

**Faculty of Engineering and Technology**

**Department of Electrical and Computer Engineering**

**ENCS436**

**COMPUTER NETWORKS**

**Project #1 Report**

**Socket Programming**

**Prepared by:**

**Haitham Daana 1121331**

**Mohammad Ibrahim Mousa 1141618**

**Section #3**

**Intructor: Dr. Iyad Tumar**

**5/12/2017**

# **Abstract**

This report represents simple socket program, it’s a simple client-server program that works on both of TCP/UDP using Python Language. The server when it receives a message/text, it replies by reversing all characters, and reverses all the capitalization of the strings.

Contents

[Abstract 2](#_Toc500259942)

[Introduction 4](#_Toc500259943)

[Socket in Python 5](#_Toc500259944)

[Methodology of Design 6](#_Toc500259945)

[Results and Discussion 7](#_Toc500259946)

[User Datagram Protocol (UDP) 7](#_Toc500259947)

[Transmission Control Protocol (TCP) 10](#_Toc500259948)

[Conclusion 13](#_Toc500259949)

[References 14](#_Toc500259950)

# Introduction

network socket is an endpoint of an inter-process communication flow across a computer network. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents. Today, most communication between computers is based on the internet protocol; therefore, most network sockets are internet sockets. To create a connection between machines.

# Socket in Python

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols. Python also has libraries that provide higher level access to specific application level network protocols, such as FTP, HTTP, SMTP, and so on.

Sockets may be implemented over a number of different channel types: UNIX domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

# Methodology of Design

As mentioned earlier, Python has many libraries that can be used to provide interfacing between network connections. For example, ***socket.socket()*** is used to open a socket: It has the following syntax:

|  |  |
| --- | --- |
| ***s.bind()*** | This function binds address (hostname, port number pair) to socket. |
| ***s.listen()*** | This method sets up and start TCP listener. |
| ***s.accept()*** | This passively accept TCP client connection, waiting until connection arrives (blocking). |
| ***s.connect()*** | This function actively initiates TCP server connection. |
| ***s.recv()*** | This method receives TCP message |
| ***s.send()*** | This method transmits TCP message |
| ***s.recvfrom()*** | This method receives UDP message |
| ***s.sendto()*** | This method transmits UDP message |
| ***s.close()*** | This method closes socket |
| ***socket.gethostname()*** | Returns the hostname |

The program, was created using two types of connections. One using TCP and the other was UDP. The server listens to the clients, waiting for a connection. In case of TCP there is a handshaking process that takes place, before exchanging data, but in case of UDP there is an open connection between host and client, it’s much faster. Then It takes the message from the client, and reverses all characters, and the capitalization of the text string sent, and sends it back to the client.

# Results and Discussion

## User Datagram Protocol (UDP)

UDP is a simple transport protocol that extends the host-to-host delivery of packets of the underlying network into a process-to-process communication. Since there are many processes running on a given host (e.g. multiple Internet browsers), UDP needs to add a level of demultiplexing, allowing multiple application processes on each host to share the network. Therefore, the only interesting issue in UDP is the form of address used to identify a process. Although it is possible to directly identify a process with the operating system (OS) assigned id (pid), such an approach is only practical in a close distributed system with one OS that assigns unique ids to all processes (does not work for the entire world!). Instead, a process is indirectly identified using an abstract locator, often called a port. A source process sends a message to a port, and a destination process receives a message from a port. The UDP header contains a 16 bit port number for both the sender (source) and the receiver (destination).

We see TCP Flow in Figer.2

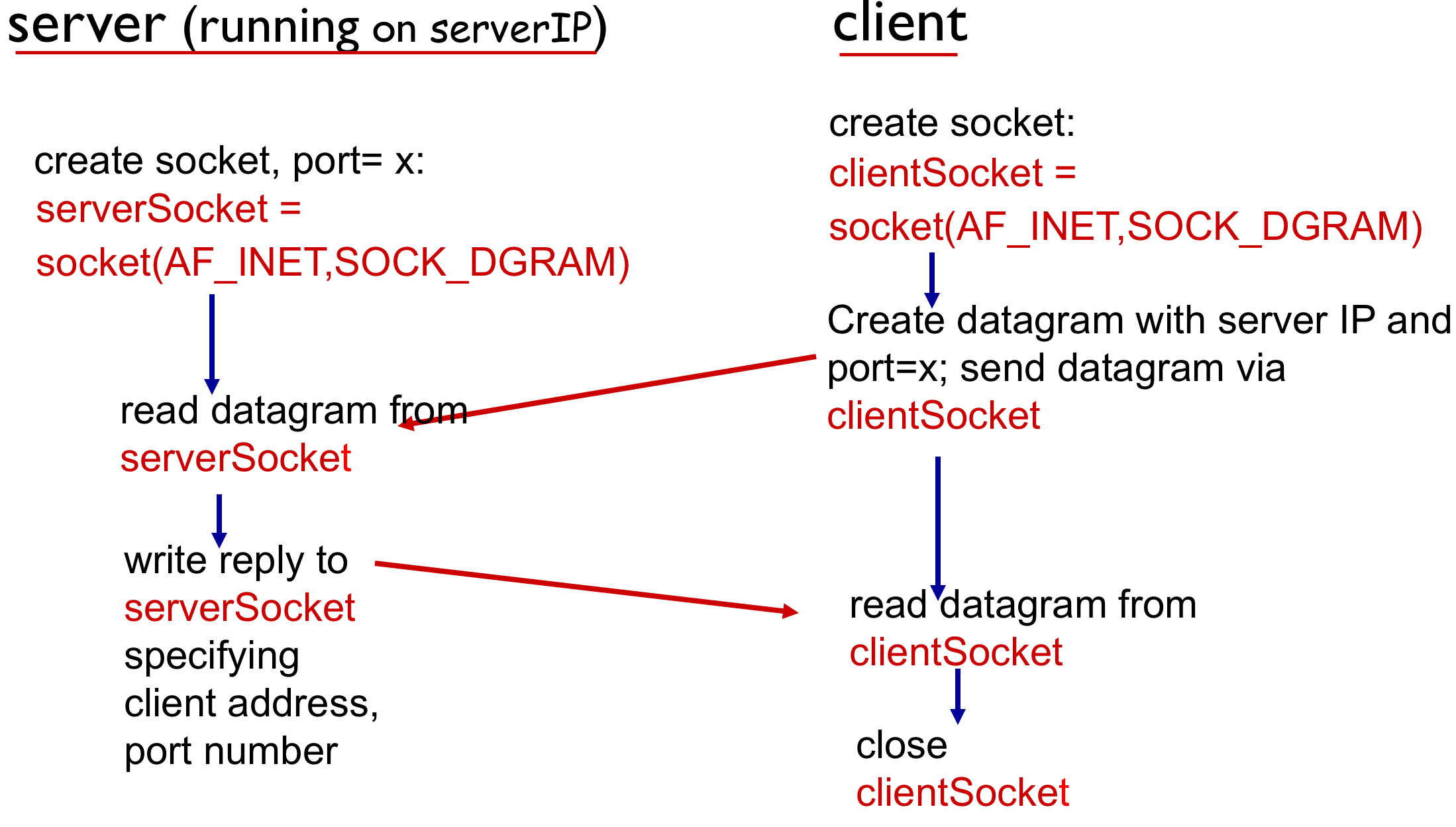
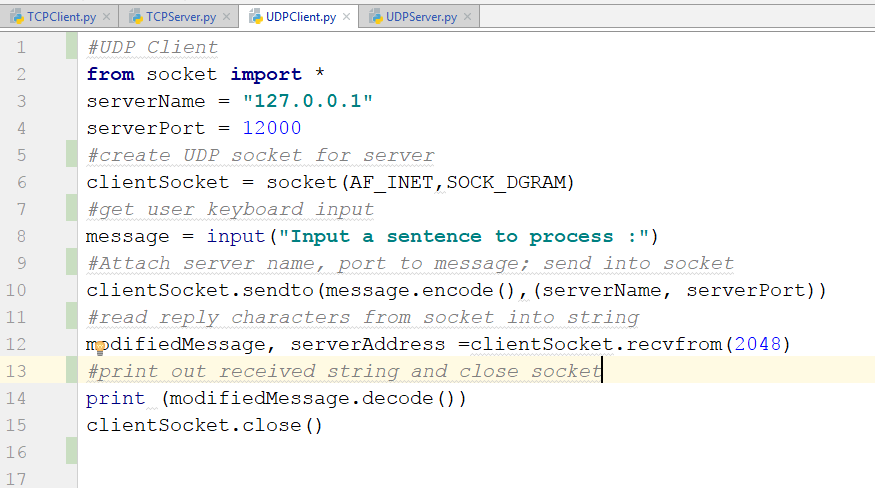
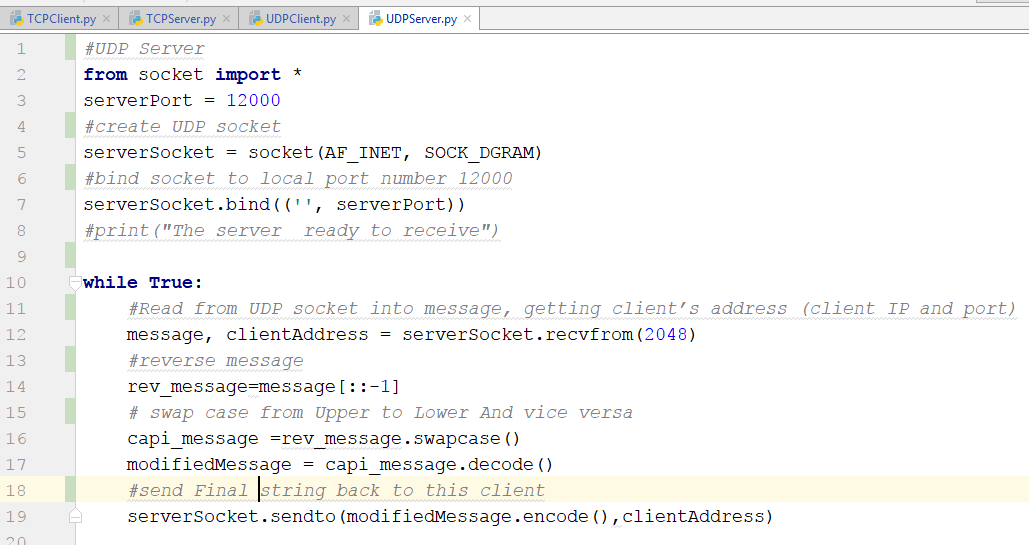


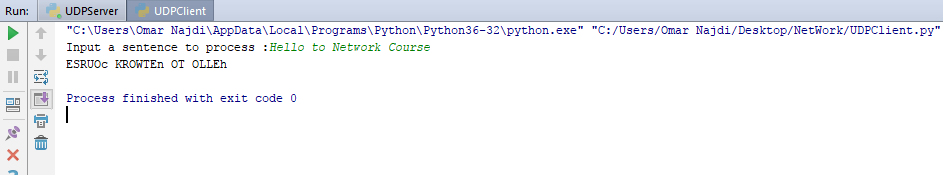
Figure.2 (Client/server socket interaction: UDP)

Code for UDP Client and Server





Output



## Transmission Control Protocol (TCP)

In contrast to a simple demultiplexing protocol like UDP, a more sophisticated transport protocol in one that offers reliable communication. The Internet’s Transmission Control Protocol (TCP) is probably the most widely used protocol of this type. As a transport protocol, TCP provides reliable, in order delivery of messages. It is a full duplex protocol, meaning that each TCP connection supports a pair of streams, one flowing in each direction. It also includes a flow control mechanism for each of these streams that allows the receiver (on both ends) to limit how much data the sender can transmit at a given time (we will look at this feature later). Of course, TCP supports the demultiplexing mechanism of UDP to allow multiple application programs on a given host to simultaneously communicate over the Internet. However, the demultiplexing key used by TCP is the 4-tuple "source port, source host, destination port, destination host " to identify the particular TCP connection.

At the heart of TCP is the sliding window algorithm. Even though this is the same basic algorithm we have seen before for DLC, because TCP runs over the network rather than a single link, there are many important differences that complicate TCP

We see TCP Flow in Figer.1

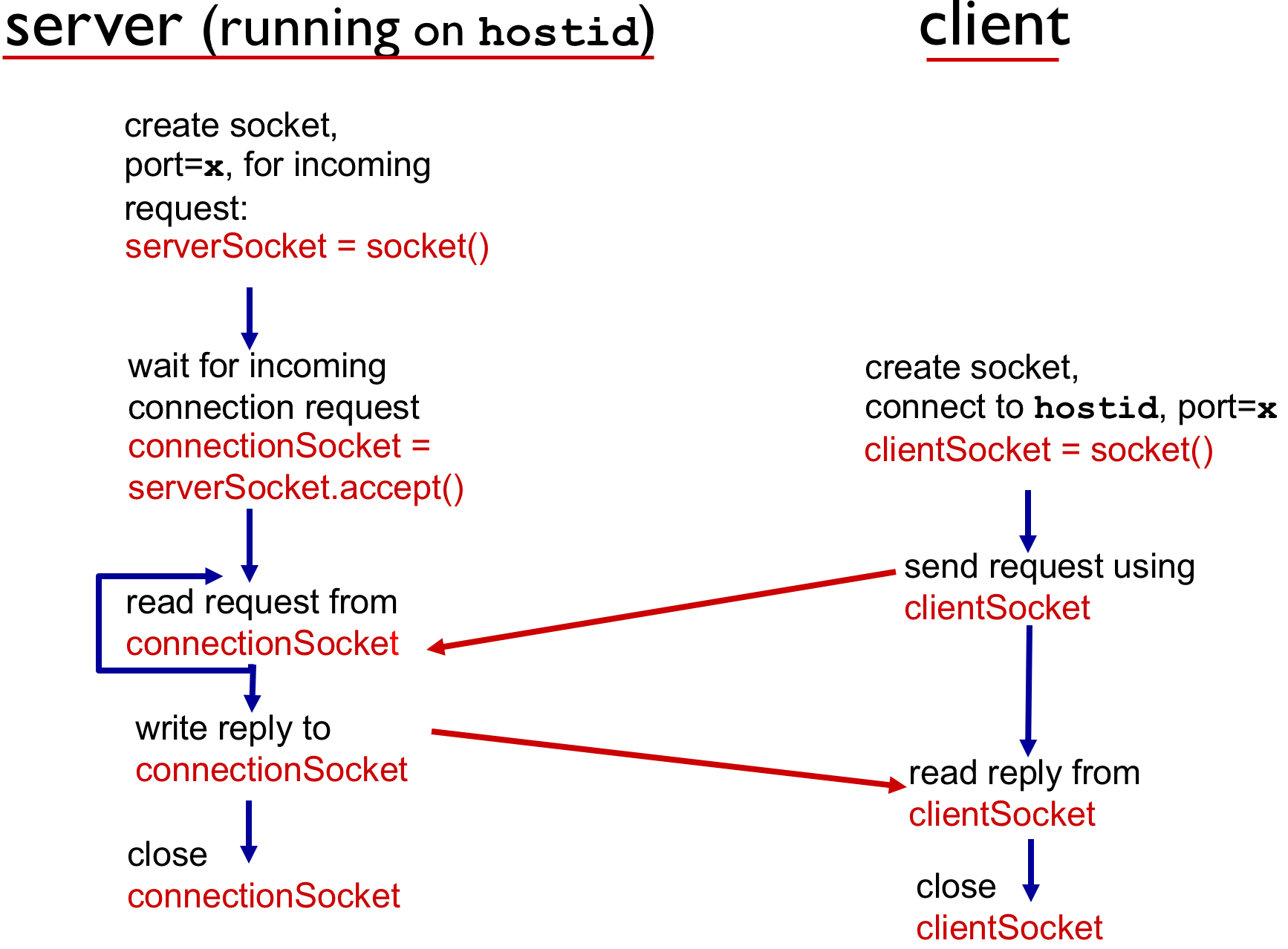
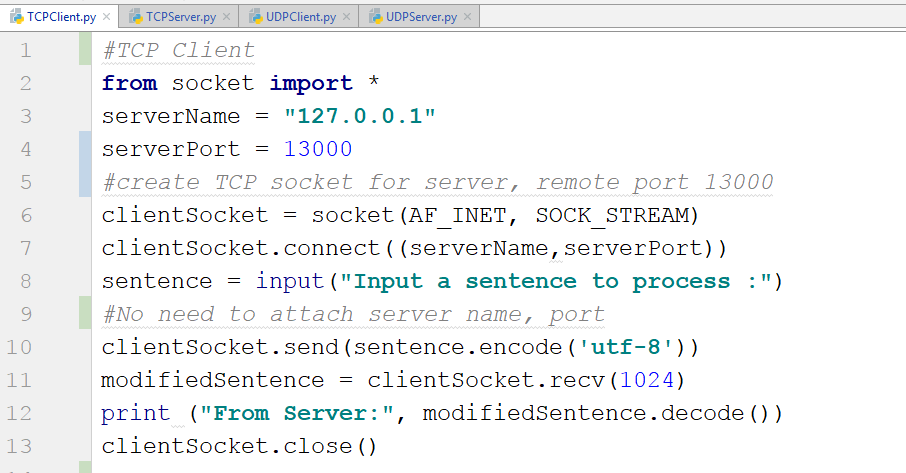
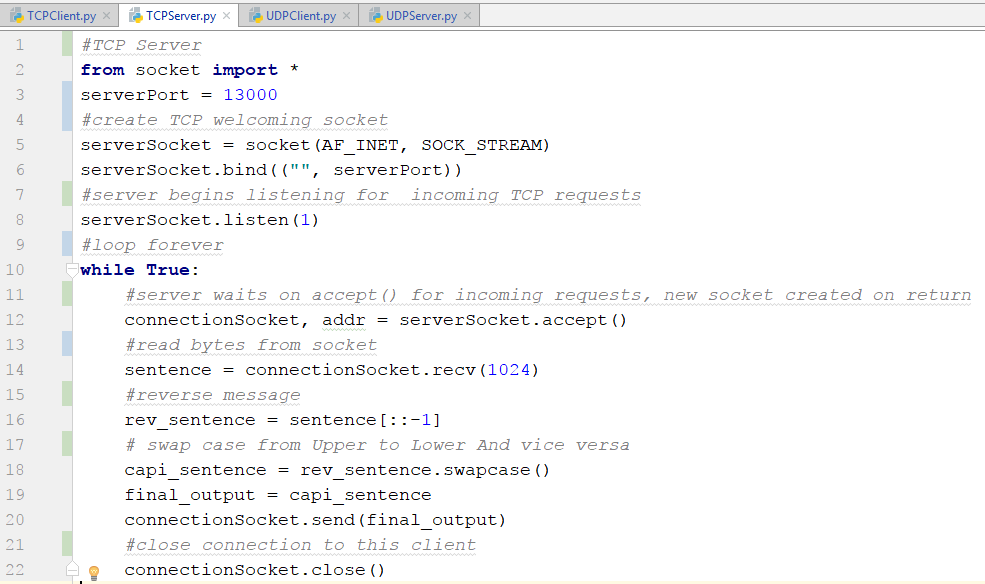


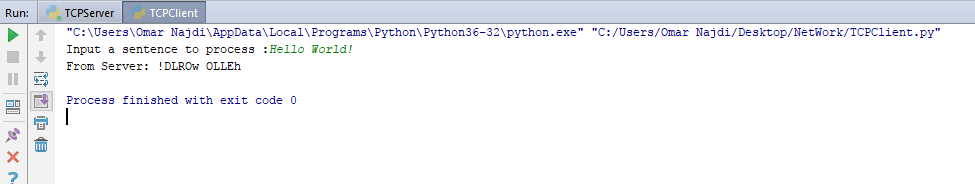
Figure.1 (Client/server socket interaction: TCP)

Code for TCP Client and Server





Output



# Conclusion

By the end of this mini-project, we were able to code in python for first time, understood the basic programming skills needed for socket programming and comprehended the main concepts of networking in that matter. Which will help us in upper Networking courses, or in programming applications that work online.

# References

[1]: Jim Kurose and Keith Ross***. Computer Networking: A Top Down Approach****, 7th*Edition, Addison-Wesley, 2013.

[2]:<http://www.biogem.org/downloads/notes/Socket%20Programming%20in%20Python.pdf>

[3]:<https://pdfs.semanticscholar.org/18f2/8c21f0b94d072c9727f670357b26a43ff735.pdf>