 54162
Connection closed from client 54162
^CCaught Keyboard Interrput. Server Closed.
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.1 1234
Another Client Connected....Please wait...
Client connected to Server
------------------------------------
Enter Arithmatic Operation : 3!6
Result : wrong expression.... resend your query
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 9>0
Result : True
Co you wish to continue? Y/N
 : n
Connection closed
------------------------------------
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$
```

Server 4

This is a Python script for creating a basic server that listens for incoming client connections, and echos back any data sent by the client. The script uses the built-in selectors and types modules to handle the I/O multiplexing and storage of per-connection data.

The script first creates a selectors.DefaultSelector() object to manage the I/O multiplexing for all sockets. It then defines two functions: accept_wrapper() to handle new client connections, and service_connection() to handle I/O on existing client connections.

The accept_wrapper() function is called whenever a new client connection is made. It accepts the connection, sends a flag back to the client to indicate a successful connection, sets the connection to non-blocking mode, creates a data namespace object to store connection-specific data, registers the connection with the selector object for I/O multiplexing, and sets the events to selectors.EVENT_READ | selectors.EVENT_WRITE.

The service_connection() function is called whenever there is data to be read from or written to an existing client connection. It reads any incoming data and appends it to the output buffer, or if there is data in the output buffer it sends it back to the client. If no data is received, it closes the connection and removes it from the selector object.

The script then sets up the server socket, binds it to the specified host and port, and registers it with the selector object. It enters a main loop where it waits for events to occur, and handles each event that occurred by either accepting a new client connection or servicing an existing one. It exits the loop and closes the selector object when a keyboard interrupt occurs.

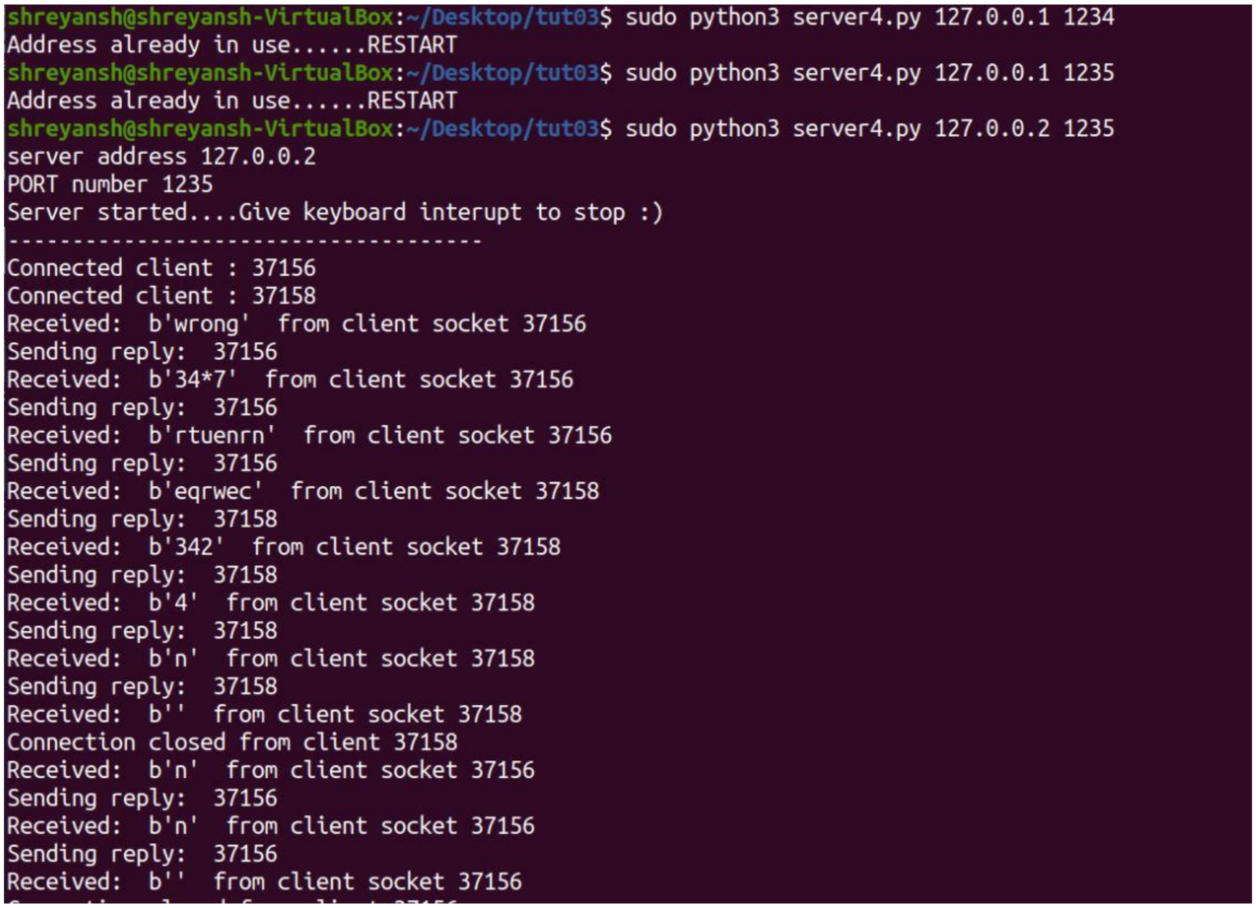
Overall, this script is a simple implementation of a non-blocking server that can handle multiple client connections simultaneously.

```
import sys

import socket

import selectors

import types


# Create a selector object

sel = selectors.DefaultSelector()


# Define a function to handle new client connections

def accept_wrapper(sock):

    # Accept a new client connection

    conn, addr = sock.accept()

    print("Connected client :", addr[1])


    # Send a flag to the client indicating that the connection was
successful
```

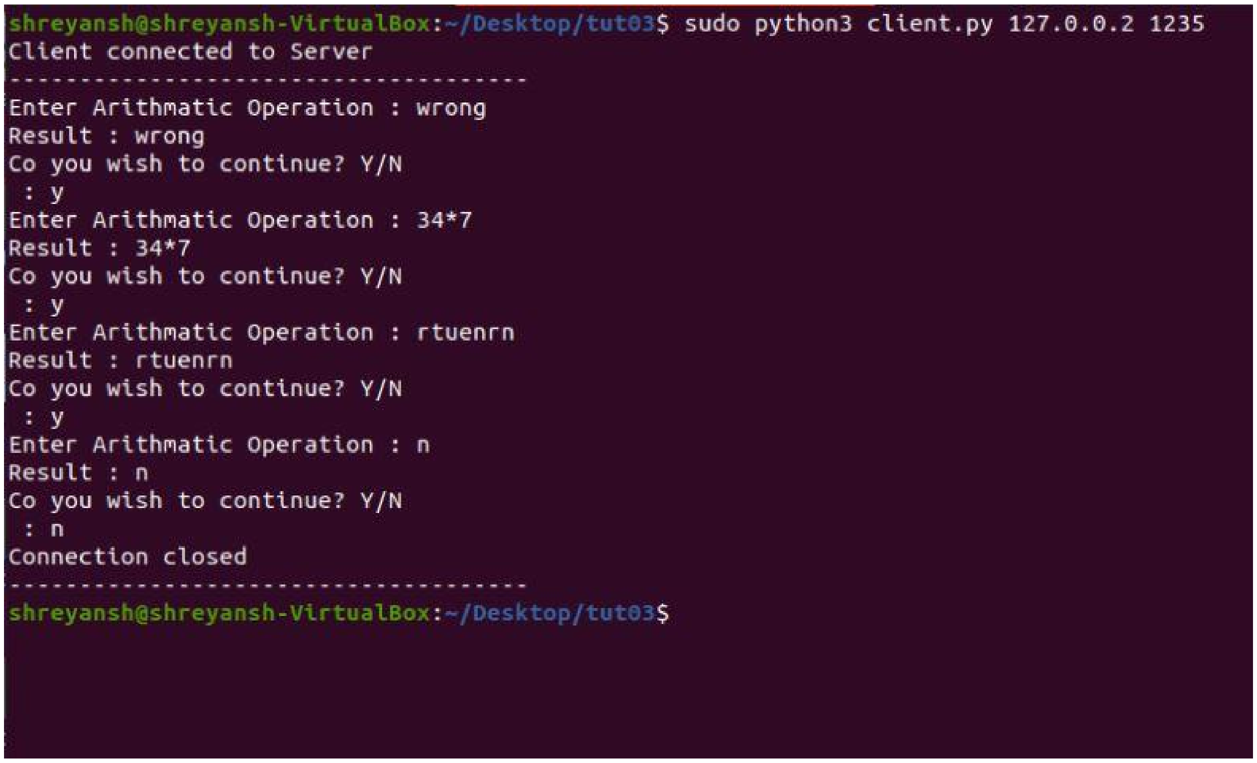
```
    flag = "1"

    conn.send(flag.encode())

    # Set the connection to non-blocking mode

    conn.setblocking(False)

    # Create a namespace object to store data related to this connection

    data = types.SimpleNamespace(addr=addr, inb=b"", outb=b"")

    # Register the connection with the selector object

    events = selectors.EVENT_READ | selectors.EVENT_WRITE

    sel.register(conn, events, data=data)

# Define a function to handle client connections

def service_connection(key, mask):

    # Get the socket object from the key

    sock = key.fileobj

    # Get the data object associated with this connection

    data = key.data

    # Handle read events

    if mask & selectors.EVENT_READ:

        # Receive data from the client

        recv_data = sock.recv(1024)

        # If data was received, append it to the output buffer for this 
connection

        if recv_data:

            data.outb += recv_data

            print("Received: ",recv_data," from client 
socket",data.addr[1])
```

```
        else:
            # If no data was received, the connection has been closed
            print("Connection closed from client", data.addr[1])
            sel.unregister(sock)
            sock.close()

    # Handle write events
    if mask & selectors.EVENT_WRITE:
        # If there is data in the output buffer for this connection, send
it back to the client
        if data.outb:
            print("Sending reply: ",data.addr[1]) #Echoing data back to
client
            sent = sock.send(data.outb)
            data.outb = data.outb[sent:]

# Get the server address and port from the command line arguments
host = sys.argv[1]
port = sys.argv[2]
port = int(port)

# Set up the server socket
try:
    lsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    lsock.bind((host, port))
    lsock.listen()
    print("server address",host)
    print("PORT number",port)
    lsock.setblocking(False)
    sel.register(lsock, selectors.EVENT_READ, data=None)
    print("Server started....Give keyboard interupt to stop :)")
    print("--------------------------------------")
```

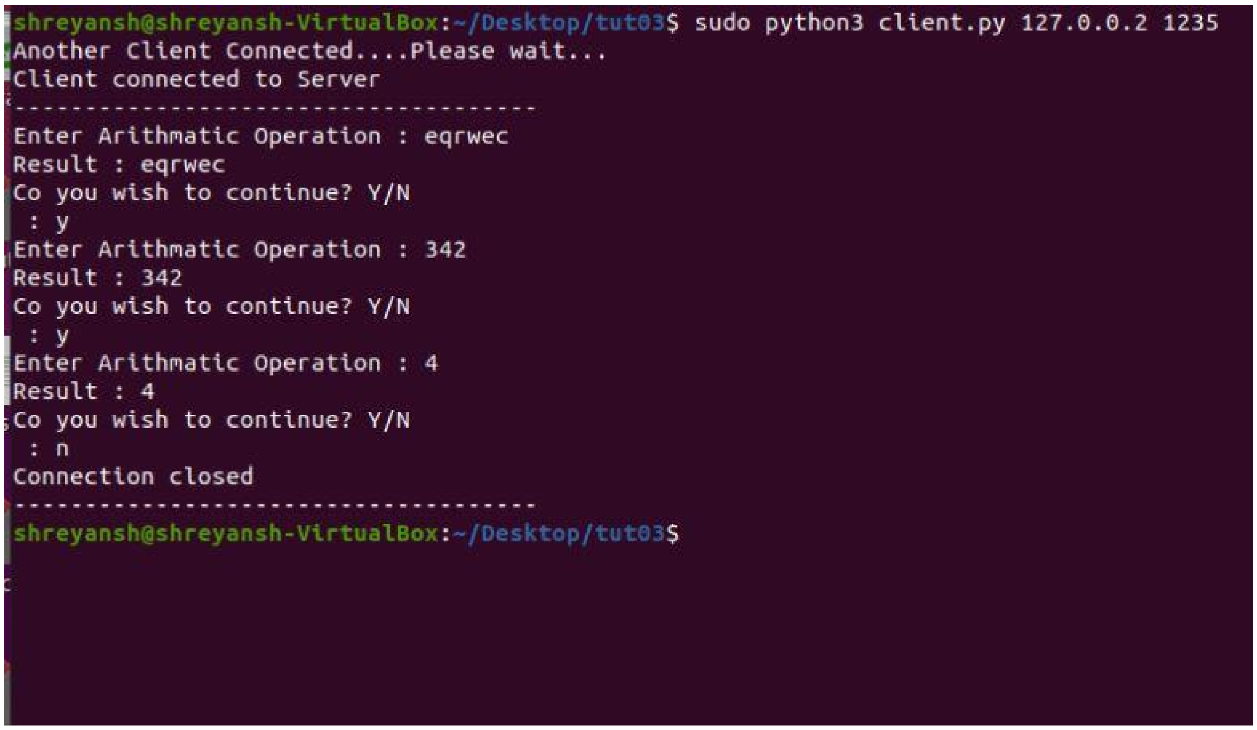
```
except:
    print("Address already in use......RESTART")
    exit()

# Main loop to handle incoming connections
try:
    while True:
        # Wait for events to occur
        events = sel.select(timeout=None)

        # Handle each event that occurred
        for key, mask in events:
            if key.data is None:
                # If there is no data associated with the key, a new
client connection is being made
                accept_wrapper(key.fileobj)
            else:
                # If there is data associated with the key, a client
connection is being serviced
                service_connection(key, mask)

# Handle a keyboard interrupt to shut down the server
except KeyboardInterrupt:
    print("Caught Keyboard Interrput. Server Closed.")

# Close the selector object
finally:
    sel.close()
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 server4.py 127.0.0.1 1234
Address already in use......RESTART
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 server4.py 127.0.0.1 1235
Address already in use......RESTART
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 server4.py 127.0.0.2 1235
server address 127.0.0.2
PORT number 1235
Server started....Give keyboard interupt to stop :)
-------------------------------------
Connected client : 37156
Connected client : 37158
Received:  b'wrong'  from client socket 37156
Sending reply:  37156
Received:  b'34*7'  from client socket 37156
Sending reply:  37156
Received:  b'rtuenrn'  from client socket 37156
Sending reply:  37156
Received:  b'eqrwec'  from client socket 37158
Sending reply:  37158
Received:  b'342'  from client socket 37158
Sending reply:  37158
Received:  b'4'  from client socket 37158
Sending reply:  37158
Received:  b'n'  from client socket 37158
Sending reply:  37158
Received:  b''  from client socket 37158
Connection closed from client 37158
Received:  b'n'  from client socket 37156
Sending reply:  37156
Received:  b'n'  from client socket 37156
Sending reply:  37156
Received:  b''  from client socket 37156
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.2 1235
Client connected to Server
-------------------------------------
Enter Arithmatic Operation : wrong
Result : wrong
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 34*7
Result : 34*7
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : rtuenrn
Result : rtuenrn
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : n
Result : n
Co you wish to continue? Y/N
 : n
Connection closed
-------------------------------------
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$
```



```
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$ sudo python3 client.py 127.0.0.2 1235
Another Client Connected....Please wait...
Client connected to Server
-------------------------------------
Enter Arithmatic Operation : eqrwec
Result : eqrwec
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 342
Result : 342
Co you wish to continue? Y/N
 : y
Enter Arithmatic Operation : 4
Result : 4
Co you wish to continue? Y/N
 : n
Connection closed
-------------------------------------
shreyansh@shreyansh-VirtualBox:~/Desktop/tut03$
```