MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

Santosh, Tangail -1902

Lab Report No	:	**06**
Lab Report Name	:	Python for Networking
Course Name	:	Computer Networks Lab
Course Code	:	ICT - 3208

Submitted by,

Name : Md. Imtyaz Ahmed

ID : IT-17017

Session : 2016-17
Dept. of ICT, MBSTU.

Submitted to,

Nazrul Islam

Assistant Professor

Dept. of ICT, MBSTU

Objective :

The objective of the lab is to :

- Install python and use third-party libraries
- Interact with network interfaces using python
- Getting information from internet using Python

Theory:

Third party libraries:

Networking Glossary: Before we start discussing networking with any depth, we must define some common terms that you simply will see throughout this guide, and in other guides and documentation regarding networking.

Connection: In networking, a connection refers to pieces of related information that are transferred through a network. This generally infers that a connection is made before the info transfer (by following the procedures laid call at a protocol) then is deconstructed at the top of the info transfer.

Packet: A packet is, generally speaking, the foremost basic unit that's transferred over a network. When communicating over a network, packets are the envelopes that carry your data (in pieces) from one end point to the opposite . Packets have a header portion that contains information about the packet including the source and destination, timestamps, network hops, etc. the most portion of a packet contains the particular data being transferred. it's sometimes called the body or the payload.

Network Interface: A network interface can ask any quite software interface to networking hardware. as an example , if you've got two network cards in your computer, you'll control and configure each network interface related to them individually. A network interface could also be related to a physical device, or it's going to be a representation of a virtual interface. The "loopback" device, which may be a virtual interface to the local machine, is an example of this.

LAN: LAN stands for "local area network". It refers to a network or some of a network that's not publicly accessible to the greater internet. A home or office network is an example of a LAN.

WAN: WAN stands for "wide area network". It means a network that's far more extensive than a LAN. While WAN is that the relevant term to use to explain large, dispersed networks generally , it's usually meant to mean the web , as an entire . If an interface is claimed to be connected to the WAN, it's generally assumed that it's reachable through the web .

Protocol: A protocol may be a set of rules and standards that basically define a language that devices can use to speak . There are an excellent number of protocols in use extensively in networking, and that they are often implemented in several layers. Some low level protocols are TCP, UDP, IP, and ICMP. Some familiar samples of application layer protocols, built on these lower protocols, are HTTP (for accessing web content), SSH, TLS/SSL, and FTP. Port: A port is an address on one machine which will be tied to a selected piece of software. it's not a physical interface or location, but it allows your server to be ready to communicate using quite one application.

Firewall: A firewall may be a program that decides whether traffic coming into a server or going out should be allowed. A firewall usually works by creating rules that sort of traffic is suitable on which ports. Generally, firewalls block ports that aren't employed by a selected application on a server.

NAT: NAT stands for network address translation. it's how to translate requests that are incoming into a routing server to the relevant devices or servers that it knows about within the LAN. this is often usually implemented in physical LANs as how to route requests through one IP address to the required backend servers.

VPN: VPN stands for virtual private network. it's a way of connecting separate LANs through the web , while maintaining privacy. this is often used as a way of connecting remote systems as if they were on an area network, often for security reasons.

Interfaces: Interfaces are networking communication points for your computer. Each interface is related to a physical or virtual networking device. Typically, your server will have one configurable network interface for every Ethernet or wireless internet card you've got . additionally , it'll define a virtual network interface called the "loopback" or localhost interface. this is often used as an interface to attach applications and processes on one computer to other applications and processes. you'll see this referenced because the "lo" interface in many tools. repeatedly , administrators configure one interface to service traffic to the web and another interface for a LAN or private network.

Protocols: Networking works by piggybacking variety of various protocols on top of every other. during this way, one piece of knowledge are often transmitted using multiple protocols encapsulated within each other . we'll mention a number of the more common protocols that you simply may encounter and plan to explain the difference, also as give context on what a part of the method they're involved . we'll start with protocols implemented on the lower networking layers and work our high to protocols with higher abstraction.

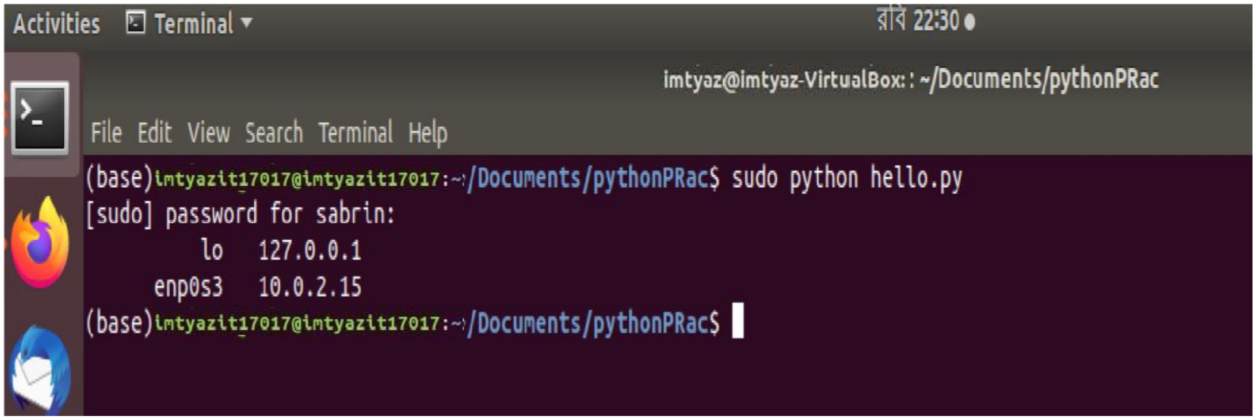
Exercise 4.1: Enumerating interfaces on your machine

Code :

```
import socket
import fcntl
import struct
import array



def all_interfaces():     max_possible = 128  #
arbitrary. raise if needed.     bytes = max_possible
* 32

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

   names = array.array('B', '\0' * bytes)

   outbytes = struct.unpack('iL', fcntl.ioctl(
s.fileno(),

      0x8912,  # SIOCGIFCONF

      struct.pack('iL', bytes, names.buffer_info()[0])

   ))[0]

   namestr = names.tostring()

   lst = []     for i in range(0, outbytes, 40):
name = namestr[i:i+16].split('\0', 1)[0]
ip   = namestr[i+20:i+24]
lst.append((name, ip))

   return lst


def format_ip(addr):

   return str(ord(addr[0])) + '.' + \
str(ord(addr[1])) + '.' + \
str(ord(addr[2])) + '.' + \
        str(ord(addr[3]))
```

```
ifs = all_interfaces()
for i in ifs:

    print "%12s   %s" % (i[0], format_ip(i[1]))
```

Output :


Activities
Terminal ▾
রবি 22:30 ●
imtyaz@imtyaz-VirtualBox: : ~/Documents/pythonPRac
File Edit View Search Terminal Help

```
(base)imtyazit17017@imtyazit17017:~/Documents/pythonPRac$ sudo python hello.py
[sudo] password for sabrin:
          lo   127.0.0.1
      enp0s3   10.0.2.15
(base)imtyazit17017@imtyazit17017:~/Documents/pythonPRac$
```

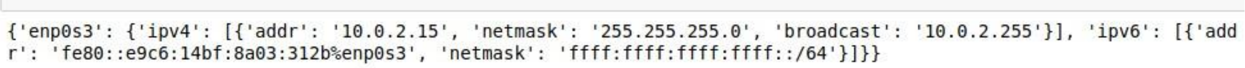
Exercise 4.2: Finding the IP address for a specific interface on your machine

Code :

```
import netifaces def
get_interfaces():

    interfaces = netifaces.interfaces()
interfaces.remove('lo')

    out_interfaces = dict()

    for interface in interfaces:         addrs =
netifaces.ifaddresses(interface)
out_addrs = dict()        if netifaces.AF_INET
in addrs.keys():

            out_addrs["ipv4"] = addrs[netifaces.AF_INET]
if netifaces.AF_INET6 in addrs.keys():
```

```
out_addrs["ipv6"] = addrs[netifaces.AF_INET6]
out_interfaces[interface] = out_addrs

    return out_interfaces

print(get_interfaces())
```

Output :



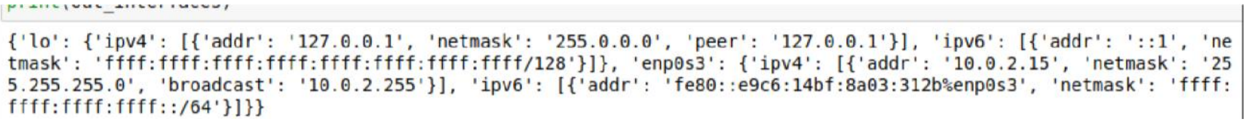
```
{'enp0s3': {'ipv4': [{'addr': '10.0.2.15', 'netmask': '255.255.255.0', 'broadcast': '10.0.2.255'}], 'ipv6': [{'add
r': 'fe80::e9c6:14bf:8a03:312b%enp0s3', 'netmask': 'ffff:ffff:ffff:ffff::/64'}]}}
```

Exercise 4.3: Finding whether an interface is up on your machine

Code :

```
interfaces = netifaces.interfaces()
out_interfaces = dict()

for interface in interfaces:     addrs =
netifaces.ifaddresses(interface)
out_addrs = dict()     if netifaces.AF_INET
in addrs.keys():

        out_addrs["ipv4"] = addrs[netifaces.AF_INET]
if netifaces.AF_INET6 in addrs.keys():
out_addrs["ipv6"] = addrs[netifaces.AF_INET6]
out_interfaces[interface] = out_addrs
print(out_interfaces)
```

Output :



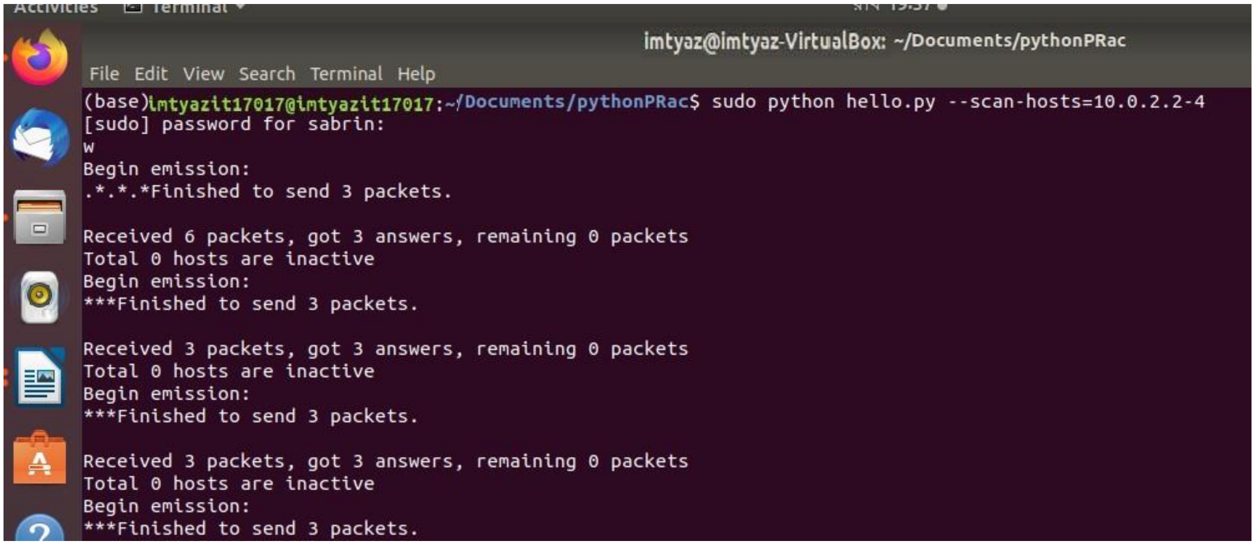
```
print(out_interfaces)

{'lo': {'ipv4': [{'addr': '127.0.0.1', 'netmask': '255.0.0.0', 'peer': '127.0.0.1'}], 'ipv6': [{'addr': '::1', 'ne
tmask': 'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff/128'}]}, 'enp0s3': {'ipv4': [{'addr': '10.0.2.15', 'netmask': '25
5.255.255.0', 'broadcast': '10.0.2.255'}], 'ipv6': [{'addr': 'fe80::e9c6:14bf:8a03:312b%enp0s3', 'netmask': 'ffff:
ffff:ffff:ffff::/64'}]}}
```

Exercise 4.4: Detecting inactive machines on your network

Code :

```
import argparse
import time import
sched
from scapy.layers.inet import *
RUN_FREQUENCY = 10

scheduler = sched.scheduler(time.time, time.sleep) def
detect_inactive_hosts(scan_hosts):


 global scheduler

 scheduler.enter(RUN_FREQUENCY, 1, detect_inactive_hosts, (scan_hosts,
))    inactive_hosts = []    try:
    ans, unans = sr(IP(dst=scan_hosts)/ICMP(), retry=0, timeout=1)
#ans.summary(lambda(s,r) : r.sprintf("%IP.src% is alive"))        for
inactive in unans:

        print ("%s is inactive" %inactive.dst)
inactive_hosts.append(inactive.dst)        print ("Total %d hosts
are inactive" %(len(inactive_hosts)))    except
KeyboardInterrupt:
    exit(0) if __name__ == "__main__":    parser =
argparse.ArgumentParser(description='Python networking utils')
parser.add_argument('--scan-hosts', action="store", dest="scan_hosts",
required=False)    given_args = parser.parse_args()    scan_hosts =
given_args.scan_hosts
 scheduler.enter(1, 1, detect_inactive_hosts, (scan_hosts, ))
scheduler.run()
```

Output :


Activities
Terminal
imtyaz@imtyaz-VirtualBox: ~/Documents/pythonPRac
File Edit View Search Terminal Help

```
(base)imtyazit17017@imtyazit17017:~/Documents/pythonPRac$ sudo python hello.py --scan-hosts=10.0.2.2-4
[sudo] password for sabrin:
w
Begin emission:
.*.*.*Finished to send 3 packets.

Received 6 packets, got 3 answers, remaining 0 packets
Total 0 hosts are inactive
Begin emission:
***Finished to send 3 packets.

Received 3 packets, got 3 answers, remaining 0 packets
Total 0 hosts are inactive
Begin emission:
***Finished to send 3 packets.

Received 3 packets, got 3 answers, remaining 0 packets
Total 0 hosts are inactive
Begin emission:
***Finished to send 3 packets.
```

Exercise 4.5: Pinging hosts on the network with ICMP

Code :

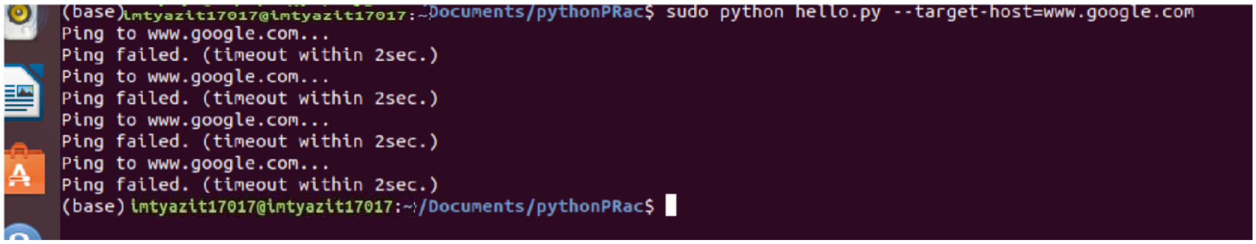
```
import os import
argparse import
socket import
struct import
select import
time
ICMP_ECHO_REQUEST = 8 # Platform specific

DEFAULT_TIMEOUT = 2
DEFAULT_COUNT = 4
class Pinger(object):


   def __init__(self, target_host,
count=DEFAULT_COUNT,timeout=DEFAULT_TIMEOUT):        self.target_host =
target_host        self.count = count        self.timeout = timeout     def do_checksum(self,
source_string):        sum = 0

      max_count = (len(source_string)/2)*2        count = 0
while count < max_count:            val =
ord(source_string[count + 1])*256
ord(source_string[count])            sum = sum + val            sum
```

```
= sum & 0xffffffff          count = count + 2        if
max_count<len(source_string):            sum = sum +
ord(source_string[len(source_string) - 1])           sum = sum &
0xffffffff        sum = (sum >> 16) + (sum & 0xffff)        sum =
sum + (sum >> 16)        answer = ~sum        answer = answer
& 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
return answer     def receive_pong(self, sock, ID,
timeout):


    time_remaining = timeout
while True:           start_time =
time.time()

        readable = select.select([sock], [], [], time_remaining)

        time_spent = (time.time() - start_time)
if readable[0] == []: # Timeout               return
        time_received = time.time()
recv_packet, addr = sock.recvfrom(1024)
icmp_header = recv_packet[20:28]

        type, code, checksum, packet_ID, sequence = struct.unpack(

        "bbHHh", icmp_header)        if
packet_ID == ID:           bytes_In_double =
struct.calcsize("d")

        time_sent = struct.unpack("d", recv_packet[28:28 +bytes_In_double])[0]
return time_received - time_sent        time_remaining = time_remaining -
time_spent        if time_remaining <= 0:           return     def send_ping(self,
sock, ID):


    target_addr = socket.gethostbyname(self.target_host)
my_checksum = 0

    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, my_checksum,ID, 1)
bytes_In_double = struct.calcsize("d")        data = (192 - bytes_In_double) * "Q"

    data = struct.pack("d", time.time()) + data
```

```
my_checksum = self.do_checksum(header + data)

    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, socket.htons(my_checksum), ID,

1)

    packet = header + data
sock.sendto(packet, (target_addr, 1))
def ping_once(self):

    icmp = socket.getprotobyname("icmp")
try:

      sock = socket.socket(socket.AF_INET, socket.SOCK_RAW,
icmp)      except socket.error(errno, msg):         if errno == 1:


        msg += "ICMP messages can only be sent from root user processes"
raise socket.error(msg)      except Exception,e:         print ("Exception: %s"
%(e))      my_ID = os.getpid() & 0xFFFF      self.send_ping(sock, my_ID)
    delay = self.receive_pong(sock, my_ID, self.timeout)
```

```
sock.close()
return delay     def
ping(self):
       for i in xrange(self.count):
print("Ping to %s..." % self.target_host)
try:
              delay = self.ping_once()           except
socket.gaierror, e:              print("Ping failed. (socket
error: '%s')" % e[1])              break           if delay ==
None:
              print("Ping failed. (timeout within %ssec.)" % self.timeout)
else:

              delay = delay * 1000               print ("Get pong in
%0.4fms" % delay) if __name__ == '__main__':     parser =
argparse.ArgumentParser(description='Python ping')

   parser.add_argument('--target-host', action="store",dest="target_host",
required=False)     given_args = parser.parse_args()     target_host =
given_args.target_host     pinger = Pinger(target_host=target_host)     pinger.ping()
```

Output :



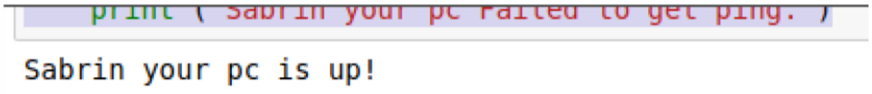
```
(base) imtyazit17017@imtyazit17017:~/Documents/pythonPRac$ sudo python hello.py --target-host=www.google.com
Ping to www.google.com...
Ping failed. (timeout within 2sec.)
Ping to www.google.com...
Ping failed. (timeout within 2sec.)
Ping to www.google.com...
Ping failed. (timeout within 2sec.)
Ping to www.google.com...
Ping failed. (timeout within 2sec.)
(base) imtyazit17017@imtyazit17017:~/Documents/pythonPRac$
```

Exercise 4.6: Pinging hosts on the network with ICMP using pc resources

Code :

```
import subprocess import shlex
command_line = "ping -c 1 10.0.2.15"
if __name__ == '__main__':

   args = shlex.split(command_line) try:

   subprocess.check_call(args,stdout=subprocess.PIPE,stderr=subprocess.PIPE)
print ("Sabrin your pc is up!") except subprocess.CalledProcessError:
   print ("Sabrin your pc Failed to get ping.")
```

Output :



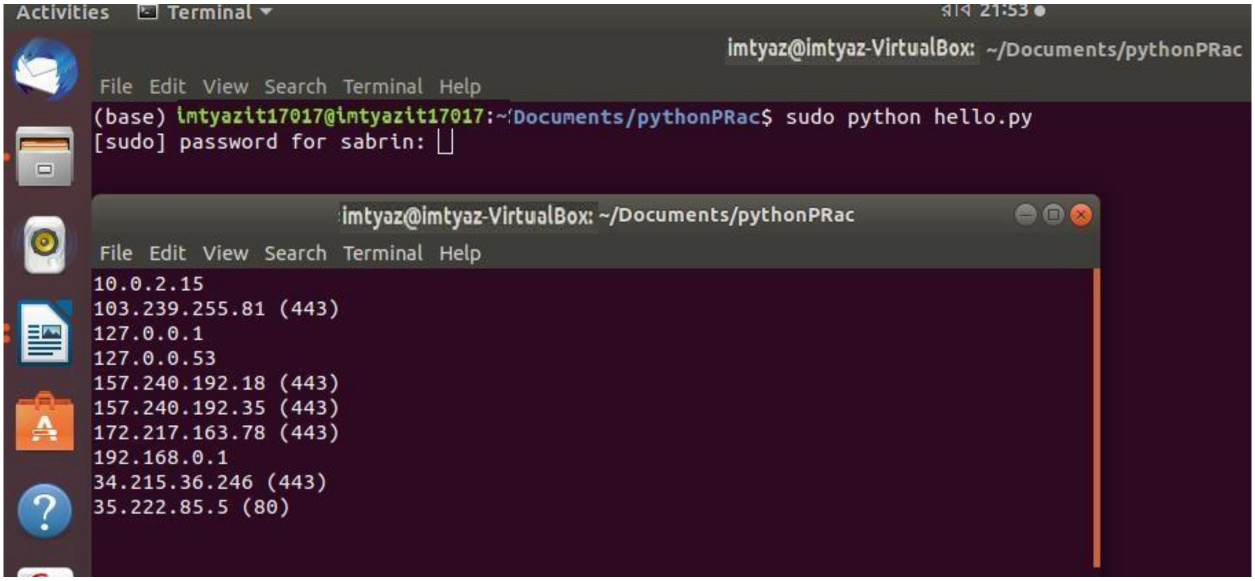
```
    print ("Sabrin your pc Failed to get ping.")

Sabrin your pc is up!
```

Exercise 4.7: Scanning the broadcast of packets

Code :

```
from scapy.all import * from
scapy.layers.inet import *

captured_data = dict()

END_PORT = 1000

def monitor_packet(pkt):     if IP in pkt:         if not
captured_data.has_key(pkt[IP].src):
captured_data[pkt[IP].src] = []     if TCP in pkt:         if
pkt[TCP].sport <= END_PORT:             if not str(pkt[TCP].sport) in
captured_data[pkt[IP].src]:
captured_data[pkt[IP].src].append(str(pkt[TCP].sport))
os.system('clear')     ip_list = sorted(captured_data.keys())     for
key in ip_list:
       ports=', '.join(captured_data[key])
if len (captured_data[key]) == 0:

          print ('%s' % key)
else:

          print ('%s (%s)' % (key, ports))
if __name__ == '__main__':
sniff(prn=monitor_packet, store=0)
```

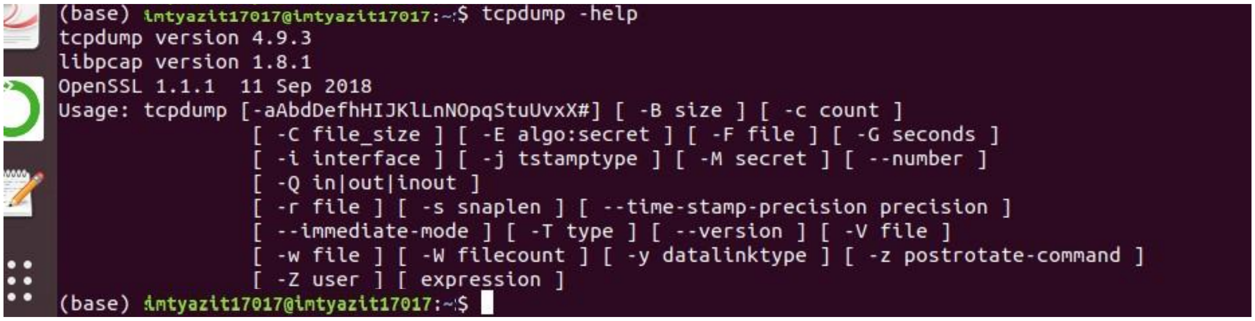
Output :


Activities
Terminal
21:53
imtyaz@imtyaz-VirtualBox: ~/Documents/pythonPRac
File Edit View Search Terminal Help
imtyaz@imtyaz-VirtualBox: ~/Documents/pythonPRac
File Edit View Search Terminal Help

```
(base) imtyazit17017@imtyazit17017:~/Documents/pythonPRac$ sudo python hello.py
[sudo] password for sabrin: 
```

```
10.0.2.15
103.239.255.81 (443)
127.0.0.1
127.0.0.53
157.240.192.18 (443)
157.240.192.35 (443)
172.217.163.78 (443)
192.168.0.1
34.215.36.246 (443)
35.222.85.5 (80)
```

Exercise 4.8: Sniffing packets on your network



```
(base) imtyazit17017@imtyazit17017:~$ tcpdump -help
tcpdump version 4.9.3
libpcap version 1.8.1
OpenSSL 1.1.1  11 Sep 2018
Usage: tcpdump [-aAbdDefhHIJKlLnNOpqStuUvxX#] [ -B size ] [ -c count ]
                [ -C file_size ] [ -E algo:secret ] [ -F file ] [ -G seconds ]
                [ -i interface ] [ -j tstamptype ] [ -M secret ] [ --number ]
                [ -Q in|out|inout ]
                [ -r file ] [ -s snaplen ] [ --time-stamp-precision precision ]
                [ --immediate-mode ] [ -T type ] [ --version ] [ -V file ]
                [ -w file ] [ -W filecount ] [ -y datalinktype ] [ -z postrotate-command ]
                [ -Z user ] [ expression ]
(base) imtyazit17017@imtyazit17017:~$ 
```

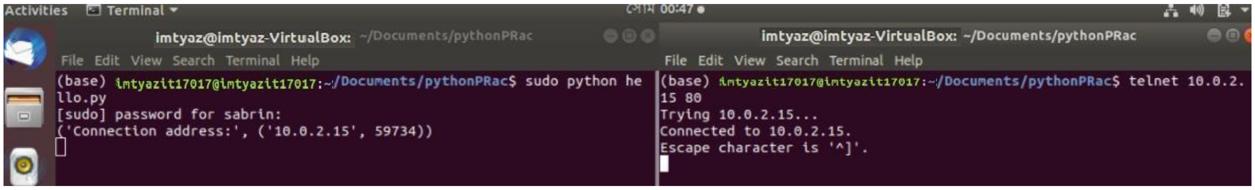
Exercise 4.9 : Performing a basic Telnet

Code :

```
import socket TCP_IP
= '10.0.2.15’

TCP_PORT = 80

BUFFER_SIZE = 20 # Normally 1024, but we want fast response
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))

s.listen(1) conn, addr =
s.accept()
```

```
print ('Connection address:', addr)


while 1:     data =
conn.recv(BUFFER_SIZE)     if not
data: break     print ("received
data:", data)     conn.send(data) #
echo conn.close()
```

Output:


Activities
Terminal
00:47
imtyaz@imtyaz-VirtualBox: ~/Documents/pythonPRac
File Edit View Search Terminal Help
imtyaz@imtyaz-VirtualBox: ~/Documents/pythonPRac
File Edit View Search Terminal Help

```
(base) imtyazit17017@imtyazit17017:~/Documents/pythonPRac$ sudo python he
llo.py
[sudo] password for sabrin:
('Connection address:', ('10.0.2.15', 59734))
```

```
(base) imtyazit17017@imtyazit17017:~/Documents/pythonPRac$ telnet 10.0.2.
15 80
Trying 10.0.2.15...
Connected to 10.0.2.15.
Escape character is '^]'.
```

Conclusion : Python provides two levels of access to network services. At a coffee level, you'll access the essential socket support within the underlying OS , 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, like FTP, HTTP, and so on. Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on an equivalent machine, or between processes on different continents. To write Internet servers, we use the socket function available in socket module to make a socket object. A socket object is then wont to call other functions to setup a socket server. Now call bind(hostname, port) function to specify a port for our service on the given host. Next, call the accept method of the returned object. This method waits until a client connects to the port we specified, then returns a connection object that represents the connection thereto client.

The END