MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

DEPARTMENT OF ICT

Lab Report No : 06

Course Code : ICT-3208

Course Title : Network Planning and Designing Lab

Lab Report Name : Programming with python

Submitted by	***Submitted to***
Name: Md. Samim Hossain ID : IT-17033 Session : 2016-2017 3rd Year 2nd Semester	Nazrul Islam Assistant Professor, Department of ICT,MBSTU Santosh,Tangail-1902

Programming with Python

Objectives:

- Understand how python function works
- Understand the use of global and local variables
- Understand how python modules works
- Learning the basis of networking programing with python

1.What is Local variable and Global variable ?

Ans: Local variable : Variables declared inside a function definition are not related in any way to other variables with the same names used outside the function (variable names are local to the function). This is called the scope of the variable. All variables have the scope of the block they are declared in starting from the point of definition of the name.

Global variable : Variables defined at the top level of the program are intended global. Global variables are intended to be used in any functions or classes). Global statement allows defining global variables inside functions as well.

2.Describe networking backgrounds for sockets?

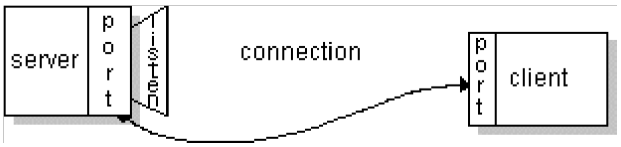
Ans: A socket is one endpoint of a two-way communication link between two programs running on the network or PC. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

Endpoint: An endpoint is a combination of an IP address and a port number.

Server and Client: Normally, a server runs on a specific computer and has a socket that is bound to a specific port number.

- On the server-side: The server just waits, listening to the socket for a client to make a connection request.
- On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs
  to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

3.Define TCP and UDP ?

TCP : TCP stands for transmission control protocol. It is implemented in the transport layer of the IP/TCP model and is used to establish reliable connections. TCP is one of the

protocols that encapsulate data into packets. It then transfers these to the remote end of the connection using the methods available on the lower layers. On the other end, it can check for errors, request certain pieces to be resent, and reassemble the information into one logical piece to send to the application layer.

The protocol builds up a connection prior to data transfer using a system called a three-way handshake. This is a way for the two ends of the communication to acknowledge the request and agree upon a method of ensuring data reliability. After the data has been sent, the connection is torn down using a similar four-way handshake.

TCP is the protocol of choice for many of the most popular uses for the internet, including WWW, FTP, SSH, and email. It is safe to say that the internet we know today would not be here without TCP.

UDP : UDP stands for user datagram protocol. It is a popular companion protocol to TCP and is also implemented in the transport layer.

The fundamental difference between UDP and TCP is that UDP offers unreliable data transfer. It does not verify that data has been received on the other end of the connection. This might sound like a bad thing, and for many purposes, it is. However, it is also extremely important for some functions.

Because it is not required to wait for confirmation that the data was received and forced to resend data, UDP is much faster than TCP. It does not establish a connection with the remote host, it simply fires off the data to that host and doesn't care if it is accepted or not. Because it is a simple transaction, it is useful for simple communications like querying for network resources. It also doesn't maintain a state, which makes it

great for transmitting data from one machine to many real-time clients. This makes it ideal for VOIP, games, and other applications that cannot afford delays.

4.Describe python functions, local and global variables and modules ?

Ans: Defining functions: Functions are defined using the def keyword. After this keyword comes an identifier name for the function, followed by a pair of parentheses which may enclose some names of variables, and by the final colon that ends the line.

```
def XX_YY(variable1, varible2):

    # block belonging to the function
    # End of function
```

Defining local and global variables: Local and global variables can be defined using:

```
x = 50 #Local global x
```

Defining modules: There are various methods of writing modules, but the simplest way is to create a file with a .py extension that contains functions and variables.

```
def xx_yy():
    aa
```

Using modules: A module can be imported by another program to make use of its functionality. This is how we can use the Python standard library as well.

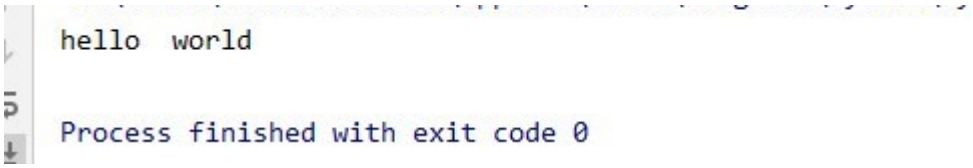
```
import xx_yy
```

5. Create python program using the syntax provided below ?

Source code :

```
def  say_hello():
  print('hello  world')
 if
__name__=='__main__':
    say_hello()    #  call  the  function
```

Output:



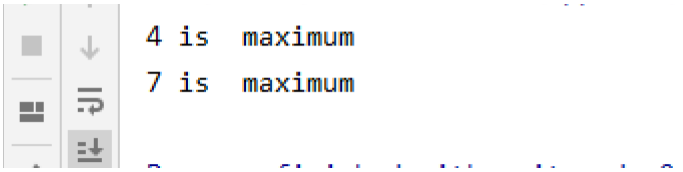
6. Create python script using the syntax provided below ?

Source code :

```
def  print_max(a,  b):
    if  a  >  b:
        print(a,  'is  maximum')
elif  a  ==  b:
        print(a,  'is  equal  to',  b)
else:
        print(b, 'is  maximum')
 if
__name__=='__main__':
    print_max(3, 4)
x = 5
```

```
y = 7
print_max(x,  y)
```

Output:

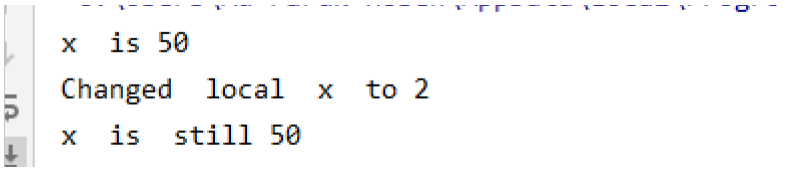


7. Create python script using the syntax provided below ? Which is the final value of variable x?

Source code :

```
x = 50 def func(x):
print('x   is',   x)
x = 2
    print('Changed  local  x  to',  x)
 if
__name__=='__main__':
func(x)
    print('x  is  still',  x)
```

Output:



8. Python modules:

Create python script using the syntax provided below (save as mymodule.py).

```
def  say_hi():
    print('Hi,  this  is  mymodule  speaking.')
__version__= '0.1'
```

Create python script using the syntax provided below (save as module_demo.py).

```
import  mymodule
```

```
 if
__name__=='__main__':
mymodule.say_hi()
    print('Version',  mymodule.__version__    )
```

Output:

```
Hi,  this  is  mymodule  speaking.
Version 0.1
```

Run the script, which is the role of import?

Create python scrip using the syntax provided below (save as module_demo2.py).

```
from  mymodule  import  say_hi,__version__
 if
__name__=='__main__':
say_hi()
    print('Version',__version__)
```

Output:

```
Hi,  this  is  mymodule  speaking.
Version 0.1
```

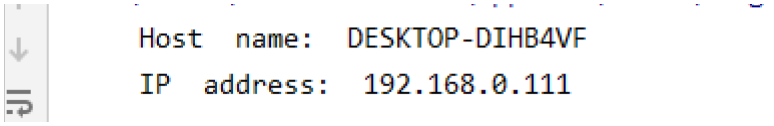
Run the script, which is the role of from, import?

9. Sockets, IPv4, and Simple Client/Server Programming:

Ex-1: Printing your machine's name and IPv4 address ?

```
import  socket
 def
print_machine_info():
    host_name  =  socket.gethostname()
ip_address  =  socket.gethostbyname(host_name)
print  ("  Host  name:  %s"  %  host_name)
print  ("  IP  address:  %s"  %  ip_address)
if __name__=='__main__':
print_machine_info()
```

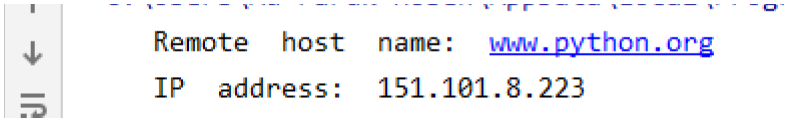
Output:



Ex-2 : Retrieving a remote machine's IP address

```
import socket
 def
get_remote_machine_info():
    remote_host = 'www.python.org'     try:         print
("  Remote  host  name:  %s"  %  remote_host)         print
("  IP  address:  %s"  %socket.gethostbyname(remote_host))
except  socket.error  as  err_msg:         print  ("Error
accesing  %s:  error  number  and  detail
%s"%(remote_host,  err_msg))
 if
__name__=='__main__':
    get_remote_machine_info()
```

Output :



Ex-3 : Converting an IPv4 address to different formats

```
import socket
from binascii import hexlify
 def
convert_ip4_address():
    for ip_addr in ['127.0.0.1', '192.168.0.1']:
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))
 if
__name__=='__main__':
convert_ip4_address()
```

Output:

```
IP Address: 192.168.0.1 => Packed: b'c0a80001', Unpacked: 192.168.0.1
```

Ex-4 : Finding a service name, given the port and protocol ?

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

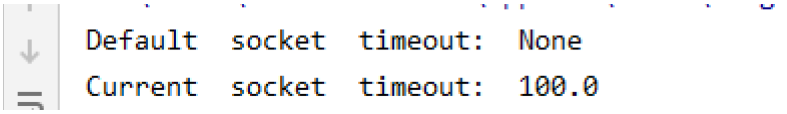
Output:

```
Port: 80 => service name: http
Port: 53 => service name: domain
Port: 25 => service name: smtp
Port: 53 => service name: domain
```

Ex-5 : Setting and getting the default socket timeout

```
import socket
 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())
 if
__name__=='__main__':
test_socket_timeout()
```

Output :



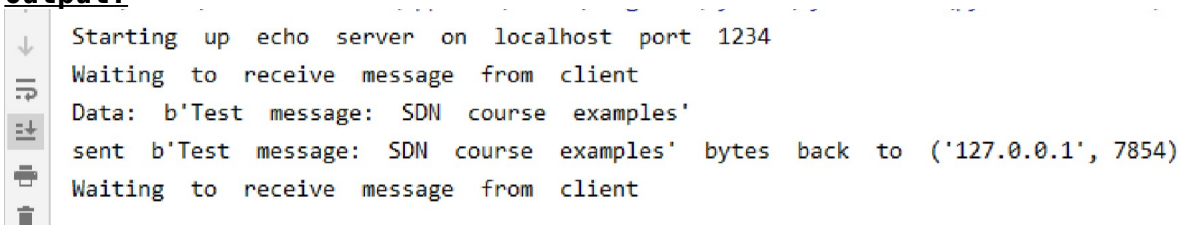
Ex-6 : Writing a simple echo client/server application?

Create python scrip using the syntax below (save as **echo_server.py**):

```
import
socket import
sys import
argparse
import codecs

from codecs import encode,
decode host = 'localhost'
data_payload = 4096 backlog = 5
  def
echo_server(port):
    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=False)
    given_args = parser.parse_args()
port = given_args.port
echo_server(1234)
```

Output:



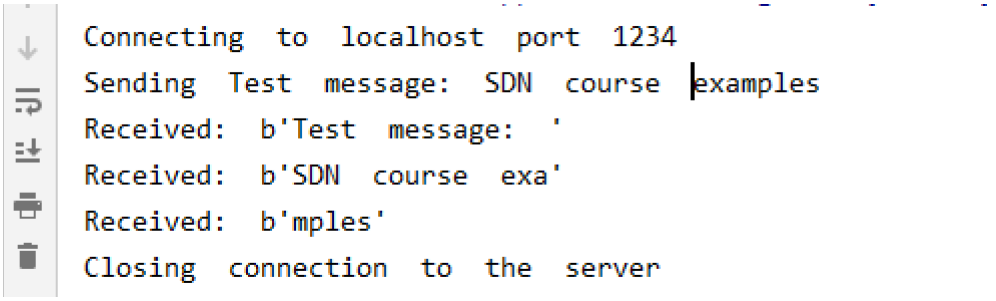
Create python scrip using the syntax below (save as **echo_client.py**):

```
import socket
import sys
import
argparse
import codecs

from codecs import encode,
decode host = 'localhost' def
echo_client(port):
    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)     try:
        message = "Test message: SDN course examples"
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(16)
amount_received      +=      len(data)
print("Received:       %s"    %    data)
except      socket.errno      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=False)
given_args = parser.parse_args()
```

```
port  =  given_args.port
echo_client(1234)
```

Output :



Conclusion :

From this lab, I learn how python function works and how python module works. I also learn the basis of networking programming.I face some problems when I do the echo_server and echo_client code. But I solve these problem with the help of my Course teacher(Nazrul Islam Sir).