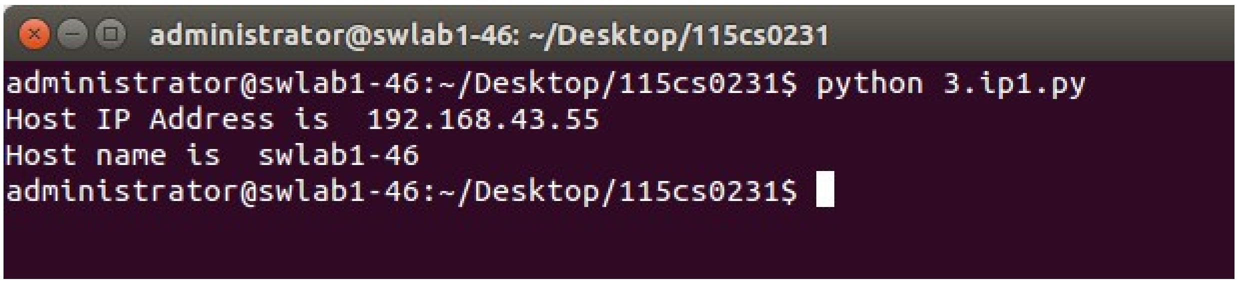
ASSIGNMENT 2

1. Printing your machine’s name and IPv4 address?

```
import os
import socket
import fcntl
import struct

def get_interface_ip(ifname):
	s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
	return socket.inet_ntoa(fcntl.ioctl(s.fileno(), 0x8915,
struct.pack('256s',ifname[:15]))[20:24])


def get_host_name():
   host_name = socket.gethostname()
   return host_name

print "Host IP Address is ",get_interface_ip('eth0')
print "Host name is ",get_host_name()
#print (1)
```

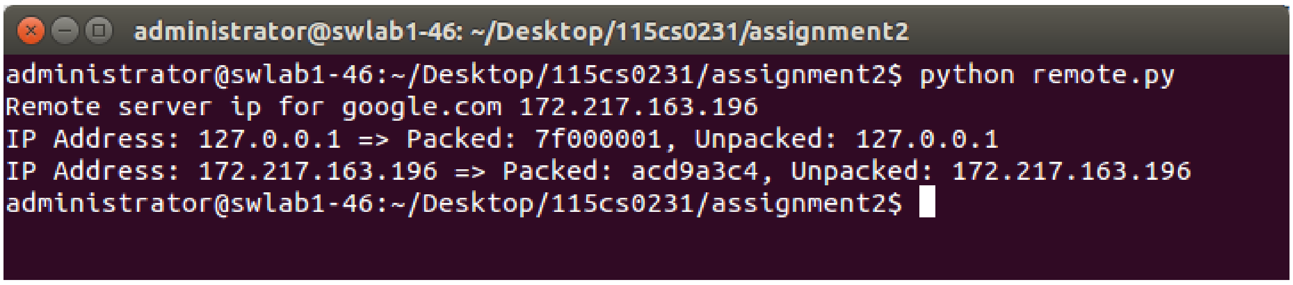


administrator@swlab1-46: ~/Desktop/115cs0231

```
administrator@swlab1-46:~/Desktop/115cs0231$ python 3.ip1.py
Host IP Address is  192.168.43.55
Host name is  swlab1-46
administrator@swlab1-46:~/Desktop/115cs0231$
```

2.Retrieve a remote machine’s IP address and convert the IP address to different format?

```
import socket
ip = socket.gethostbyname('www.google.com')
print "Remote server ip for google.com",ip

from binascii import hexlify
def convert_ip4_address():
	for ip_addr in ['127.0.0.1', ip]:
		packed_ip_addr = socket.inet_aton(ip_addr)
		unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
		print "IP Address: %s => Packed: %s, Unpacked: %s"\
	%(ip_addr, hexlify(packed_ip_addr), unpacked_ip_addr)


convert_ip4_address()
```

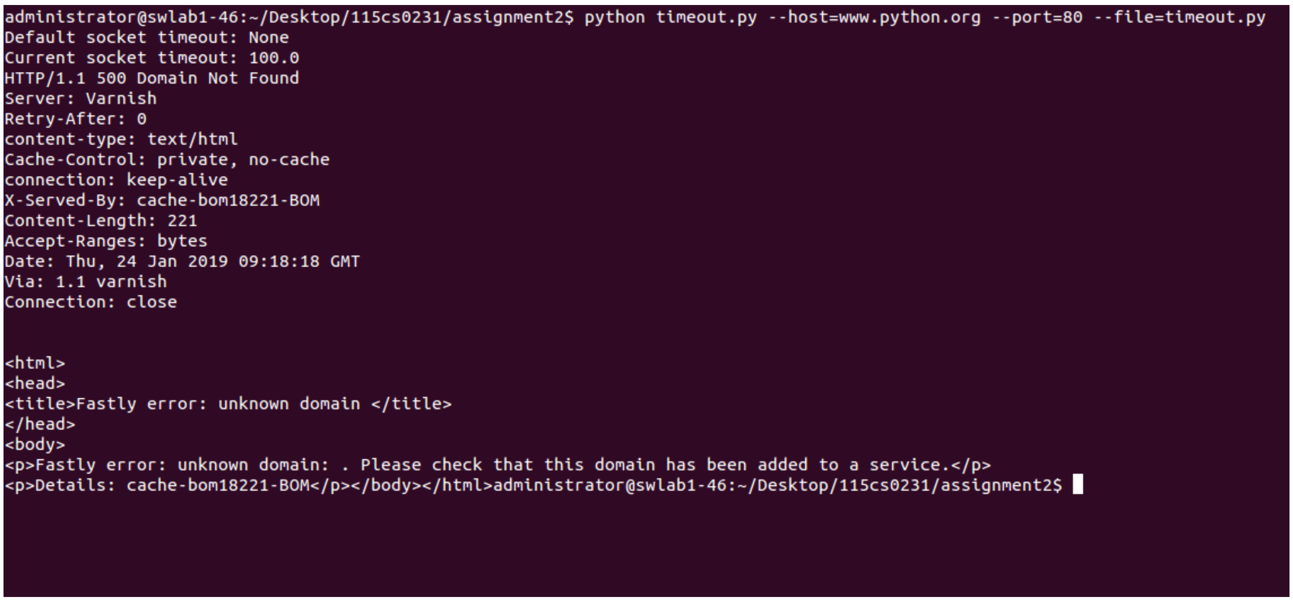


```
administrator@swlab1-46: ~/Desktop/115cs0231/assignment2
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python remote.py
Remote server ip for google.com 172.217.163.196
IP Address: 127.0.0.1 => Packed: 7f000001, Unpacked: 127.0.0.1
IP Address: 172.217.163.196 => Packed: acd9a3c4, Unpacked: 172.217.163.196
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$
```

3.Setting and getting the default socket timeout, the program should include how to handle the socket error gracefully?

```
import sys
import socket
import argparse
def test_socket_timeout():
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print "Default socket timeout: %s" %s.gettimeout()
        s.settimeout(100)
        print "Current socket timeout: %s" %s.gettimeout()


def main():
        test_socket_timeout()
        # setup argument parsing
        parser = argparse.ArgumentParser(description='Socket Error Examples')
        parser.add_argument('--host', action="store", dest="host",
        required=False)
        parser.add_argument('--port', action="store", dest="port",
        type=int, required=False)
        parser.add_argument('--file', action="store", dest="file",
        required=False)
        given_args = parser.parse_args()
        host = given_args.host
        port = given_args.port
        filename = given_args.file
        # First try-except block -- create socket
        try:
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        except socket.error, e:
                print "Error creating socket: %s" % e
                sys.exit(1)
        # Second try-except block -- connect to given host/port
        try:
                s.connect((host, port))
        except socket.gaierror, e:
                print "Address-related error connecting to server: %s" % e
                sys.exit(1)
        except socket.error, e:
                print "Connection error: %s" % e
                sys.exit(1)
        try:
                s.sendall("GET %s HTTP/1.0\r\n\r\n" % filename)
        except socket.error, e:
                print "Error sending data: %s" % e
                sys.exit(1)
        while 1:
```

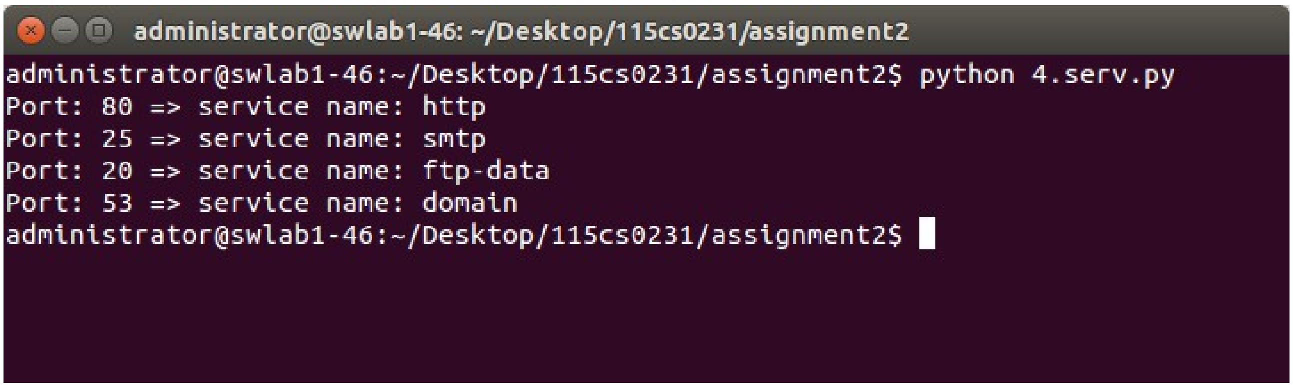
```
        # Fourth tr-except block -- waiting to receive data from remote host
                try:
                        buf = s.recv(2048)
                except socket.error, e:
                        print "Error receiving data: %s" % e
                        sys.exit(1)
                if not len(buf):
                        break
                # write the received data
                sys.stdout.write(buf)

main()
```



```
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python timeout.py --host=www.python.org --port=80 --file=timeout.py
Default socket timeout: None
Current socket timeout: 100.0
HTTP/1.1 500 Domain Not Found
Server: Varnish
Retry-After: 0
content-type: text/html
Cache-Control: private, no-cache
connection: keep-alive
X-Served-By: cache-bom18221-BOM
Content-Length: 221
Accept-Ranges: bytes
Date: Thu, 24 Jan 2019 09:18:18 GMT
Via: 1.1 varnish
Connection: close


<html>
<head>
<title>Fastly error: unknown domain </title>
</head>
<body>
<p>Fastly error: unknown domain: . Please check that this domain has been added to a service.</p>
<p>Details: cache-bom18221-BOM</p></body></html>administrator@swlab1-46:~/Desktop/115cs0231/assignment2$
```

4. Finding the service name, given the port and protocol of the remote host (server)?

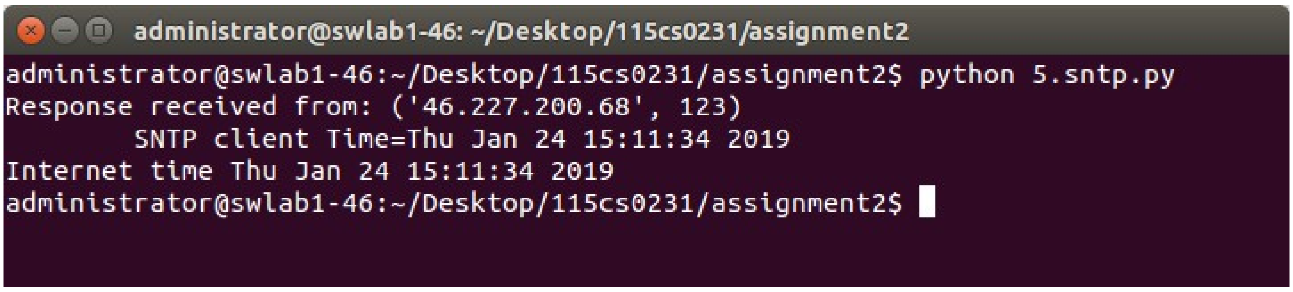
```
import socket
def find_service_name():
        protocolname = 'tcp'
        for port in [80, 25,20]:
                print "Port: %s => service name: %s" %(port, socket.
                getservbyport(port, protocolname))
        print "Port: %s => service name: %s" %(53, socket.
        getservbyport(53, 'udp'))

find_service_name()
```



```
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python 4.serv.py
Port: 80 => service name: http
Port: 25 => service name: smtp
Port: 20 => service name: ftp-data
Port: 53 => service name: domain
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$
```

5.Printing the current time from the internet time server with the help of NTP? Also write an SNTP client that prints the current time from the internet time server received with the SNTP protocol?

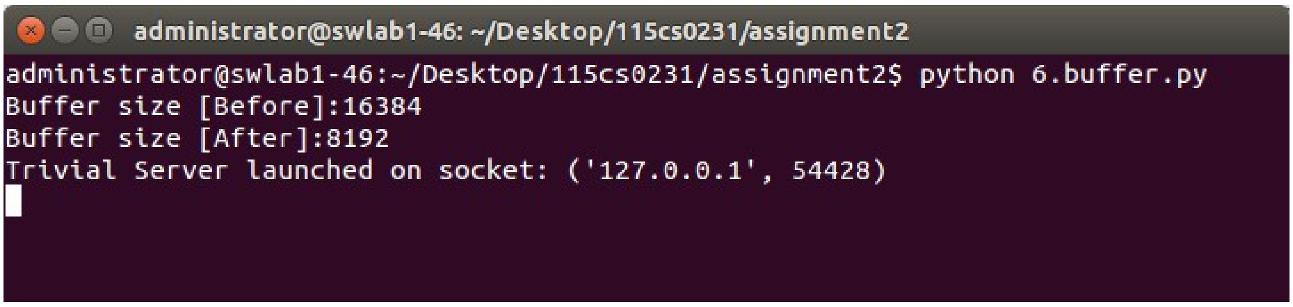
```
import ntplib
import socket
import struct
import sys
import time
from time import ctime
def print_time():
        ntp_client = ntplib.NTPClient()
        response = ntp_client.request('pool.ntp.org')
        print "Internet time",ctime(response.tx_time)


def sntp_client():
        NTP_SERVER = "0.uk.pool.ntp.org"
        TIME1970 = 2208988800L
        client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        data = '\x1b' + 47 * '\0'

        client.sendto(data, (NTP_SERVER, 123))
        data, address = client.recvfrom( 1024 )
        if data:
                print 'Response received from:', address
        t = struct.unpack( '!12I', data )[10]
        t -= TIME1970
        print '\tSNTP client Time=%s' % time.ctime(t)

sntp_client()
print_time()
```



```
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python 5.sntp.py
Response received from: ('46.227.200.68', 123)
        SNTP client Time=Thu Jan 24 15:11:34 2019
Internet time Thu Jan 24 15:11:34 2019
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$
```

6. Modifying sockets send/receive buffer size and changing the socket to blocking/non-blocking mode?

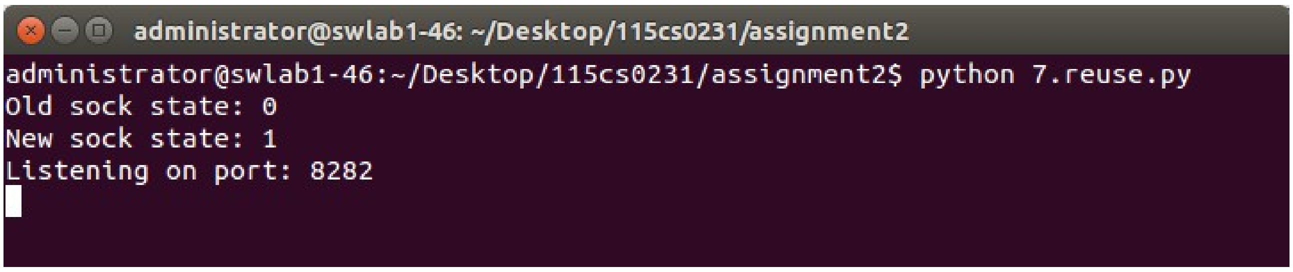
```
import socket
SEND_BUF_SIZE = 4096
RECV_BUF_SIZE = 4096
def modify_buff_size():
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM )
        # Get the size of the socket's send buffer
        bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
        print "Buffer size [Before]:%d" %bufsize
        sock.setsockopt(socket.SOL_TCP, socket.TCP_NODELAY, 1)
        sock.setsockopt(socket.SOL_SOCKET,socket.SO_SNDBUF, SEND_BUF_SIZE)
        sock.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF,  RECV_BUF_SIZE)
        bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
        print "Buffer size [After]:%d" %bufsize

def test_socket_modes():
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setblocking(1)
        s.settimeout(0.5)
        s.bind(("127.0.0.1", 0))
        socket_address = s.getsockname()
        print "Trivial Server launched on socket: %s" %str(socket_address)
        while(1):
                s.listen(1)


modify_buff_size()
test_socket_modes()
```



```
administrator@swlab1-46: ~/Desktop/115cs0231/assignment2
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python 6.buffer.py
Buffer size [Before]:16384
Buffer size [After]:8192
Trivial Server launched on socket: ('127.0.0.1', 54428)
```

7.Write a program that demonstrates the reuse socket addresses?

```
import socket
import sys
def reuse_socket_addr():
        sock = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
        # Get the old state of the SO_REUSEADDR option
        old_state = sock.getsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR)
        print "Old sock state: %s" %old_state
        # Enable the SO_REUSEADDR option
        sock.setsockopt( socket.SOL_SOCKET, socket.SO_REUSEADDR, 1 )
        new_state = sock.getsockopt( socket.SOL_SOCKET, socket.SO_REUSEADDR )
        print "New sock state: %s" %new_state
        local_port = 8282
        srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        srv.bind( ('', local_port) )
        srv.listen(1)
        print ("Listening on port: %s " %local_port)
        while True:
                try:
                        connection, addr = srv.accept()
                        print 'Connected by %s:%s' % (addr[0], addr[1])
                except KeyboardInterrupt:
                        break
                except socket.error, msg:
                        print '%s' % (msg,)

reuse_socket_addr()
```



```
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python 7.reuse.py
Old sock state: 0
New sock state: 1
Listening on port: 8282
```

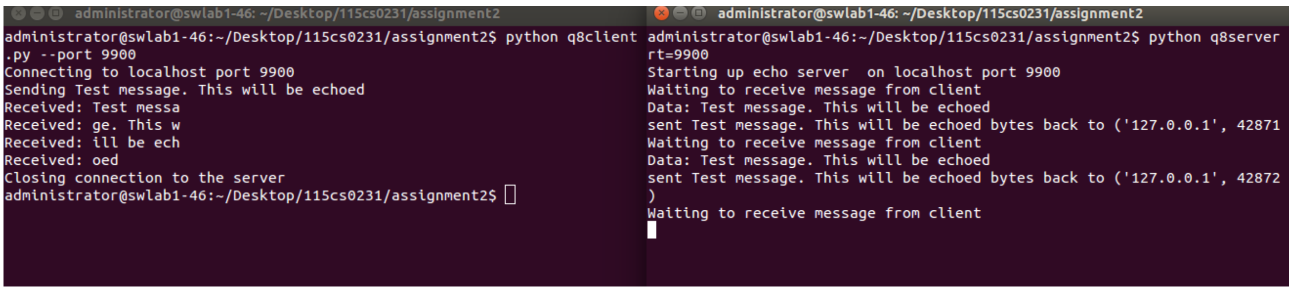
8.Write a simple TCP echo client/server application with the help of TCP socket object. The server wait for the client to be connected and send some data to the server. When the data is received, the server echoes the data to the client.

Server side

```
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048
backlog = 5

def echo_server(port):
    """ A simple echo server """
    # Create a TCP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Enable reuse address/port
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # Bind the socket to the port
    server_address = (host, port)
    print ("Starting up echo server  on %s port %s" % server_address)
    sock.bind(server_address)
    # Listen to clients, backlog argument specifies the max no. of queued connections
    sock.listen(backlog)
    while True:
        print ("Waiting to receive message from client")
        client, address = sock.accept()
        data = client.recv(data_payload)
        if data:
            print ("Data: %s" %data)
            client.send(data)
            print ("sent %s bytes back to %s" % (data, address))
        # end connection
        client.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

Client side:

```python
import socket
import sys

import argparse

host = 'localhost'

def echo_client(port):
    """ A simple echo client """
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Connect the socket to the server
    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    sock.connect(server_address)

    # Send data
    try:
        # Send data
        message = "Test message. This will be echoed"
        print ("Sending %s" % message)
        sock.sendall(message.encode('utf-8'))
        # Look for the response
        amount_received = 0
        amount_expected = len(message)
        while amount_received < amount_expected:
            data = sock.recv(10)
            amount_received += len(data)
            print ("Received: %s" % data)
    except socket.error as e:
        print ("Socket error: %s" %str(e))
    except Exception as e:
        print ("Other exception: %s" %str(e))
    finally:
        print ("Closing connection to the server")
        sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```



```
administrator@swlab1-46:~/Desktop/115cs0231/assignment2
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python q8client
.py --port 9900
Connecting to localhost port 9900
Sending Test message. This will be echoed
Received: Test messa
Received: ge. This w
Received: ill be ech
Received: oed
Closing connection to the server
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$
```

```
administrator@swlab1-46:~/Desktop/115cs0231/assignment2
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python q8server
rt=9900
Starting up echo server  on localhost port 9900
Waiting to receive message from client
Data: Test message. This will be echoed
sent Test message. This will be echoed bytes back to ('127.0.0.1', 42871
Waiting to receive message from client
Data: Test message. This will be echoed
sent Test message. This will be echoed bytes back to ('127.0.0.1', 42872
)
Waiting to receive message from client
```

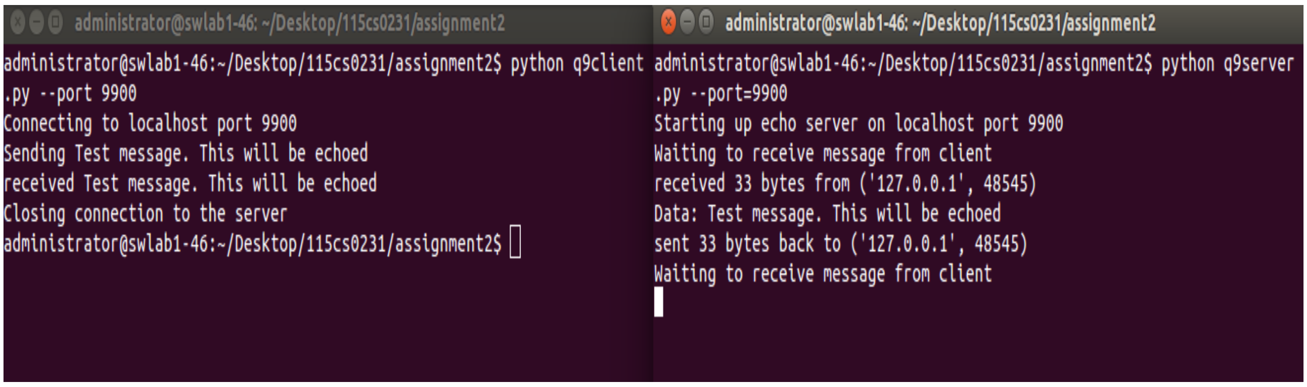
9.Write a simple UDP echo client/server application with the help of TCP socket object. The server wait for the client to be connected and send some data to the server. When the data is received, the server echoes the data to the client.

Server Side:

```
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_server(port):
    """ A simple echo server """
    # Create a UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # Bind the socket to the port
    server_address = (host, port)
    print ("Starting up echo server on %s port %s" % server_address)

    sock.bind(server_address)

    while True:
        print ("Waiting to receive message from client")
        data, address = sock.recvfrom(data_payload)

        print ("received %s bytes from %s" % (len(data), address))
        print ("Data: %s" %data)

        if data:
            sent = sock.sendto(data, address)
            print ("sent %s bytes back to %s" % (sent, address))


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

Client side:

```
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_client(port):
    """ A simple echo client """
    # Create a UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    message = 'This is the message.  It will be repeated.'

    try:
        # Send data
        message = "Test message. This will be echoed"
        print ("Sending %s" % message)
        sent = sock.sendto(message.encode('utf-8'), server_address)

        # Receive response
        data, server = sock.recvfrom(data_payload)
        print ("received %s" % data)

    finally:
        print ("Closing connection to the server")
        sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```



```
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python q9client
.py --port 9900
Connecting to localhost port 9900
Sending Test message. This will be echoed
received Test message. This will be echoed
Closing connection to the server
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$
```

```
administrator@swlab1-46:~/Desktop/115cs0231/assignment2$ python q9server
.py --port=9900
Starting up echo server on localhost port 9900
Waiting to receive message from client
received 33 bytes from ('127.0.0.1', 48545)
Data: Test message. This will be echoed
sent 33 bytes back to ('127.0.0.1', 48545)
Waiting to receive message from client
```

administrator@swlab1-46: ~/Desktop/115cs0231/assignment2
administrator@swlab1-46: ~/Desktop/115cs0231/assignment2