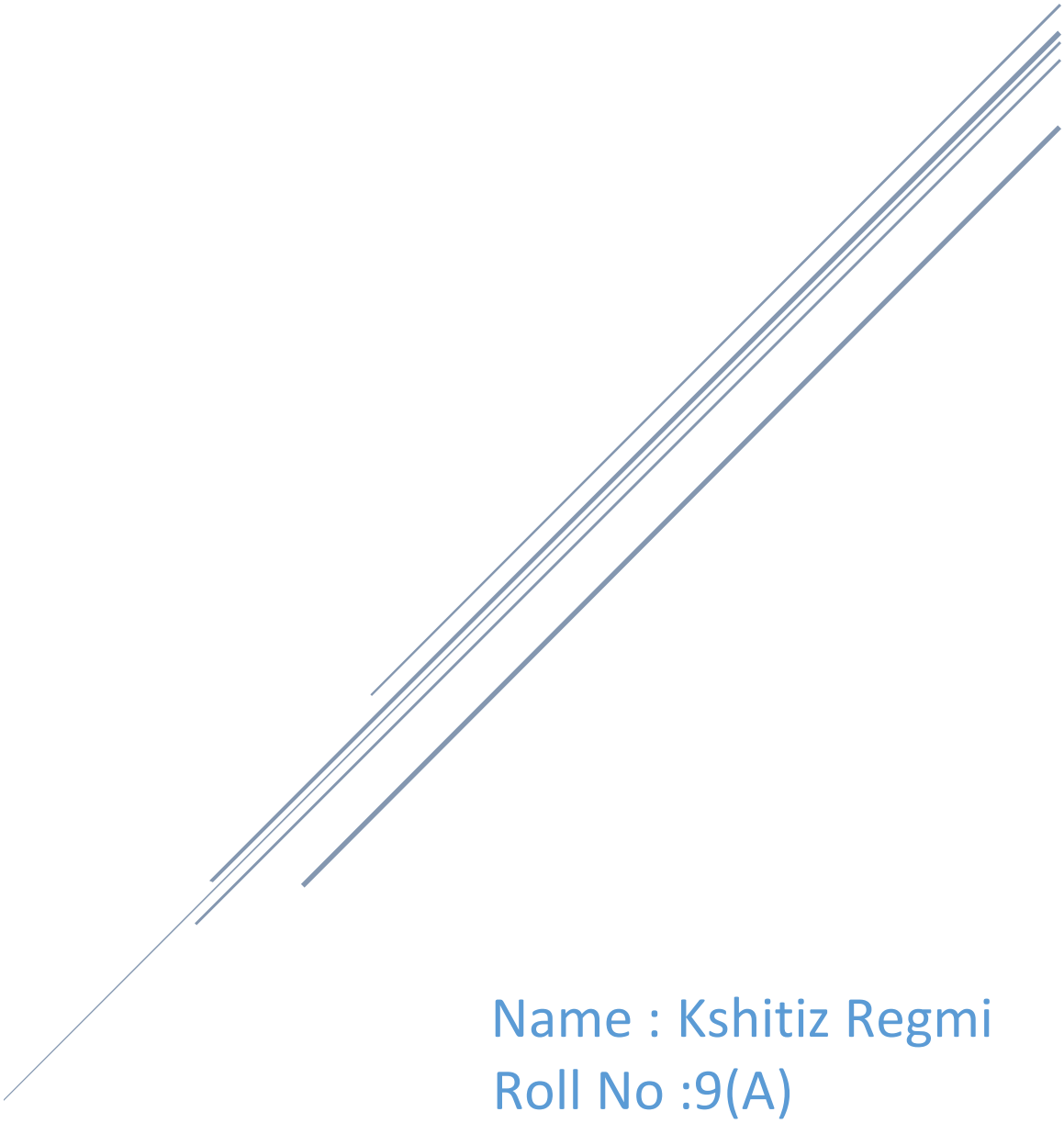
NETWORK SOCKET PROGRAMMING WITH PYTHON

Madan Bhandari Memorial College



Name : Kshitiz Regmi
Roll No :9(A)
5th Semester

Contents

Socket Basics

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Socket programming

Socket programming is one of the most fundamental technologies of computer network programming .It 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.

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.

Vocabulary of Sockets

Term	Description
domain	The family of protocols that will be used as the transport mechanism. These values are constants such as **`AF_INET`**, **`PF_INET`**, **`PF_UNIX`**, **`PF_X25`**, and so on.
type	The type of communications between the two endpoints, typically **`SOCK_STREAM`** for connection-oriented protocols and **`SOCK_DGRAM`** for connectionless protocols.
protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type.
hostname	The identifier of a network interface: • A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation • A string "**`<broadcast>`**", which specifies an **`INADDR_BROADCAST`** address. • A zero-length string, which specifies **`INADDR_ANY`**, or • An Integer, interpreted as a binary address in host byte order.
port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

The socket Module

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.

Server Socket Methods

Method	Description
s.bind()	This method 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).

Client Socket Methods

Method	Description
s.connect()	This method actively initiates TCP server connection.

General Socket Methods

Method	Description
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.

A Simple Server

To write Internet servers, we use the socket function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server. Now call bind(hostname, port) function to specify a port for your service on the given host. Next, call the accept method of the returned object. This method waits until a client connects to the port you specified, and then returns a connection object that represents the connection to that client as:

- First of all we import socket which is necessary.
- Then we made a socket object and reserved a port 65432 on our pc.
- we would have passed 127.0.0.1 then it would have listened to only those calls made within the local computer.
- After that we put the server into listen mode.
- At last we make a while loop and start to accept all in coming connections and close those connections after a blank message to all connected sockets.
- Now save this file as server.ipynb and run it.

```
import socket

HOST = '127.0.0.1'  # Standard loopback interface address (localhost)
PORT = 65432        # Port to listen on (non-privileged ports are > 1023)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()


conn, addr = s.accept()
x = True


while x:
    indata = conn.recv(1024)
    print("Recived from client: " ,indata.decode())
    dat = input("Enter a message to send: ")
    if dat == "":
        x = False
    else:
        conn.sendall(bytes(dat,'UTF-8'))
```

A Simple Client

Let us write a very simple client program which opens a connection to a given port 80 and given host. This is very simple to create a socket client using Python's socket module function. The socket. connect (hostname, port) opens a TCP connection to hostname on the port. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits −

- First of all we make a socket object.
- Then we connect to local host 127.0.0.1 on port 65432 (the port on which our server runs)
- At last we make a while loop and start to accept all in coming connections and close those connections after a blank message to all connected sockets.
- Now save this file as client.ipynb and run it after starting the server script.

```
import socket

HOST = '127.0.0.1'  # The server's hostname or IP address
PORT = 65432        # The port used by the server
x = True
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))


while(x):

    dat = input("Enter a message to send: ")
    if dat == "":
        x = False
    else:
        s.sendall(bytes(dat,'UTF-8'))
        data = s.recv(1024)
        print('Received from server:', repr(data))
```

Output

Step 1: Run server.py. It would start a server in background.

Step 2: Run client.py. Once server is started run client.

Step 3: Output of server.py generates as follows:

```
Recived from client:  Hello I am Client.
Enter a message to send: Hey! Buddy I am Server.
Recived from client:  Connection Success.
Enter a message to send: Yeh! I can Talk .
```

Step 4: Output of client.py generates as follows:

```
Enter a message to send: Hello I am Client.
Received from server: b'Hey! Buddy I am Server.'
Enter a message to send: Connection Success.
Received from server: b'Yeh! I can Talk .'
Enter a message to send:
```

Discussion and Conclusion

To create a connection between machines, Python programs import the socket module, create a

socket object, and call the object's methods to establish connections and send and receive data. Run client script from the terminal after starting the server script.