



# **Mawlana Bhashani Science & Technology University**

Lab Report No : 02

Course Code : ICT3208

Course Title : Computer Network Lab

Report Name : Programming with python

## **Submitted by**

Name : Maskur Al Shal Sabil

Id: IT18021

3<sup>rd</sup> year 2<sup>nd</sup> Semester

Session 2017-2018

Dept of ICT

MBSTU

## **Submitted To**

Nazrul Islam

Assistant Professor

Dept of ICT

MBSTU

## Theory

**Python functions:** Functions are reusable pieces of programs. They allow you to give a name to a block of statements, allowing you to run that block using the specified name anywhere in the program and any number of times. This is known as calling the function.

**Local Variables:** 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.

**The global statement:** 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.

**Modules:** Modules allow reusing a number of functions in other programs.

### Networking background for sockets

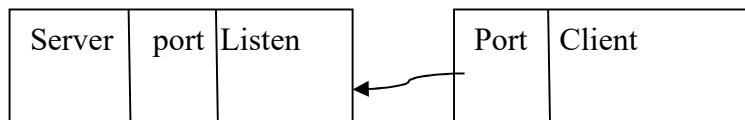
What is a socket and how use it? A socket is one endpoint of a two-way communication link between two programs running on the network or PC. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

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

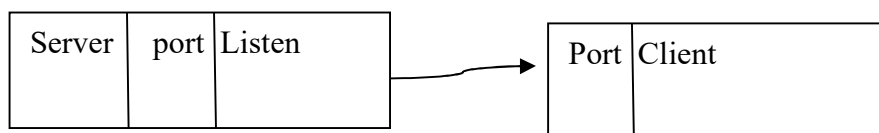
Server and Client: Normally, a server runs on a specific computer and has a socket that is bound to a specific port number.

□ **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

- **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.

### 3. Methodology

**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, variable2):  
# 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
```

## 4. Exercises

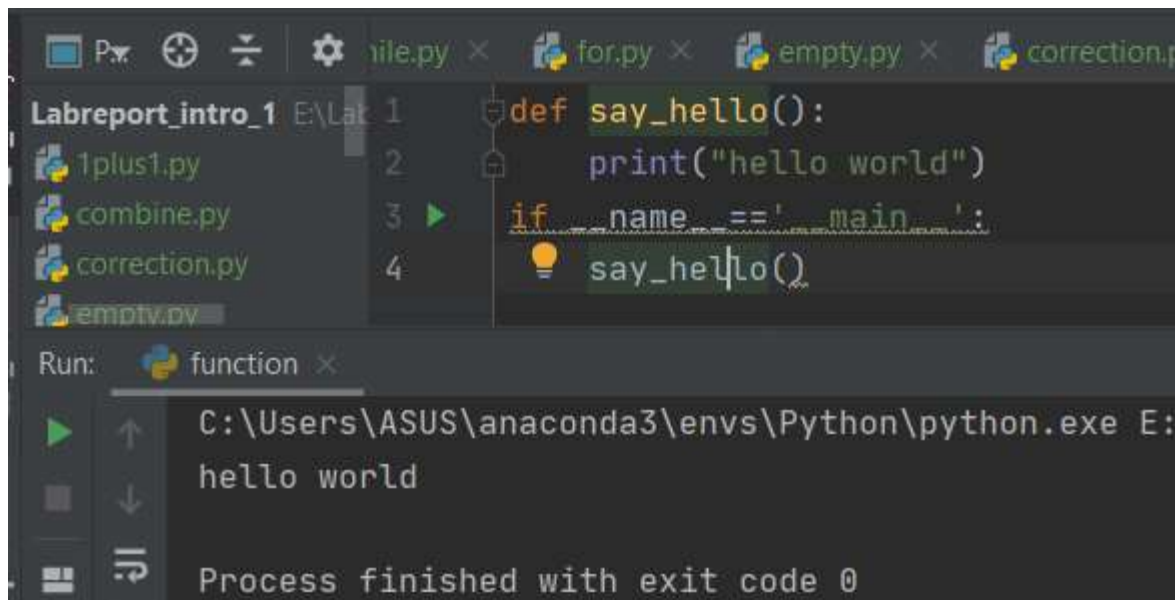
**Section 4.1:** Python function variables and modules.

- **Exercise 4.1.1:** Create a python project using with SDN\_LAB
- **Exercise 4.1.2:** Python function (save as function.py)

Create python scrip using the syntax provided below.

```
def say_hello():  
# block belonging to the function  
print('hello world')  
# End of function  
if __name__ == '__main__':  
say_hello()
```

Which is the output of this function? Does the function need any parameter?

A screenshot of a Python IDE interface. The top part shows a file explorer with several files: 'Labreport\_intro\_1', '1plus1.py', 'combine.py', 'correction.py', and 'empty.py'. The main editor window displays a Python script with the following code:

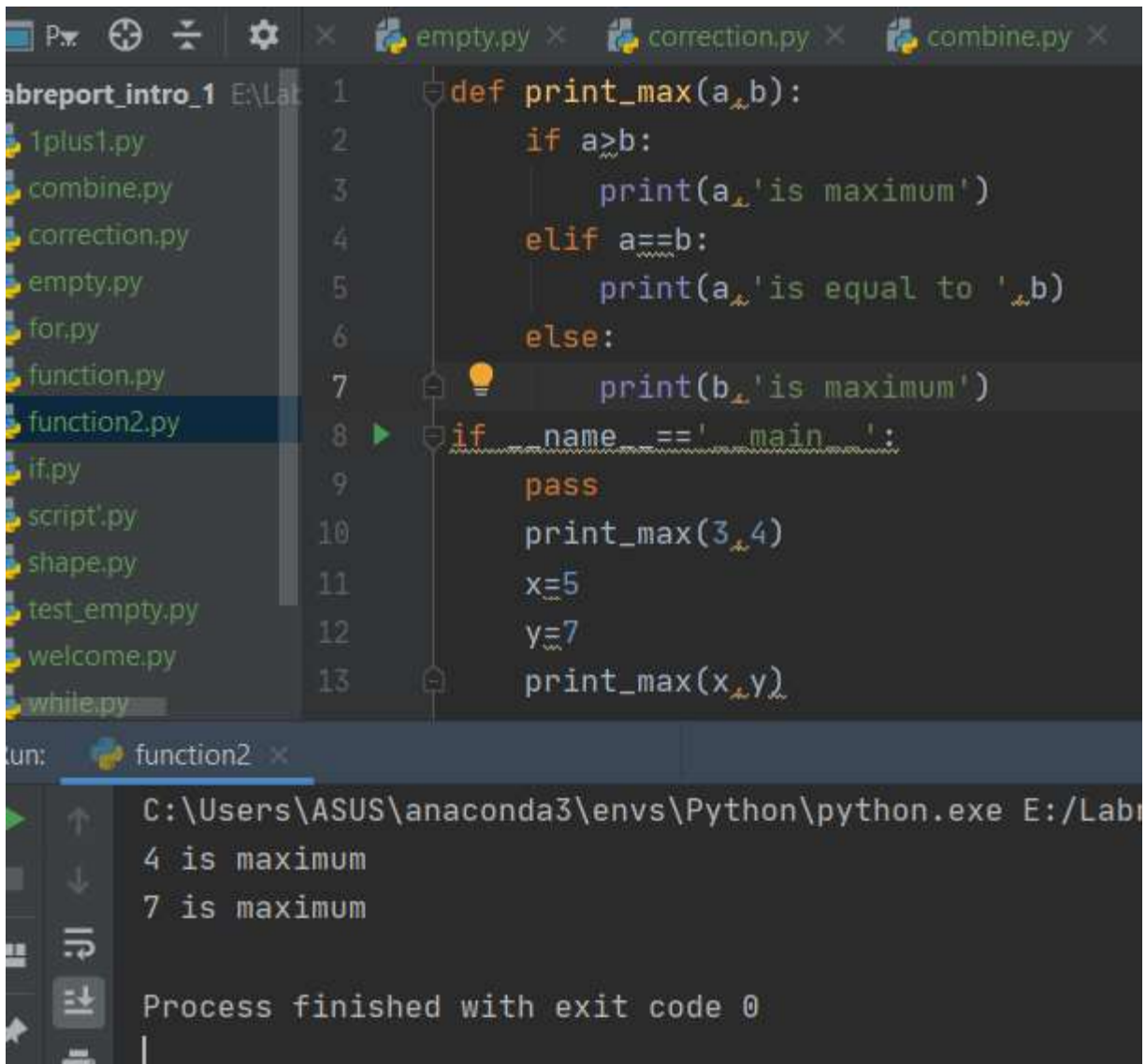
```
def say_hello():  
    print("hello world")  
if __name__ == '__main__':  
    say_hello()
```

The bottom part of the IDE shows a 'Run' panel with the command 'C:\Users\ASUS\anaconda3\envs\Python\python.exe E:' and the output 'hello world'. Below the output, it says 'Process finished with exit code 0'.

The function print 'hellow world' . It doesn't need any parameter.

**Exercise 4.1.3:** Python function (save as function\_2.py)

Create python scrip using the syntax provided below. Which is the output of this function? Does the function need any parameter?



The screenshot shows a Python IDE with a file explorer on the left and a code editor on the right. The file explorer lists several files, including 'function2.py' which is selected. The code editor displays the following Python code:

```
1 def print_max(a, b):
2     if a > b:
3         print(a, 'is maximum')
4     elif a == b:
5         print(a, 'is equal to', b)
6     else:
7         print(b, 'is maximum')
8 if __name__ == '__main__':
9     pass
10    print_max(3, 4)
11    x = 5
12    y = 7
13    print_max(x, y)
```

Below the code editor, a terminal window shows the output of the program:

```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Lab
4 is maximum
7 is maximum
Process finished with exit code 0
```

Here the function takes two parameter and the output is : 4 is maximum and 7 is maximum.

**Exercise 4.1.4:** Local variable (save as function\_local.py) Create python scrip using the syntax provided below . Which is the final value of variable x? Why variable x does not change to 2?

The screenshot shows a Python IDE with a file explorer on the left containing files like Labreport\_intro\_1, 1plus1.py, combine.py, correction.py, empty.py, for.py, function.py, function2.py, and function\_local.py. The main editor displays the code in function\_local.py:

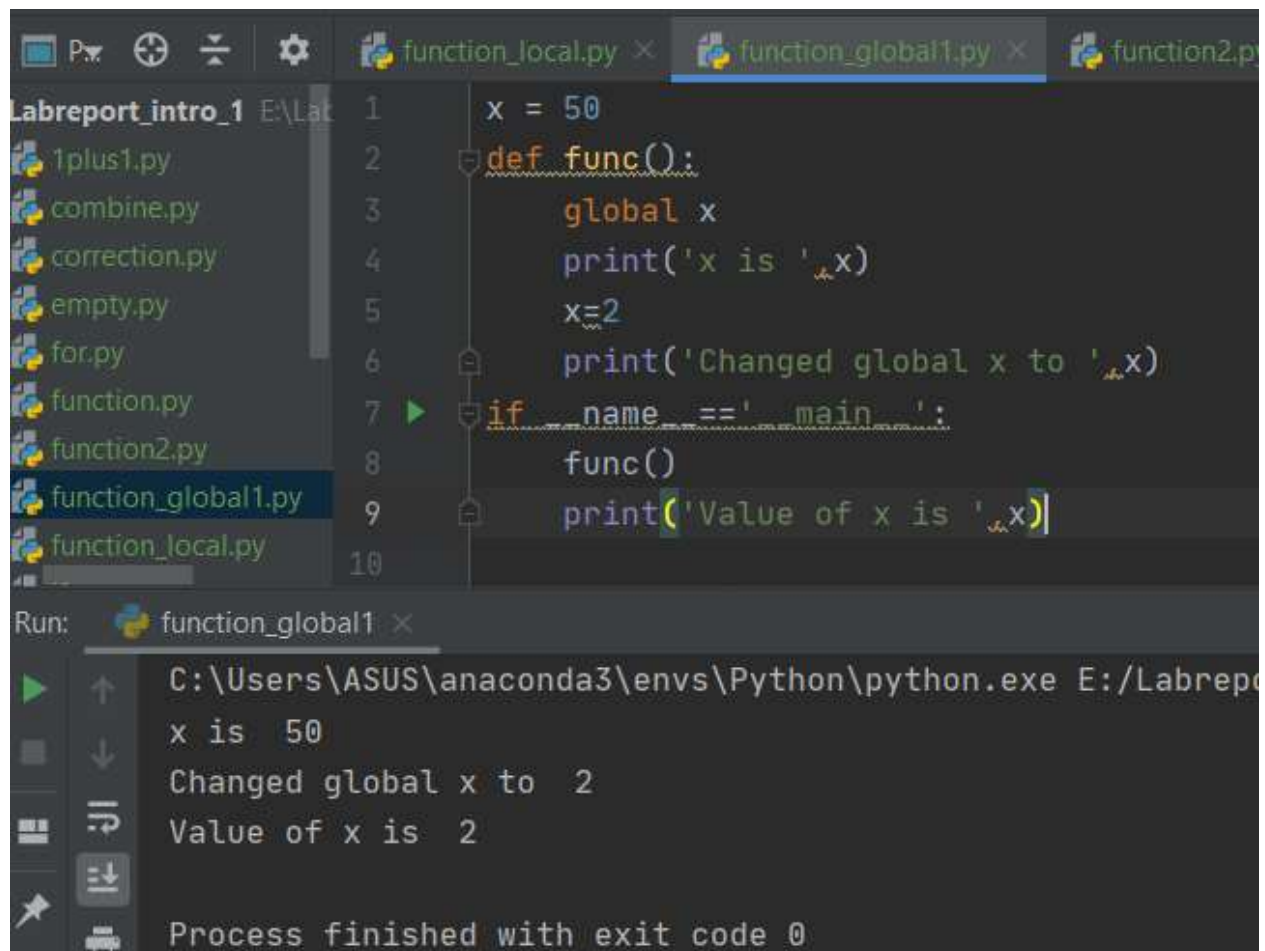
```
1 x = 50
2 def func(x):
3     print('x is ', x)
4     x=2
5     print('Changed local x to ', x)
6 if __name__ == '__main__':
7     func(x)
8     print('x is still', x)
```

The Run console at the bottom shows the output of the script:

```
Run: function_local x
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/L
x is 50
Changed local x to 2
x is still 50
```

The final value of x is 50 . The value of x doesn't change because it is a local variable and it's scope is under the function .

- **Exercise 4.1.5:** Global variable (save as function\_global.py) Create python scrip using the syntax provided below. Which is the final value of variable x? Why variable x change this time?



```
function_global1.py
1  x = 50
2  def func():
3      global x
4      print('x is ', x)
5      x=2
6      print('Changed global x to ', x)
7  if __name__ == '__main__':
8      func()
9      print('Value of x is ', x)
```

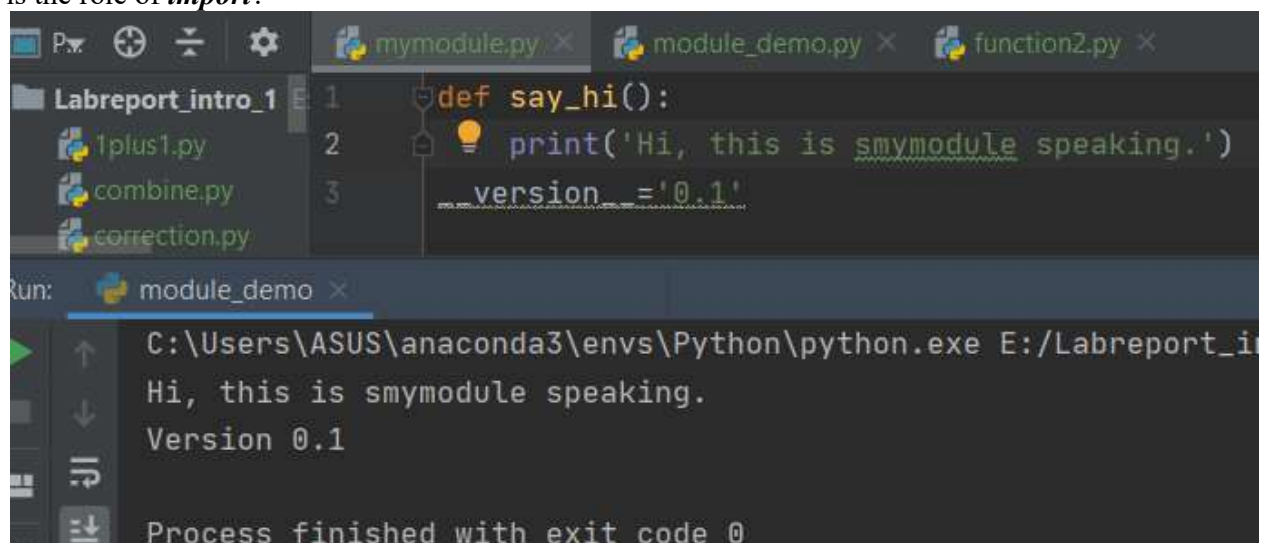
Run: function\_global1

```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport_intro_1/function_global1.py
x is 50
Changed global x to 2
Value of x is 2
Process finished with exit code 0
```

The final value of x is 2 . The value of x changed because of its global declaration.

• **Exercise 4.1.6:** Python modules

Create python script using the syntax provided below (save as mymodule.py). Run the script, which is the role of *import*?

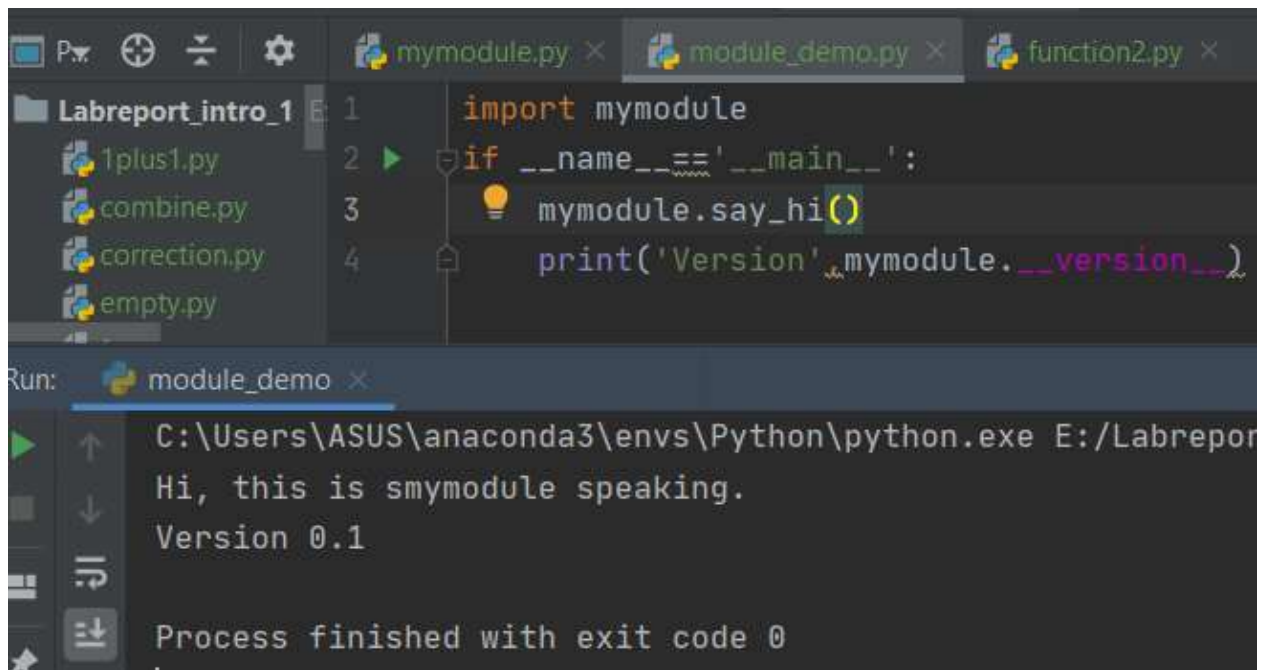


```
mymodule.py
1  def say_hi():
2      print('Hi, this is smymodule speaking.')
3      version_='0.1'
```

Run: module\_demo

```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport_intro_1/mymodule.py
Hi, this is smymodule speaking.
Version 0.1
Process finished with exit code 0
```





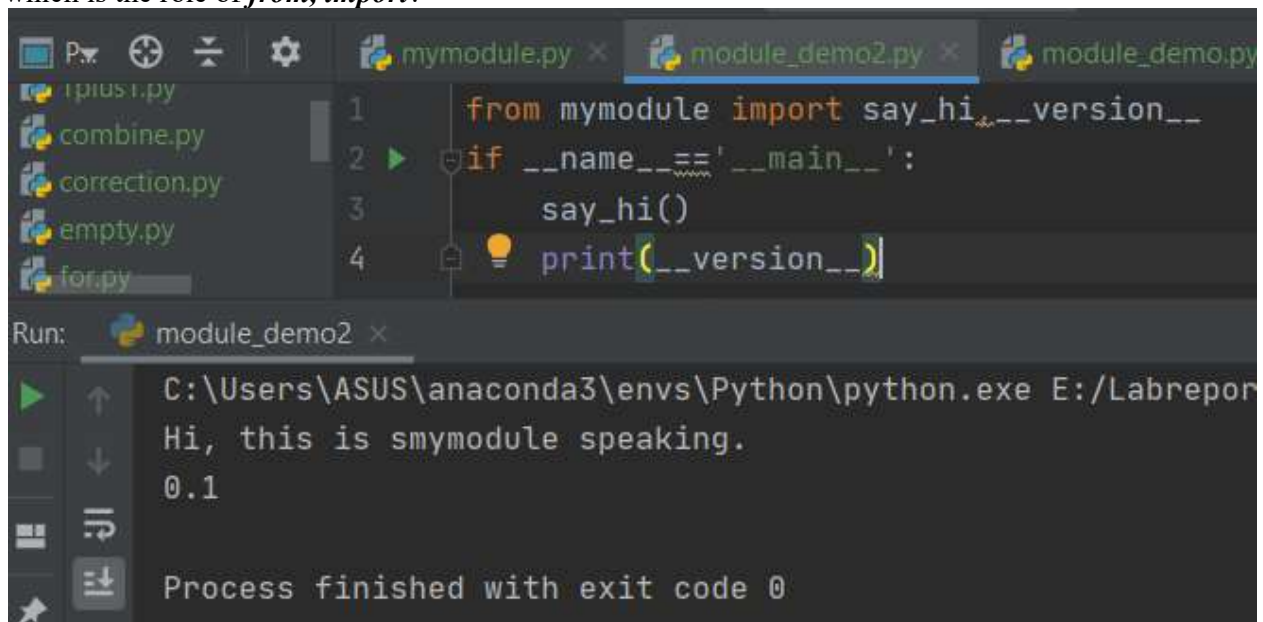
```
1 import mymodule
2 if __name__ == '__main__':
3     mymodule.say_hi()
4     print('Version', mymodule.__version__)
```

Run: module\_demo x

```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport
Hi, this is smymodule speaking.
Version 0.1
Process finished with exit code 0
```

The role of import is that inherit the properties of mymdoule. Thus we can use the component of this module.

Create python scrip using the syntax provided below (save as module\_demo2.py). Run the script, which is the role of *from, import*?



```
1 from mymodule import say_hi, __version__
2 if __name__ == '__main__':
3     say_hi()
4     print(__version__)
```

Run: module\_demo2 x

```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport
Hi, this is smymodule speaking.
0.1
Process finished with exit code 0
```

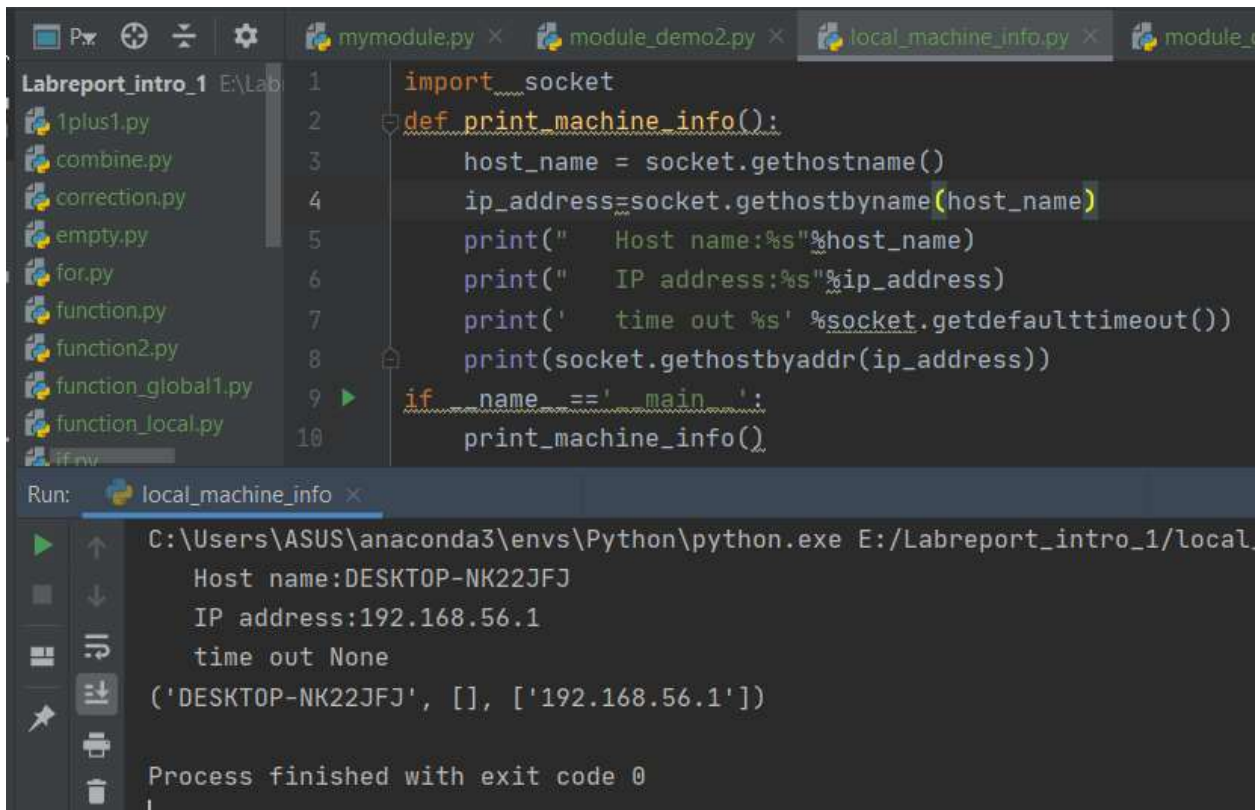
The role of using from , import is that we can now directly use the say\_hi() and \_\_version\_\_

## Section 4.2: Sockets, IPv4, and Simple Client/Server Programming

### □ Exercise 4.2.1: Printing your machine's name and IPv4 address

Create python scrip using the syntax provided below (save as local\_machine\_info.py): Run the script, which module the program uses? Provide two additional functions of socket?





The screenshot shows an IDE with a file explorer on the left containing various Python files. The main editor displays a script named `local_machine_info.py`. The script imports the `socket` module and defines a function `print_machine_info()` that prints the local host name, IP address, and timeout. A main block calls this function. The Run window at the bottom shows the execution output for `local_machine_info.py`.

```
1 import socket
2 def print_machine_info():
3     host_name = socket.gethostname()
4     ip_address = socket.gethostbyname(host_name)
5     print("    Host name:%s"%host_name)
6     print("    IP address:%s"%ip_address)
7     print('    time out %s' %socket.getdefaulttimeout())
8     print(socket.gethostbyaddr(ip_address))
9 if __name__ == '__main__':
10     print_machine_info()
```

Run: local\_machine\_info.py

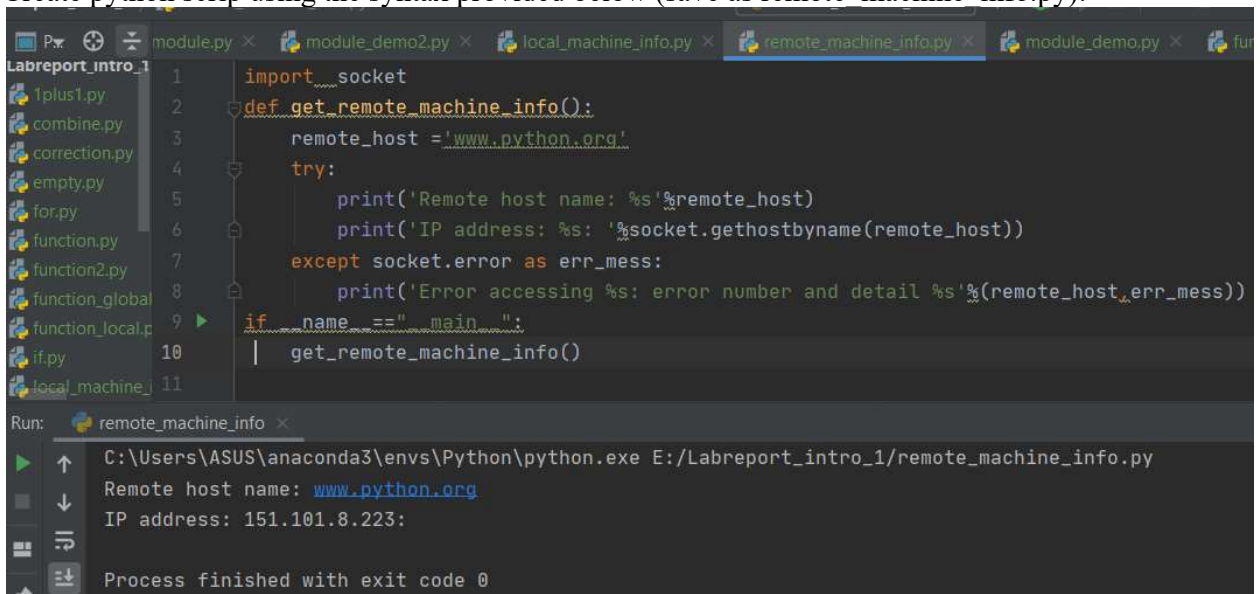
```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport_intro_1/local_
Host name:DESKTOP-NK22JFJ
IP address:192.168.56.1
time out None
('DESKTOP-NK22JFJ', [], ['192.168.56.1'])

Process finished with exit code 0
```

The program use the socket module .

#### Exercise 4.2.2: Retrieving a remote machine's IP address

Create python scrip using the syntax provided below (save as `remote_machine_info.py`):



The screenshot shows the same IDE with a new script named `remote_machine_info.py` open. The script imports the `socket` module and defines a function `get_remote_machine_info()` that attempts to retrieve the IP address of a remote host. It includes a try-except block to handle potential socket errors. A main block calls this function. The Run window shows the execution output for `remote_machine_info.py`.

```
1 import socket
2 def get_remote_machine_info():
3     remote_host = 'www.python.org'
4     try:
5         print('Remote host name: %s'%remote_host)
6         print('IP address: %s: '%socket.gethostbyname(remote_host))
7     except socket.error as err_mess:
8         print('Error accessing %s: error number and detail %s'%(remote_host,err_mess))
9 if __name__ == '__main__':
10     get_remote_machine_info()
```

Run: remote\_machine\_info.py

```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport_intro_1/remote_machine_info.py
Remote host name: www.python.org
IP address: 151.101.8.223:

Process finished with exit code 0
```

Modify the code for getting the RMIT website info.

The screenshot shows a Python IDE with a file explorer on the left and a code editor. The file explorer lists files like 1plus1.py, combine.py, correction.py, empty.py, for.py, function.py, function2.py, function\_global1.py, function\_local.py, if.py, local\_ma, module\_, module\_, mymodu, remote\_, and script.py. The code editor displays the following Python code:

```
1 import socket
2 def get_remote_machine_info():
3     remote_host = 'www.rmit.edu.au'
4     try:
5         print('Remote host name: %s'%remote_host)
6         print('IP address: %s: '%socket.gethostbyname(remote_host))
7     except socket.error as err_mess:
8         print('Error accessing %s: error number and detail %s'%(remote_host,err_mess))
9 if __name__ == "__main__":
10     get_remote_machine_info()
11
```

Below the code editor, the 'Run' panel shows the execution of the script 'remote\_machine\_info.py'. The output is:

```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport_intro_1/remote_machine_info.py
Remote host name: www.rmit.edu.au
IP address: 54.79.133.82:
Process finished with exit code 0
```

• **Exercise 4.2.3:** Converting an IPv4 address to different formats

Create python scrip using the syntax below (save as ip4\_address\_conversion.py): Run the script, which is the output? How *binascii* works?

The screenshot shows a Python IDE with a file explorer on the left and a code editor. The file explorer lists files like 1plus1.py, combine.py, correction.py, empty.py, for.py, function.py, function2.py, function\_global1.py, function\_local.py, if.py, local\_ma, module\_, module\_, mymodu, remote\_, and script.py. The code editor displays the following Python code:

```
1 import socket
2 from binascii import hexlify
3 def convert_ip4_address():
4     for ip_addr in ['127.0.0.1', '192.168.0.1']:
5         packed_ip_addr = socket.inet_aton(ip_addr)
6         unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
7         print('Ip Address: %s => Packed: %s,'
8               ' Unpacked : %s'%(ip_addr,hexlify(packed_ip_addr),unpacked_ip_addr))
9 if __name__ == '__main__':
10     convert_ip4_address()
```

Below the code editor, the 'Run' panel shows the execution of the script 'ip4\_address\_conversion.py'. The output is:

```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport_intro_1/ip4_address_conversion.py
Ip Address: 127.0.0.1 => Packed: b'7f000001', Unpacked : 127.0.0.1
Ip Address: 192.168.0.1 => Packed: b'c0a80001', Unpacked : 192.168.0.1
Process finished with exit code 0
```

Binascii is a module which contains a number of methods to convert between binary and various ASCII-encoded binary representation . So using binascii we are able to use hexlify() function to convert the binary data to hexadecimal representation.

**Exercise 4.2.4:** Finding a service name, given the port and protocol . Create python scrip using the syntax below (save as finding service name.py):

The screenshot shows a Python IDE with a project named 'Labreport\_intro\_1'. The file 'finding\_service\_name.py' is open and contains the following code:

```
1 import socket
2 def find_service_name():
3     protocolname = 'tcp'
4     for port in [80,25]:
5         print("port : %s +> service name: "
6             "%s"%(port,socket.getservbyport(port,protocolname)))
7     print('Port : %s => service name: %s'
8         %(53,socket.getservbyport(53,'udp')))
9 if __name__ == '__main__':
10    find_service_name()
```

The Run window shows the output of the script:

```
port : 80 +> service name: http
port : 25 +> service name: smtp
Port : 53 => service name: domain
Process finished with exit code 0
```

Run the script, which is the output? Modify the code for getting complete the table:

Port	Protocol Name
21	
22	
110	

The screenshot shows the same Python IDE with the modified 'finding\_service\_name.py' script. The code is now:

```
1 import socket
2 def find_service_name():
3     protocolname = 'tcp'
4     for port in [21,22,110]:
5         print("port : %s +> service name: "
6             "%s"%(port,socket.getservbyport(port,protocolname)))
7     print('Port : %s => service name: %s'
8         %(53,socket.getservbyport(53,'udp')))
9 if __name__ == '__main__':
10    find_service_name()
```

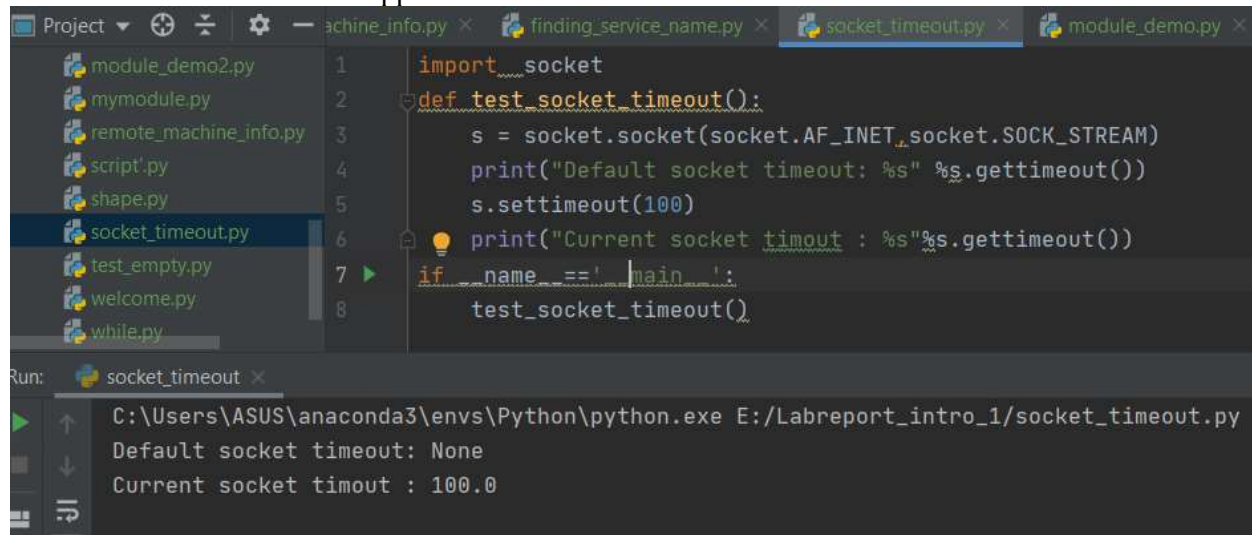
The Run window shows the updated output:

```
port : 21 +> service name: ftp
port : 22 +> service name: ssh
port : 110 +> service name: pop3
Port : 53 => service name: domain
```

Port	Protocol Name
21	ftp
22	ssh
110	Pop3

### Exercise 4.2.5: Setting and getting the default socket timeout

Create python scrip using the syntax below (save as socket\_timeout.py): Run the script, which is the role of socket timeout in real applications?



```
Project ▾ ⚙ ⚡ ⚙ — achine_info.py × finding_service_name.py × socket_timeout.py × module_demo.py ×
module_demo2.py 1 import socket
mymodule.py 2 def test_socket_timeout():
remote_machine_info.py 3     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
script'.py 4     print("Default socket timeout: %s" %s.gettimeout())
shape.py 5     s.settimeout(100)
socket_timeout.py 6     print("Current socket timeout : %s"%s.gettimeout())
test_empty.py 7 if __name__ == '__main__':
welcome.py 8     test_socket_timeout()
while.py

Run: socket_timeout ×
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport_intro_1/socket_timeout.py
Default socket timeout: None
Current socket timeout : 100.0
```

The role of socket timeout : A *socket timeout* is the timeout when waiting for individual packets.

When a client request a server there is a response time but it is silence that's why it is hanging . In order to avoid the annoying problem of waiting for an undetermined amount of time, sockets (which are responsible for network communication) support a *timeout* option which raises an error based on a time limit

### Exercise 4.2.6: Writing a simple echo client/server application (**Tip:** Use port 9900)

Create python scrip using the syntax below (save as echo\_server.py):

- Run the script, which is the output?

- ☐ What you need to do for running the program?
- ☐ Which program you need to run first client of server?

First I need to run the echo\_server program and then echo\_client.

- ☐ Explain how the program works?

The server program run on the port 9900 and then it wait for client request .On the other side client run and it request the server program using the same port . In the server program we need to bind the port and host . And their establish a communication .



The image shows a Python IDE with a project named 'Labreport\_intro\_1'. The file explorer on the left lists various Python files, including 'echo\_client.py' and 'echo\_server.py'. The main editor displays the code for 'echo\_server.py', which is a simple echo server. The code imports 'socket', 'sys', 'argparse', and 'codecs', then defines a function 'echo\_server(port)' that sets up a socket, binds it to 'localhost' on the specified port, and enters a loop to accept and echo client messages. The script is executed, and the console output shows the server starting on port 9900 and waiting for a client connection.

```
1 import socket
2 import sys
3 import argparse
4 import codecs
5 from codecs import encode, decode
6 host = 'localhost'
7 data_payload = 4096
8 backlog = 5
9 def echo_server(port):
10     """A simple echo server"""
11     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12     sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
13     server_address = (host, port)
14     print("Starting up echo server on %s port %s" % server_address)
15     sock.bind(server_address)
16     sock.listen(backlog)
17     while True:
18         print("Waiting to receive message from client")
19         client, address = sock.accept()
20         data = client.recv(data_payload)
21         if data:
22             print("Data : %s" % data)
23             client.send(data)
24             print("sent %s bytes back to %s" % (data, address))
25         client.close()
26 if __name__ == '__main__':
27     echo_server(9900)
```

Run: server x echo\_server x echo\_client x

Starting up echo server on localhost port 9900  
Waiting to receive message from client

```
echo_server.py x echo_client.py x socket_server.py x socket_client.py x mo
1 import socket
2 import sys
3 import argparse
4 import codecs
5 from codecs import encode, decode
6 host = 'localhost'
7
8 def echo_client(port):
9     """A simple echo client"""
10    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11    server_address = (host, port)
12    print("Connecting to %s port %s" % server_address)
13    sock.connect(server_address)
14    try:
15        message = "Tesst message : SDN course examples"
16        print("sending %s" % message)
17        sock.sendall(message.encode('utf_8'))
18        amount_received = 0
19        amount_expected = len(message)
20        while amount_received < amount_expected:
21            data = sock.recv(16)
22            amount_received += len(data)
23            print("Received : %s" % data)
24        except socket.error as e:
25            print("Socket error : %s" % str(e))
26        except Exception as e_:
27            print("Other exception : %s" % str(e))
28        finally:
29            print("Closing connection to the server")
30            sock.close()
31 if __name__ == "__main__":
32     echo_client(9900)
33
```

Run: server x echo\_server x echo\_client x

```
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport_intro_1/echo_client.py
Connecting to localhost port 9900
Sending Tesst message : SDN course examples
Received : b'Tesst message : '
Received : b'SDN course examp'
Received : b'les'
Closing connection to the server

Process finished with exit code 0
```

□ What you need to do for communicating with another server in the classroom?

To communicate with another server of the classroom we can use the ip address of that computer and make sure that both computer are in Local network . so using the local network ip we can communicate with each other.

## 5. Questions

▣ **Question 5.1:** Explain in your own words which are the difference between functions and modules?

The modules can be used in the different program where a function is in a single program . To use function in different program it must be included under a module. A module may have many function and variable that can be use by declaring using from keyword

□ **Question 5.2:** Explain in your own words when to use local and global variables?

The local variable is use for temporary use which value may change and same within a certain scope but global variable all time takes a common value in all the program .

□ **Question 5.3:** Which is the role of sockets in computing networking? Are the sockets defined random or there is a rule?

The socket is a way to bind the client side and server side so that the communication between two or more nodes can be possible. It is the endpoint of the server and client . Y

Yes the socket follow the specific rule to establish the connection.

▣ **Question 5.4:** Why is relevant to have the IPv4 address of remote server? Explain what is Domain Name System (DNS)?

Simply put to communicate via the internet we need a source and destination address and a way to get from the source to destination. So every device , end point whatever is connected has an assigned Ip address as a way to identify our source and destination. The internet can then be used to route packets of data between source and destination ( including diversions) . IPv4, was created back in the 1970s. Vint Cerf led the team who invented it. It however is made up of four 8-bit numbers or 32-bits total, providing for 4.3 billion addresses. It also support DNS .

The Domain Name System (DNS) supports both IPV4 and IPV6s. The DNS stores the IP addresses for either or both and responds to each request for domain name resolution with both IP addresses (A site can have multiple addresses for either protocol).

DNS puts IPv4 addresses into the A record. The DNS system stores IPv6 addresses in the AAAA record.

The client can then decide which protocol to use.

▣ **Question 5.5:** Create a program that allows exchange messages increased by the user between client and server.

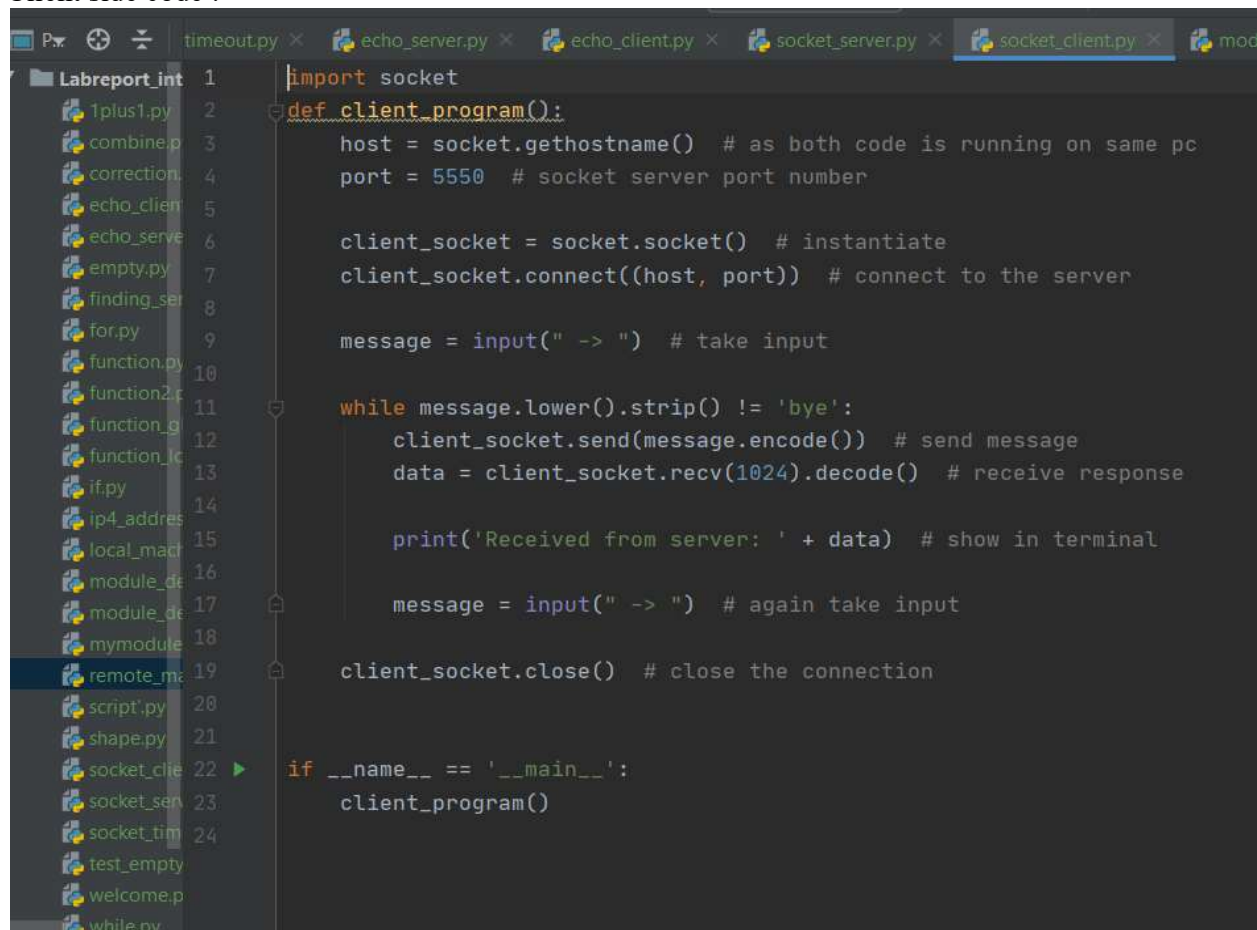


Server side code :

```
finding_service_name.py x socket_timeout.py x echo_server.py x echo_client.py x socket_server.py
1 import socket
2 def server_program():
3     # get the hostname
4     host = socket.gethostname()
5     print('host name', host)
6     port = 5550 # initiate port no above 1024
7
8     server_socket = socket.socket() # get instance
9     # look closely. The bind() function takes tuple as argument
10    server_socket.bind((host, port)) # bind host address and port together
11
12    # configure how many client the server can listen simultaneously
13    server_socket.listen(2)
14    conn, address = server_socket.accept() # accept new connection
15    print("Connection from: " + str(address))
16    while True:
17        # receive data stream. it won't accept data packet greater than 1024
18        data = conn.recv(1024).decode()
19        if not data:
20            # if data is not received break
21            break
22        print("from connected user: " + str(data))
23        data = input(' -> ')
24        conn.send(data.encode()) # send data to the client
25
26    conn.close() # close the connection
27
28
29 if __name__ == '__main__':
30     server_program()
```

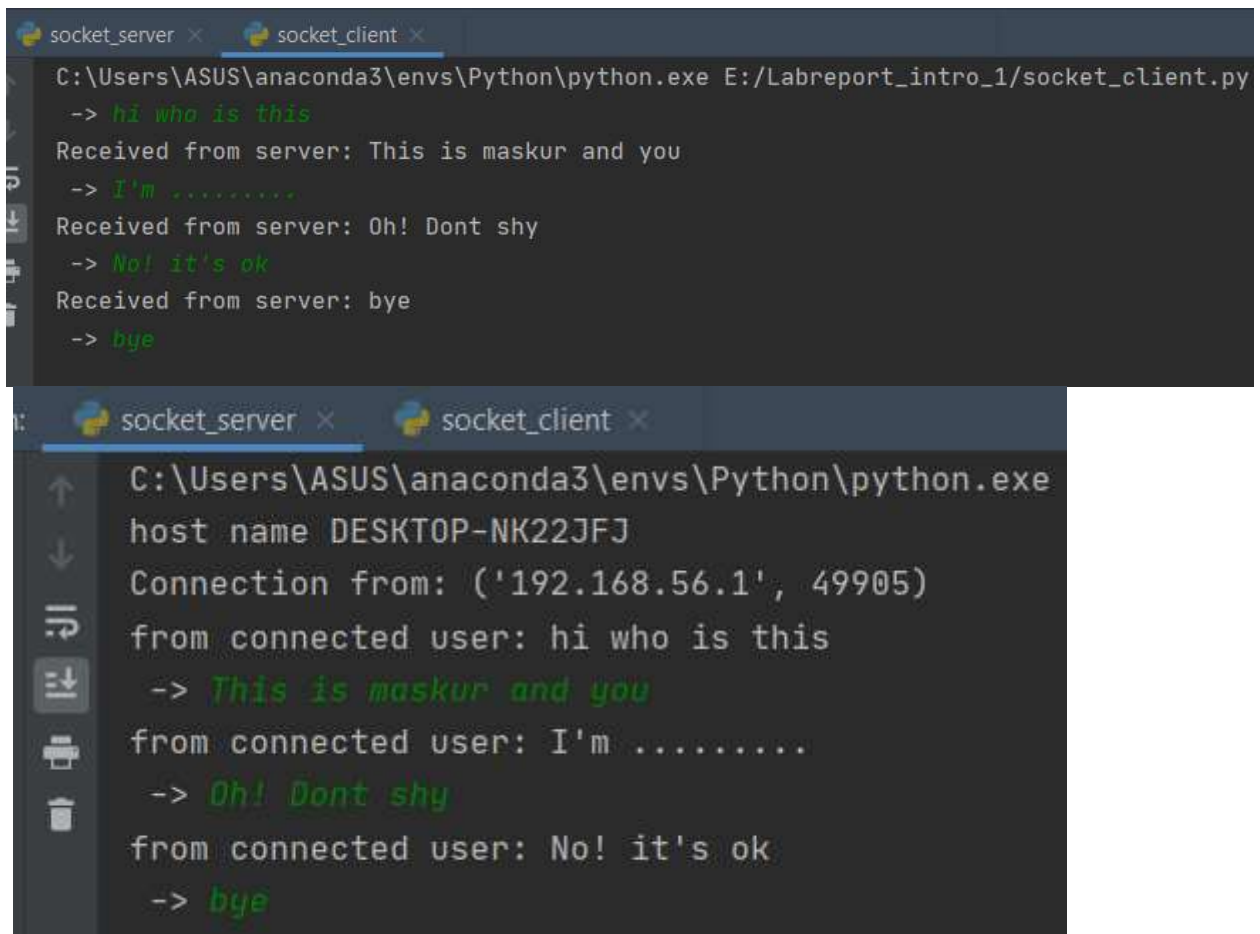
PyCharm 2020.1.4 available

Client side code :



```
1 import socket
2
3 def client_program():
4     host = socket.gethostname() # as both code is running on same pc
5     port = 5550 # socket server port number
6
7     client_socket = socket.socket() # instantiate
8     client_socket.connect((host, port)) # connect to the server
9
10    message = input(" -> ") # take input
11
12    while message.lower().strip() != 'bye':
13        client_socket.send(message.encode()) # send message
14        data = client_socket.recv(1024).decode() # receive response
15
16        print('Received from server: ' + data) # show in terminal
17
18        message = input(" -> ") # again take input
19
20    client_socket.close() # close the connection
21
22 if __name__ == '__main__':
23     client_program()
```

The output :



The image contains two screenshots of a Python socket client and server interaction. The top screenshot shows the client's perspective, and the bottom screenshot shows the server's perspective.

**Top Screenshot (Client):**

```
socket_server x socket_client x
C:\Users\ASUS\anaconda3\envs\Python\python.exe E:/Labreport_intro_1/socket_client.py
-> hi who is this
Received from server: This is maskur and you
-> I'm .....
Received from server: Oh! Dont shy
-> No! it's ok
Received from server: bye
-> bye
```

**Bottom Screenshot (Server):**

```
socket_server x socket_client x
C:\Users\ASUS\anaconda3\envs\Python\python.exe
host name DESKTOP-NK22JFJ
Connection from: ('192.168.56.1', 49905)
from connected user: hi who is this
-> This is maskur and you
from connected user: I'm .....
-> Oh! Dont shy
from connected user: No! it's ok
-> bye
```

Conclusion : To do this lab report I have taken help from the slide given by my course teacher .That was so good and I have learned many things about network .Besides I also taken help from Internet resource to construct the last exercise . In this lab report I have learned how to get the ip address of my computer and other host . And communicate a server code and a client code . I have learned how the data transfer from one host to another and I enjoyed this lab . .