11. Write a simple web server that can return a single line/multiple line of text to any connected web browser.

Server.py

```
import socket

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

s.bind(('127.0.0.1', 12345))

s.listen(1)

while True:

    connection, client_address = s.accept()

    msg = 'Thank you, you are now connected.'

    connection.send(msg.encode())

    connection.close()
```

Output:

```
Thank you, you are now connected.
```

12. Write an efficient chart server that can handle several hundred or a large number of client connections. The chart server initializes with a few data attributes. It stores the count of clients, map of each client, and output sockets. The chart client initializes with a name argument and sends this name to the chart server.

Server.py

```
import socket
import select
import sys
from _thread import *
```

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
IP_address = '127.0.0.1'
Port = 13457
server.bind((IP_address, Port))
server.listen(100)
list_of_clients = []
def clientthread(conn, addr):
        conn.send("Welcome to this chatroom!".encode())
        while True:
                        try:
                                message = conn.recv(2048)
                                if message:
                                        print("<", addr[0], "> ", message)
                                        message_to_send = "<" + addr[0] + "> " + message
                                        broadcast(message_to_send, conn)
                                else:
                                        remove(conn)
                        except:
                                continue
def broadcast(message, connection):
        for clients in list_of_clients:
                if clients!=connection:
                        try:
                                clients.send(message.encode())
                        except:
                                clients.close()
                                remove(clients)
def remove(connection):
        if connection in list_of_clients:
                list_of_clients.remove(connection)
while True:
        conn, addr = server.accept()
        list_of_clients.append(conn)
        print(addr[0], " connected")
        start_new_thread(clientthread,(conn,addr))
conn.close()

server.close()
```

Client.py

```
import socket

import select

import sys

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

IP_address = '127.0.0.1'

Port = 13457

server.connect((IP_address, Port))

while True:

        sockets_list = [sys.stdin, server]

        read_sockets,write_socket, error_socket = select.select(sockets_list,[],[])

        for socks in read_sockets:

                if socks == server:

                        message = socks.recv(2048)

                        print(message)

                else:

                        message = sys.stdin.readline()

                        server.send(message.encode())

                        sys.stdout.write("<You>")

                        sys.stdout.write(message)

                        sys.stdout.flush()

server.close()
```

13 . Write program for local port forwarder, that will redirect all traffic from a local port to a particular remote host?

```
import argparse
```

```
LOCAL_SERVER_HOST = 'localhost'
REMOTE_SERVER_HOST = 'www.nitrkl.ac.in'
BUFSIZE = 4096
import asyncore
import socket
class PortForwarder(asyncore.dispatcher):
    def __init__(self, ip, port, remoteip, remoteport, backlog=5):
        asyncore.dispatcher.__init__(self)
        self.remoteip=remoteip
        self.remoteport=remoteport
        self.create_socket(socket.AF_INET,socket.SOCK_STREAM)
        self.set_reuse_addr()
        self.bind((ip,port))
        self.listen(backlog)
    def handle_accept(self):
        conn, addr = self.accept()
        print ("Connected to :" , addr)
        Sender(Receiver(conn),self.remoteip,self.remoteport)
class Receiver(asyncore.dispatcher):
    def __init__(self,conn):
       asyncore.dispatcher.__init__(self,conn)
       self.from_remote_buffer=''
       self.to_remote_buffer=''
       self.sender=None
    def handle_connect(self):
        pass
    def handle_read(self):
        read = self.recv(BUFSIZE)
```

```
        self.from_remote_buffer += read.decode()
    def writable(self):
        return (len(self.to_remote_buffer) > 0)
    def handle_write(self):
        sent = self.send(self.to_remote_buffer.encode())
        self.to_remote_buffer = self.to_remote_buffer[sent:]
    def handle_close(self):
        self.close()
        if self.sender:
            self.sender.close()
class Sender(asyncore.dispatcher):
    def __init__(self, receiver, remoteaddr, remoteport):
        asyncore.dispatcher.__init__(self)
        self.receiver=receiver
        receiver.sender=self
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connect((remoteaddr,remoteport))
    def handle_connect(self):
        pass
    def handle_read(self):
        read = self.recv(BUFSIZE)
        self.receiver.to_remote_buffer += read.decode()
    def writable(self):
        return (len(self.receiver.from_remote_buffer) > 0)
    def handle_write(self):
        sent = self.send(self.receiver.from_remote_buffer.encode())
        self.receiver.from__remote_buffer=self.receiver.from_remote_buffer[sent:]
    def handle_close(self):
```

```
        self.close()

        self.receiver.close()

if __name__=="__main__":

    try:

        parser = argparse.ArgumentParser(description='Stackless Socket Server Example')

        parser.add_argument('--local-host', action="store", dest="local_host",
default=LOCAL_SERVER_HOST)

        parser.add_argument('--local-port', action="store", dest="local_port", type=int, required=True)

        parser.add_argument('--remote-host', action="store", dest="remote_host",
default=REMOTE_SERVER_HOST)

        parser.add_argument('--remote-port', action="store", dest="remote_port", type=int, default=80)

        given_args = parser.parse_args()

        local_host, remote_host = given_args.local_host, given_args.remote_host

        local_port, remote_port = given_args.local_port, given_args.remote_port

        print("Starting port forwarding local %s:%s => remote %s:%s" % (local_host, local_port,
remote_host, remote_port))

        PortForwarder(local_host, local_port, remote_host, remote_port)

        asyncore.loop()

    except:

        print()
```

OUTPUT:

```
02
Starting port forwarding local localhost:8802 => remote www.nitrkl.ac.in:80
Connected to : ('127.0.0.1', 1303)
Connected to : ('127.0.0.1', 1305)
Connected to : ('127.0.0.1', 1306)
Connected to : ('127.0.0.1', 1307)
Connected to : ('127.0.0.1', 1308)
Connected to : ('127.0.0.1', 1310)
Connected to : ('127.0.0.1', 1315)
```

14. Write a client that will wait for a particular network service forever or for a time out?

```
import argparse
import socket
import errno
from time import time as now
DEFAULT_TIMEOUT = 20
DEFAULT_SERVER_HOST = '127.0.0.1'
DEFAULT_SERVER_PORT = 12345
class NetServiceChecker(object):
    def __init__(self, host, port, timeout=DEFAULT_TIMEOUT):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    def end_wait(self):
        self.sock.close()
    def check(self):
        if self.timeout:
            end_time = now() + self.timeout
        while True:
            try:
                if self.timeout:
                    next_timeout = end_time - now()
                    if next_timeout < 0:
                        return False
                    else:
```

```
                print("setting socket next timeout ", round(next_timeout),"s")
                self.sock.settimeout(next_timeout)
            self.sock.connect((self.host, self.port))
        # handle exceptions
        except socket.timeout:
            if self.timeout:
                return False
        except socket.error:
            print("Exception")
        else: # if all goes well
            self.end_wait()
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Wait for Network Service')
    parser.add_argument('--host', action="store", dest="host",default=DEFAULT_SERVER_HOST)
    parser.add_argument('--port', action="store", dest="port",type=int, default=DEFAULT_SERVER_PORT)
    parser.add_argument('--timeout', action="store", dest="timeout",type=int,
default=DEFAULT_TIMEOUT)
    given_args = parser.parse_args()
    host, port, timeout = given_args.host, given_args.port, given_args.timeout
    service_checker = NetServiceChecker(host, port, timeout=timeout)
    print("Checking for network service %s:%s ..." %(host, port))
    if service_checker.check():
        print("Service is available again!")
```

OUTPUT:

```
Checking for network service 127.0.0.1:12345 ...
setting socket next timeout  20 s
Exception
setting socket next timeout  19 s
Exception
setting socket next timeout  18 s
Exception
setting socket next timeout  17 s
Exception
setting socket next timeout  16 s
Exception
setting socket next timeout  15 s
Exception
setting socket next timeout  14 s
Exception
setting socket next timeout  13 s
Exception
setting socket next timeout  12 s
Exception
setting socket next timeout  11 s
Exception
setting socket next timeout  10 s
Exception
setting socket next timeout  9 s
Exception
setting socket next timeout  8 s
Exception
setting socket next timeout  7 s
Exception
setting socket next timeout  6 s
Exception
setting socket next timeout  5 s
Exception
setting socket next timeout  4 s
Exception
setting socket next timeout  3 s
Exception
setting socket next timeout  2 s
Exception
setting socket next timeout  0 s
```

Q15. Write a program to list the network interfaces present in your machine?

```
import netifaces

print(netifaces.interfaces())
```

OUTPUT:

```
['{C672CC8E-4EE1-4080-B9A0-6AFC654A15E9}', '{FFF266BF-59C7-4A4F-85EE-E38170D8FBE
A}', '{21930592-68F6-497C-B981-5E237703049D}', '{E29AC6C2-7037-11DE-816D-806E6F6
E6963}', '{DAB1C05F-5BFC-4E9E-9673-0F80E8D3E780}', '{410D6A35-2B41-48EA-978A-6BF
B702CA90B}', '{5A386972-36B9-4FCD-90AC-43968D055978}', '{3F59EB0E-DC66-4B5B-B665
-EEFFDA65B13E}']
```

17. Extend the client/server interaction to simulate a password dialogue. After receiving data from a client, the server returns access granted or access denied depending on whether the received data matches the password.

Code:

<u>Server</u>

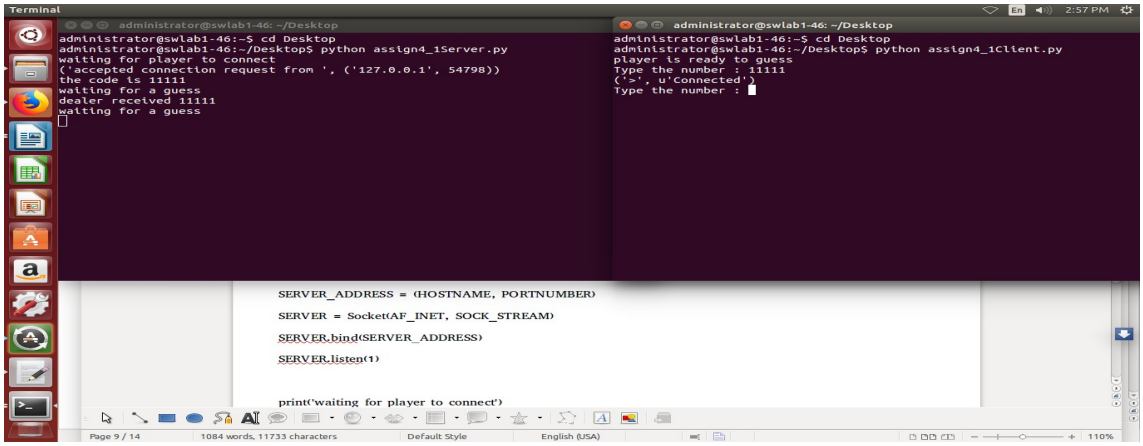
```
from random import randint

from socket import socket as Socket

from socket import AF_INET, SOCK_STREAM
```

```
HOSTNAME = ''        # blank so any address can be used
PORTNUMBER = 12345  # number for the port
BUFFER = 80          # size of the buffer

SERVER_ADDRESS = (HOSTNAME, PORTNUMBER)
SERVER = Socket(AF_INET, SOCK_STREAM)
SERVER.bind(SERVER_ADDRESS)
SERVER.listen(1)

print('waiting for player to connect')
PLAYER, PLAYER_ADDRESS = SERVER.accept()
print('accepted connection request from ',PLAYER_ADDRESS)
CODE = 11111
print('the code is %d' % CODE)

while True:
    print('waiting for a guess')
    GUESS = PLAYER.recv(BUFFER).decode()
    print('dealer received ' + GUESS)
    if int(GUESS) == CODE:
        REPLY = 'Connected'
    else:
        REPLY = 'Wrong Password'
    PLAYER.send(REPLY.encode())
SERVER.close()
```

Client:

```
from socket import socket as Socket

from socket import AF_INET, SOCK_STREAM

HOSTNAME = 'localhost'  # on same host

PORTNUMBER = 12345      # same port number

BUFFER = 80             # size of the buffer

SERVER = (HOSTNAME, PORTNUMBER)

PLAYER = Socket(AF_INET, SOCK_STREAM)

PLAYER.connect(SERVER)

print('player is ready to guess')

while True:

    GUESS = input('Type the number : ')

    PLAYER.send(repr(GUESS).encode())

    ANSWER = PLAYER.recv(BUFFER).decode()

    print('>', ANSWER)

    if ANSWER == 'Connecte':

        break

PLAYER.close()
```

Output:



18. Write a program that compress your working directory and email to a specific address?

Code:

```
import os

import argparse

import smtplib

import zipfile

import tempfile

from email import encoders

from email.mime.base import MIMEBase

from email.mime.multipart import MIMEMultipart

def email_dir_zipped(sender, recipient):

    zf = tempfile.TemporaryFile(prefix='mail', suffix='.zip')

    zip = zipfile.ZipFile(zf, 'w')

    print ("Zipping current dir: %s" %(os.getcwd()))

    for file_name in os.listdir(os.getcwd()):

        zip.write(file_name)

    zip.close()

    zf.seek(0)

    # Create the message

    print ("Creating email message...")

    email_msg = MIMEMultipart()

    email_msg['Subject'] = ('File from path %s' %os.getcwd())

    email_msg['To'] = ', '.join(recipient)

    email_msg['From'] = sender

    email_msg.preamble = 'Testing email from Python.\n'

    msg = MIMEBase('application', 'zip')

    msg.set_payload(zf.read())
```

```
    encoders.encode_base64(msg)
    msg.add_header('Content-Disposition', 'attachment',
    filename=os.getcwd()[-1] + '.zip')
    email_msg.attach(msg)
    email_msg = email_msg.as_string()
    # send the message
    print ("Sending email message...")
    smtp = None
    try:
        smtp = smtplib.SMTP('localhost')
        smtp.set_debuglevel(1)
        smtp.sendmail(sender, recipient, email_msg)
    except Exception as e:
        print ("Error: %s" %str(e))
    finally:
        if smtp:
            smtp.close()
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Email Example')
    parser.add_argument('--sender', action="store", dest="sender",default='115cs0016@nitrkl.ac.in')
    parser.add_argument('--recipient', action="store",dest="recipient",default='115cs0016@nitrkl.ac.in')
    given_args = parser.parse_args()
    email_dir_zipped(given_args.sender, given_args.recipient)
```

19. Write a python script to check email message from your Google account with Internet Message Access Protocol) IMAP.

Code:

```
import imaplib

import pprint


imap_host = 'imap.gmail.com'

imap_user = 'username@gmail.com'

imap_pass = 'password'


# connect to host using SSL

imap = imaplib.IMAP4_SSL(imap_host)


## login to server

imap.login(imap_user, imap_pass)


imap.select('Inbox')


tmp, data = imap.search(None, 'ALL')

for num in data[0].split():

        tmp, data = imap.fetch(num, '(RFC822)')

        print('Message: {0}\n'.format(num))

        pprint.pprint(data[0][1])

        break

imap.close()
```

22. Write a program to demonstrate the loading of a local file to a remote FTP?

```
import os

import argparse

import ftplib

import getpass

LOCAL_FTP_SERVER = 'localhost'

LOCAL_FILE = 'readme.txt'

def ftp_upload(ftp_server, username, password, file_name):

        print "Connecting to FTP server: %s" %ftp_server

        ftp = ftplib.FTP(ftp_server)

        print "Login to FTP server: user=%s" %username

        ftp.login(username, password)

        ext = os.path.splitext(file_name)[1]

        if ext in (".txt", ".htm", ".html"):

                ftp.storlines("STOR " + file_name, open(file_name))

        else:

                ftp.storbinary("STOR " + file_name, open(file_name, "rb"), 1024)

        print "Uploaded file: %s" %file_name

if __name__ == '__main__':

        parser = argparse.ArgumentParser(description='FTP Server Upload Example')

        parser.add_argument('--ftp-server', action="store", dest="ftp_server", default=LOCAL_FTP_SERVER)

        parser.add_argument('--file-name', action="store", dest="file_name", default=LOCAL_FILE)

        parser.add_argument('--username', action="store", dest="username",default=getpass.getuser())

        given_args = parser.parse_args()

        ftp_server, file_name, username = given_args.ftp_server, given_args.file_name,
given_args.username
```

```
    password = getpass.getpass(prompt="Enter you FTP password: ")
    ftp_upload(ftp_server, username, password, file_name)
```

23) Write a program that compress your current working directory and then email as a message. You can send the email message via an external Gmail SMTP host, or you can use a local email server to do this.

```
import os
import argparse
import smtplib
import zipfile
import tempfile
from email import encoders
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart
def email_dir_zipped(sender, recipient):
    zf = tempfile.TemporaryFile(prefix='mail', suffix='.zip')
    zip = zipfile.ZipFile(zf, 'w')
    print "Zipping current dir: %s" %os.getcwd()
    for file_name in os.listdir(os.getcwd()):
        zip.write(file_name)
    zip.close()
    zf.seek(0)
    # Create the message
    print "Creating email message..."
    email_msg = MIMEMultipart()
    email_msg['Subject'] = 'File from path %s' %os.getcwd()
    email_msg['To'] = ', '.join(recipient)
```

```
        email_msg['From'] = sender
        email_msg.preamble = 'Testing email from Python.\n'
        msg = MIMEBase('application', 'zip')
        msg.set_payload(zf.read())
        encoders.encode_base64(msg)
        msg.add_header('Content-Disposition', 'attachment',
        filename=os.getcwd()[-1] + '.zip')
        email_msg.attach(msg)
        email_msg = email_msg.as_string()
        print "Sending email message..."
        smtp = None
        try:
                smtp = smtplib.SMTP('localhost')
                smtp.set_debuglevel(1)
                smtp.sendmail(sender, recipient, email_msg)
        except Exception, e:
                print "Error: %s" %str(e)
        finally:
                if smtp:
                        smtp.close()
if __name__ == '__main__':
        parser = argparse.ArgumentParser(description='Email Example')
        parser.add_argument('--sender', action="store", dest="sender",
        default='you@you.com')
        parser.add_argument('--recipient', action="store",
        dest="recipient")
        given_args = parser.parse_args()
        email_dir_zipped(given_args.sender, given_args.recipient)
```