Assignment 8

Title:Write a Socket Program in Python to connect to www.google.co.in and print “the socket has successfully connected to Google”

Objective:To create a Python socket program that establishes a connection to www.google.co.in and prints a success message upon successful connection.

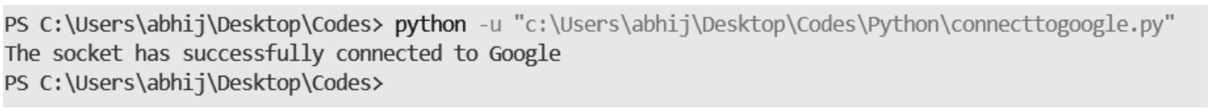
Brief Description: The Python socket program establishes a TCP connection to www.google.co.in, using its IP address and port number. It creates a socket object and connects to the specified host and port. Upon successful connection, it prints a confirmation message. This demonstrates the basic functionality of socket programming in Python. The program showcases the ability to establish connections with remote servers. Through this implementation, users gain insight into network communication fundamentals. It highlights Python's simplicity in handling socket operations for establishing connections.

Program

```
import socket

def main():
    host = 'www.google.co.in'
    port = 80
    try:
            client_socket =
socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        client_socket.connect((host, port))
        print("The socket has successfully connected to Google")
            client_socket.close()
    except Exception as e:
        print("An error occurred:", e)
if __name__ == "__main__":
    main()
```

Output



```
PS C:\Users\abhij\Desktop\Codes> python -u "c:\Users\abhij\Desktop\Codes\Python\connecttogoogle.py"
The socket has successfully connected to Google
PS C:\Users\abhij\Desktop\Codes>
```

Conclusion:

The Python socket program successfully establishes a TCP connection to www.google.co.in and prints "the socket has successfully connected to Google", confirming the successful connection.

Assignment 9

Title: Write Socket Programs in Python to implement a simple TCP Client Server application and send “Hello” messages.

Objective:
Develop Python socket programs to create a basic TCP client-server application capable of exchanging "Hello" messages.

Brief Description:
The TCP server script listens for incoming connections on a specified port. Upon connection from a client, it receives a "Hello" message and responds with its own "Hello" message. The TCP client script connects to the server and sends a "Hello" message, then waits for a response.

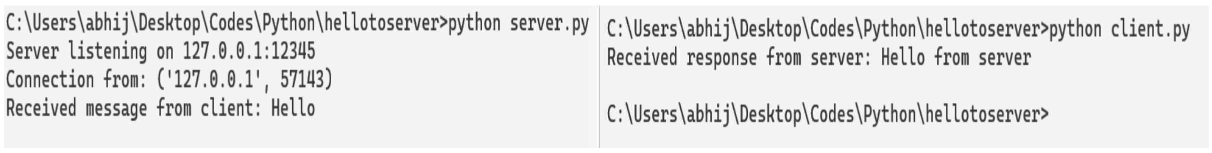
This implementation demonstrates the fundamental aspects of TCP socket programming in Python, including server setup, client connection, message exchange, and response handling. It serves as a foundational example for building more complex network applications using sockets in Python.

```
import socket
```

Server code

```
def main():
    host = '127.0.0.1'  # localhost
    port = 12345  # port to listen on
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind((host, port))
      server_socket.listen(5)
    print("Server listening on {}:{}".format(host, port))
    while True:
            client_socket, client_address = server_socket.accept()
        try:
            print("Connection from:", client_address)
            data = client_socket.recv(1024).decode('utf-8')
            print("Received message from client:", data)
                    response = "Hello from server"
            client_socket.sendall(response.encode('utf-8'))
        finally:
                    client_socket.close()
if __name__ == "__main__":
    main()
```
Client code
```
import socket
def main():
    server_host = '127.0.0.1'  # Server's IP address
    server_port = 12345  # Port the server is listening on
       client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    try:
               client_socket.connect((server_host, server_port))

               message = "Hello"
        client_socket.sendall(message.encode('utf-8'))
               response = client_socket.recv(1024).decode('utf-8')
        print("Received response from server:", response)
    finally:
                client_socket.close()
if __name__ == "__main__":
    main()
```
Output



```
C:\Users\abhij\Desktop\Codes\Python\hellotoserver>python server.py
Server listening on 127.0.0.1:12345
Connection from: ('127.0.0.1', 57143)
Received message from client: Hello
```

```
C:\Users\abhij\Desktop\Codes\Python\hellotoserver>python client.py
Received response from server: Hello from server

C:\Users\abhij\Desktop\Codes\Python\hellotoserver>
```

Conclusion:

The Python TCP client-server application successfully establishes a connection and exchanges "Hello" messages, showcasing the basic functionality of TCP socket programming in Python. This serves as a starting point for developing more sophisticated network applications.

Assignment 10

TITLE: Write Socket Programs in Python to implement a simple UDP Client Server application and send “Welcome” messages

Objective: Develop Python socket programs to create a basic UDP client-server application capable of exchanging "Welcome" messages.

Brief Description: The UDP server listens on a specified port for incoming messages from clients. Upon receiving a message, it prints the message and sends a "Welcome" response back to the client. The UDP client sends a "Welcome" message to the server and waits for a response. Upon receiving the response, it prints it to the console.

This implementation showcases the basic functionality of UDP socket programming in Python, enabling communication between a UDP server and client using datagrams.

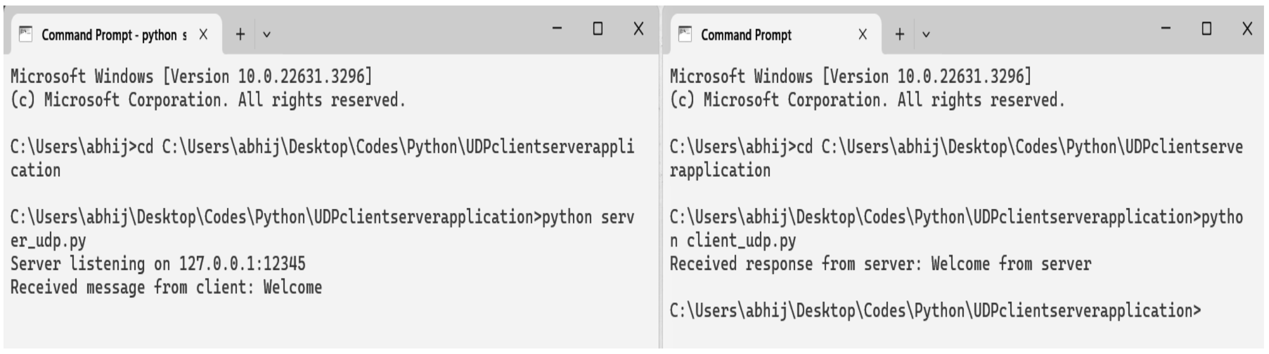
Server code

```
import socket
def main():
    host = '127.0.0.1'
    port = 12345
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
      server_socket.bind((host, port))
    print("Server listening on {}:{}".format(host, port))
    while True:
        data, client_address = server_socket.recvfrom(1024)
        print("Received message from client:", data.decode('utf-8'))
        response = "Welcome from server"
        server_socket.sendto(response.encode('utf-8'),
client_address)
if __name__ == "__main__":
    main()
```

Client code

```
import socket
def main():
    server_host = '127.0.0.1'  # Server's IP address
    server_port = 12345  # Port the server is listening on
       client_socket = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    try:
               message = "Welcome"
        client_socket.sendto(message.encode('utf-8'), (server_host,
server_port))
               response, server_address = client_socket.recvfrom(1024)
        print("Received response from server:", response.decode('utf-8'))
    finally:
             client_socket.close()
if __name__ == "__main__":
  main()
```

Output



```
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\abhij>cd C:\Users\abhij\Desktop\Codes\Python\UDPclientserverappli
cation

C:\Users\abhij\Desktop\Codes\Python\UDPclientserverapplication>python serv
er_udp.py
Server listening on 127.0.0.1:12345
Received message from client: Welcome
```

```
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\abhij>cd C:\Users\abhij\Desktop\Codes\Python\UDPclientserve
rapplication

C:\Users\abhij\Desktop\Codes\Python\UDPclientserverapplication>pytho
n client_udp.py
Received response from server: Welcome from server

C:\Users\abhij\Desktop\Codes\Python\UDPclientserverapplication>
```

Conclusion:

The Python UDP client-server application successfully exchanges "Welcome" messages, demonstrating the basic functionality of UDP socket programming for communication between processes. This provides a foundation for building more complex network applications using UDP protocols.

Assignment 11

Title: Write Client Server Socket Programs to implement file transfer using TCP/IPv4.

Objective:

To develop TCP/IPv4 client-server socket programs in Python for transferring files between systems.

Brief Description:

The TCP server listens for incoming connections and receives files from clients. Upon connection, it receives the file data, writes it to a file on the server's system, and sends a confirmation back to the client. The TCP client connects to the server, reads a file from its system, sends it to the server, and waits for confirmation.

<u>Server code</u>

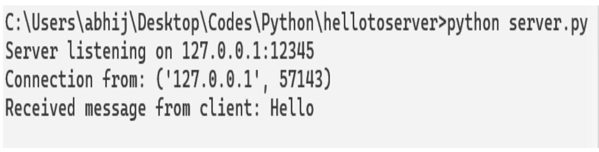
```
import socket
def main():
    host = '127.0.0.1'  # localhost
    port = 12345  # port to listen on
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(5)
    print("Server listening on {}:{}".format(host, port))
    while True:
        client_socket, client_address = server_socket.accept()
        try:
            print("Connection from:", client_address)
            file_name = client_socket.recv(1024).decode('utf-8')
            file_data = client_socket.recv(1024)
            with open(file_name, 'wb') as file:
                file.write(file_data)
            client_socket.sendall("File received by server".encode('utf-8'))
        finally:
            client_socket.close()
```

```
if __name__ == "__main__":
    main()
```

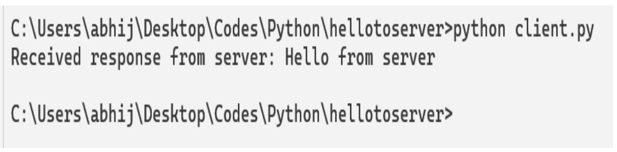
Client Code

```
import socket
def main():
    server_host = '127.0.0.1'  # Server's IP address
    server_port = 12345  # Port the server is listening on
    file_name = 'example.txt'  # Name of the file to be transferred
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client_socket.connect((server_host, server_port))
        client_socket.sendall(file_name.encode('utf-8'))
        with open(file_name, 'rb') as file:
            file_data = file.read()
            client_socket.sendall(file_data)
        confirmation = client_socket.recv(1024).decode('utf-8')
        print("Server confirmation:", confirmation)
    finally:
        client_socket.close()
if __name__ == "__main__":
    main()
```

Output



```
C:\Users\abhij\Desktop\Codes\Python\hellotoserver>python server.py
Server listening on 127.0.0.1:12345
Connection from: ('127.0.0.1', 57143)
Received message from client: Hello
```



```
C:\Users\abhij\Desktop\Codes\Python\hellotoserver>python client.py
Received response from server: Hello from server

C:\Users\abhij\Desktop\Codes\Python\hellotoserver>
```

Conclusion:

The TCP client-server socket programs effectively enable the transfer of files between systems using TCP/IPv4, showcasing the versatility of TCP sockets in facilitating reliable data transmission.

Assignment 12

TITLE: Build a simple chat room using Python socket programming and allow multiple clients to connect and transfer messages.

Objective:

To develop a Python socket program that creates a basic chat room, enabling multiple clients to connect and exchange messages.

Brief Description:

The Python server script listens for incoming connections from multiple clients and manages the communication between them. Upon connection, clients can send messages to the server, which then broadcasts them to all connected clients. Each client can receive messages from other clients via the server and send their own messages, creating a simple chat room environment.

# Server Code

```
import socket
import threading

def handle_client(client_socket, client_address, clients):
    print("Connection from:", client_address)
    while True:
        try:
            message = client_socket.recv(1024).decode('utf-8')
            if not message:
                break
            for client in clients:
                if client != client_socket:
                    client.sendall(message.encode('utf-8'))
        except Exception as e:
            print("Error:", e)
            break
    clients.remove(client_socket)
    client_socket.close()
    print("Disconnected:", client_address)
```

```
def main():
    host = '127.0.0.1'
    port = 12345
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(5)
    print("Server listening on {}:{}".format(host, port))
    clients = []
    while True:
        client_socket, client_address = server_socket.accept()
        clients.append(client_socket)
        client_thread = threading.Thread(target=handle_client,
args=(client_socket, client_address, clients))
        client_thread.start()

if __name__ == "__main__":
    main()
```

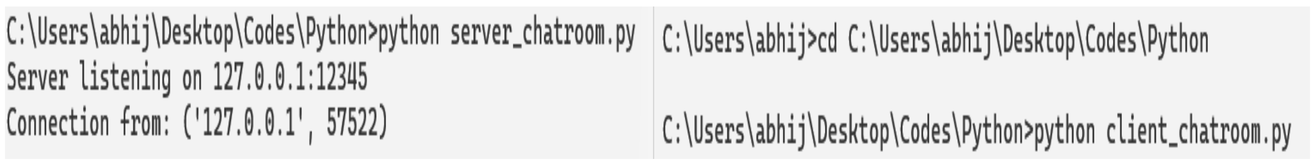
# Client Code

```
import socket
import threading

def receive_messages(client_socket):
    while True:
        try:
            message = client_socket.recv(1024).decode('utf-8')
            if not message:
                break
            print(message)
        except Exception as e:
            print("Error:", e)
            break

def main():
    server_host = '127.0.0.1'
    server_port = 12345
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client_socket.connect((server_host, server_port))
        receive_thread = threading.Thread(target=receive_messages, args=(client_socket,))
        receive_thread.start()
        while True:
            message = input()
            client_socket.sendall(message.encode('utf-8'))
    except Exception as e:
        print("Error:", e)
    finally:
        client_socket.close()
if __name__ == "__main__":
    main()
```

Output



```
C:\Users\abhij\Desktop\Codes\Python>python server_chatroom.py
Server listening on 127.0.0.1:12345
Connection from: ('127.0.0.1', 57522)
```

```
C:\Users\abhij>cd C:\Users\abhij\Desktop\Codes\Python

C:\Users\abhij\Desktop\Codes\Python>python client_chatroom.py
```

Conclusion:

The Python chat room implementation using socket programming successfully allows multiple clients to connect and exchange messages, demonstrating the capabilities of sockets for facilitating real-time communication between clients and server.