



MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

Santosh, Tangail-1902

LAB REPORT

Lab No : 02
Lab name : Programming with Python
Course Title : Network Planning and Designing Lab
Course Code : ICT-3208

Submitted by,
Tanvir Ahmed
IT-18043
Farhana Afrin Shikha
IT-18038
3rd year 2nd semester
Dept. of ICT

Submitted to,
Nazrul Islam
Assistant professor
Dept. of ICT,
MBSTU

Lab 2: Programing with Python

1. Objectives :

The objective of the lab 2 is to:

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

2. 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.

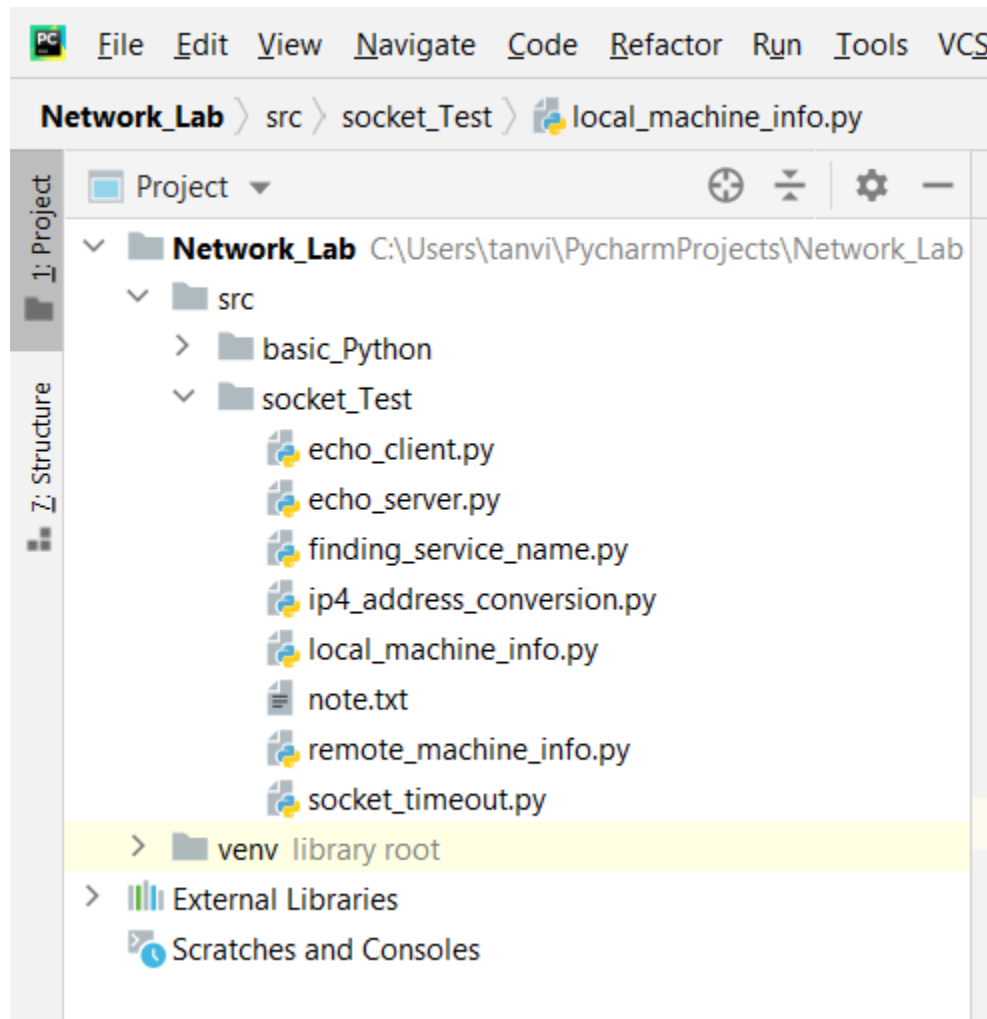
```
import xx_yy
```

4. Exercises

Section 4.1: Python function variables and modules.

Exercise 4.1.1: Create a python project using with SDN_LAB

Answer:

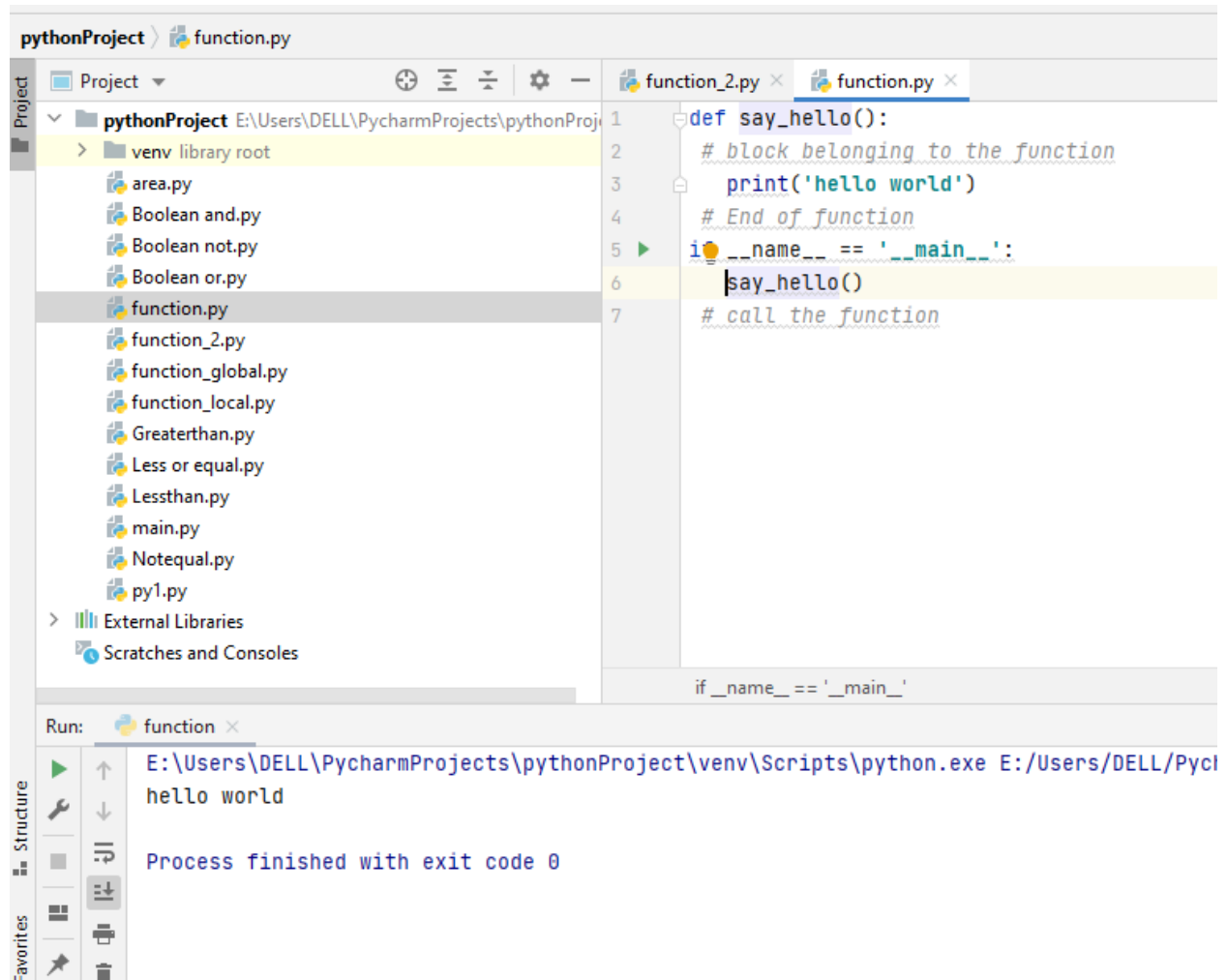


Exercise 4.1.2: Python function (save as function.py)

Create a python script using the syntax provided below.

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

Answer:



The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for adding, removing, and toggling views, as well as a settings gear. The 'Project' view on the left shows a project named 'pythonProject' with a 'venv' directory and a 'library root'. The 'function.py' file is selected in the project view. The main editor window displays the code for 'function.py' with line numbers 1 through 7. The code defines a function 'say_hello()' that prints 'hello world' and calls it within a main block. The 'Run' view at the bottom shows the execution command and the output 'hello world'.

```
pythonProject > function.py
Project
  pythonProject E:\Users\DELL\PycharmProjects\pythonProject
    venv library root
      area.py
      Boolean and.py
      Boolean not.py
      Boolean or.py
      function.py
      function_2.py
      function_global.py
      function_local.py
      Greaterthan.py
      Less or equal.py
      Lessthan.py
      main.py
      Notequal.py
      py1.py
    External Libraries
    Scratches and Consoles

function_2.py function.py
1 def say_hello():
2     # block belonging to the function
3     print('hello world')
4     # End of function
5 if __name__ == '__main__':
6     say_hello()
7     # call the function

if __name__ == '__main__'
```

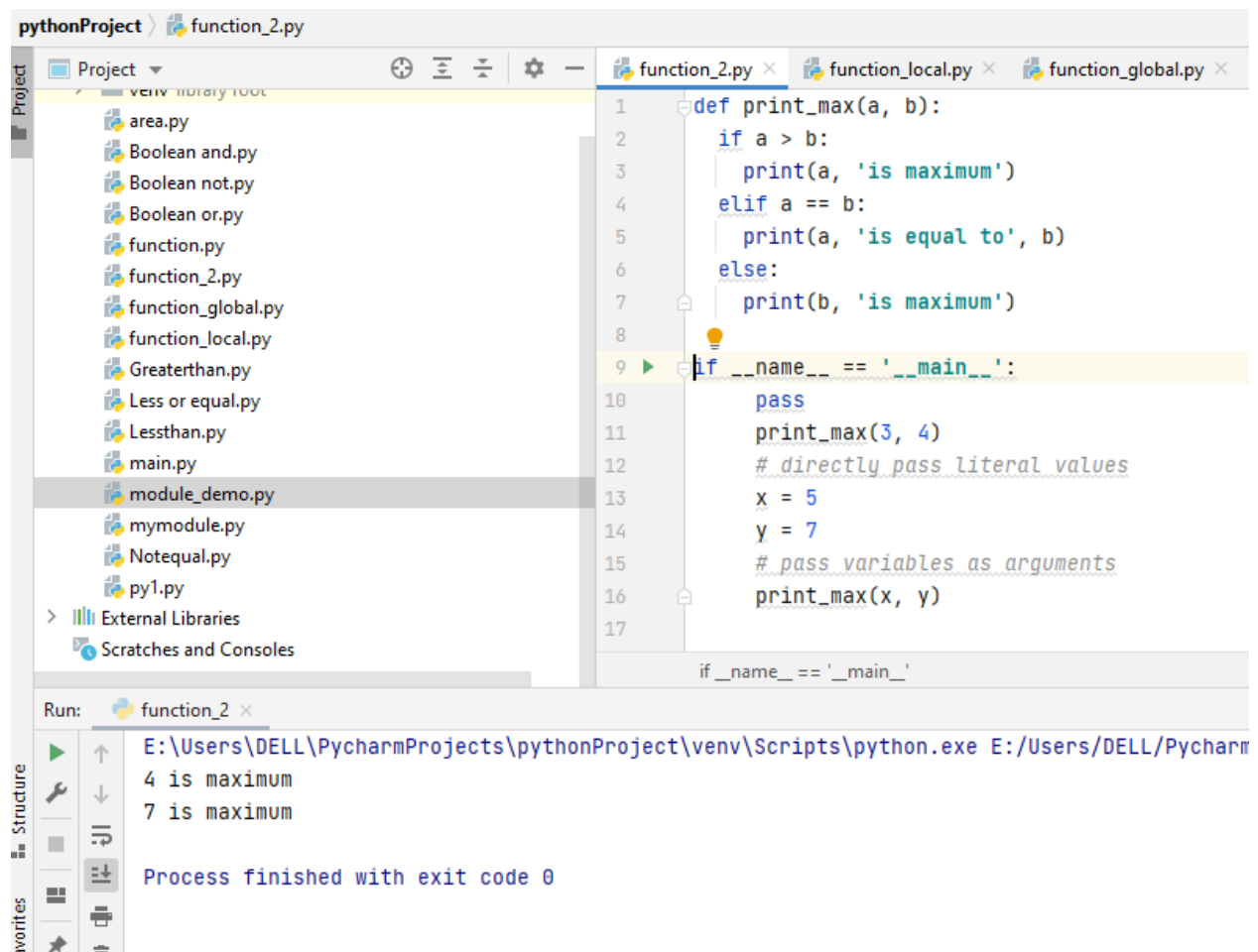
Run: function ×
E:\Users\DELL\PycharmProjects\pythonProject\venv\Scripts\python.exe E:/Users/DELL/PycharmProjects/pythonProject/function.py
hello world
Process finished with exit code 0

Exercise 4.1.3: Python function (save as function_2.py)

Create a python script using the syntax provided below.

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

Answer:

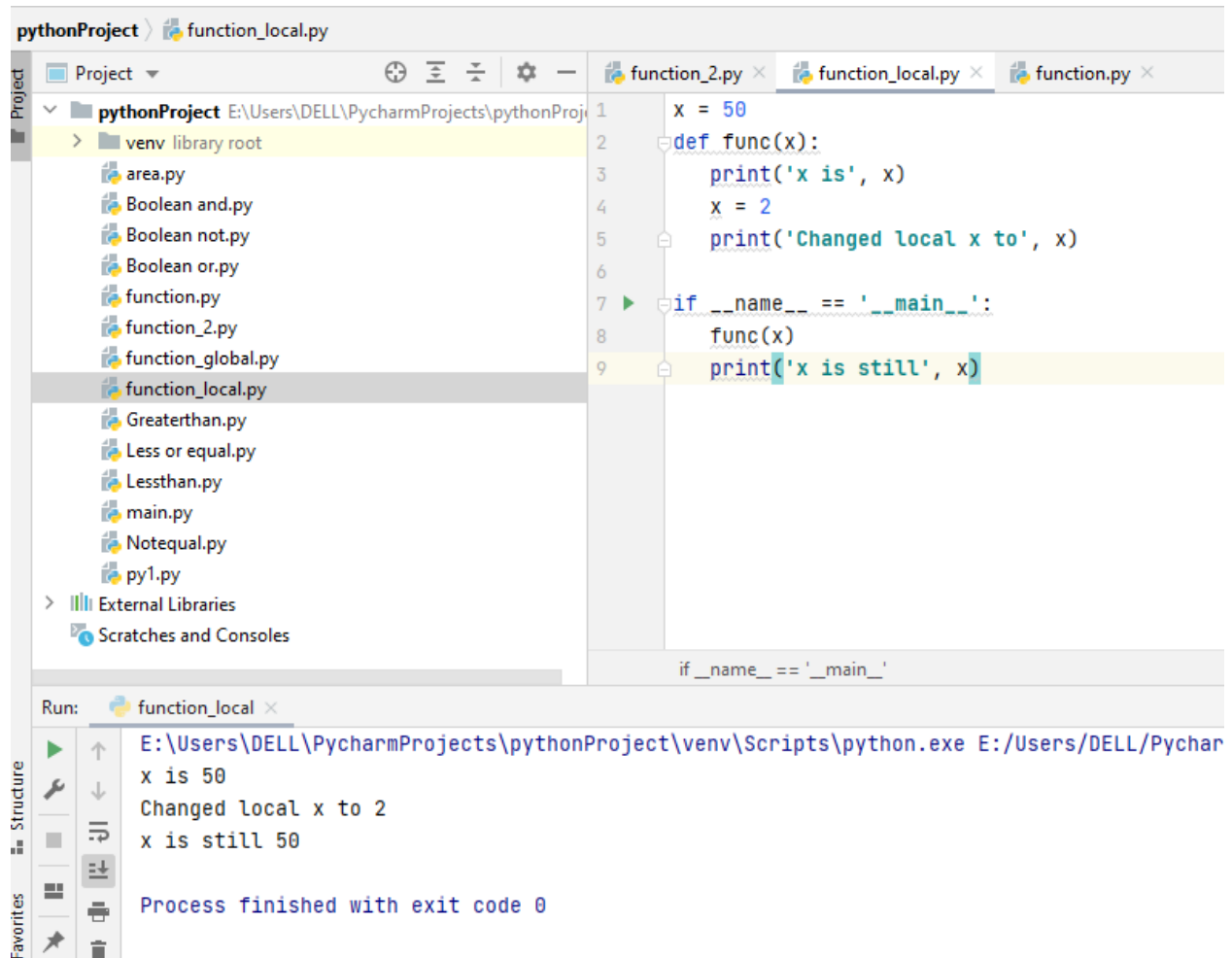


Exercise 4.1.4: Local variable (save as function_local.py)

Create a python script using the syntax provided below.

Which is the final value of variable x? Why does variable x not change to 2?

Answer:

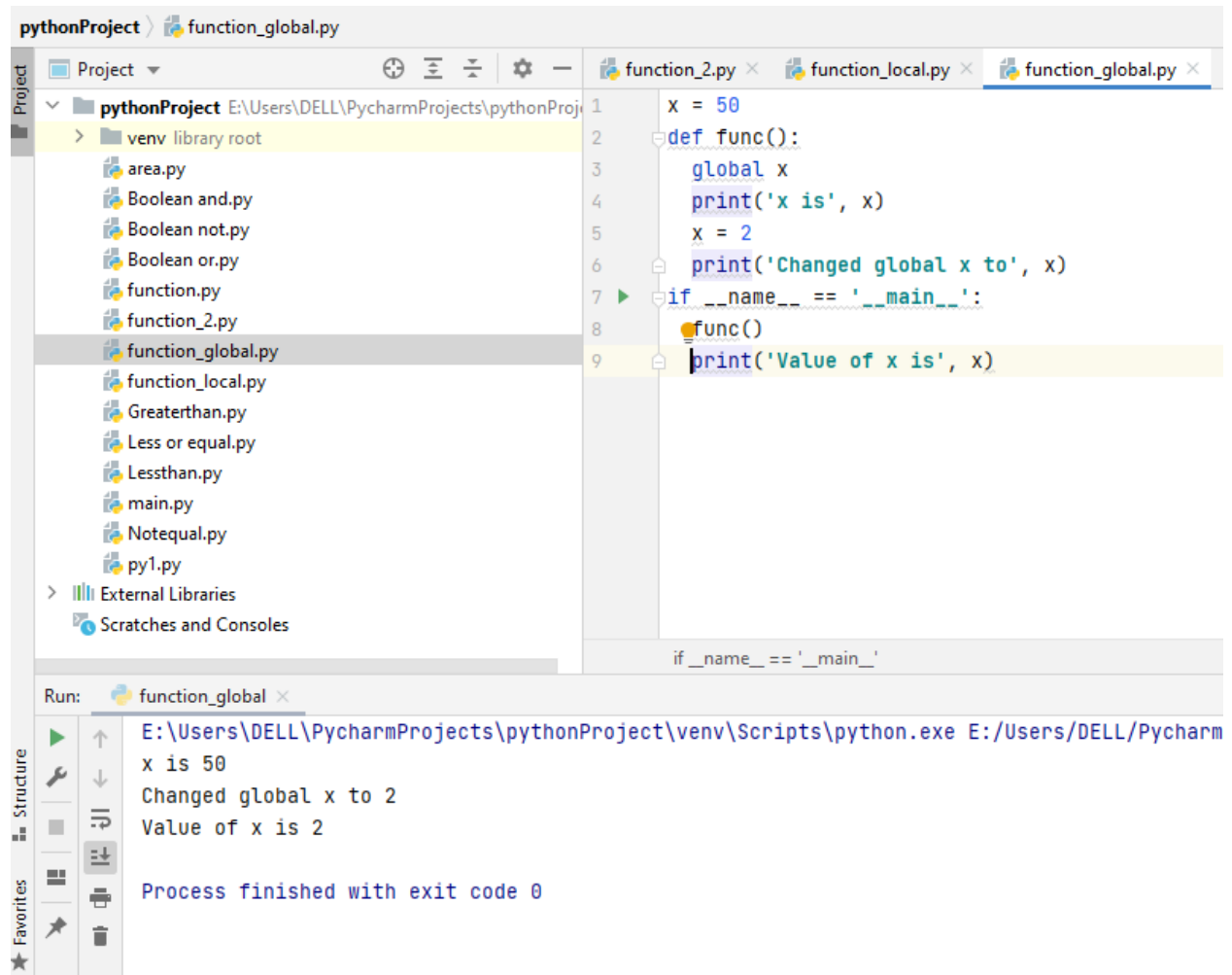


Exercise 4.1.5: Global variable (save as function_global.py)

Create a python script using the syntax provided below.

Which is the final value of variable x? Why variable x change this time?

Answer:



Exercise 4.1.6: Python modules

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

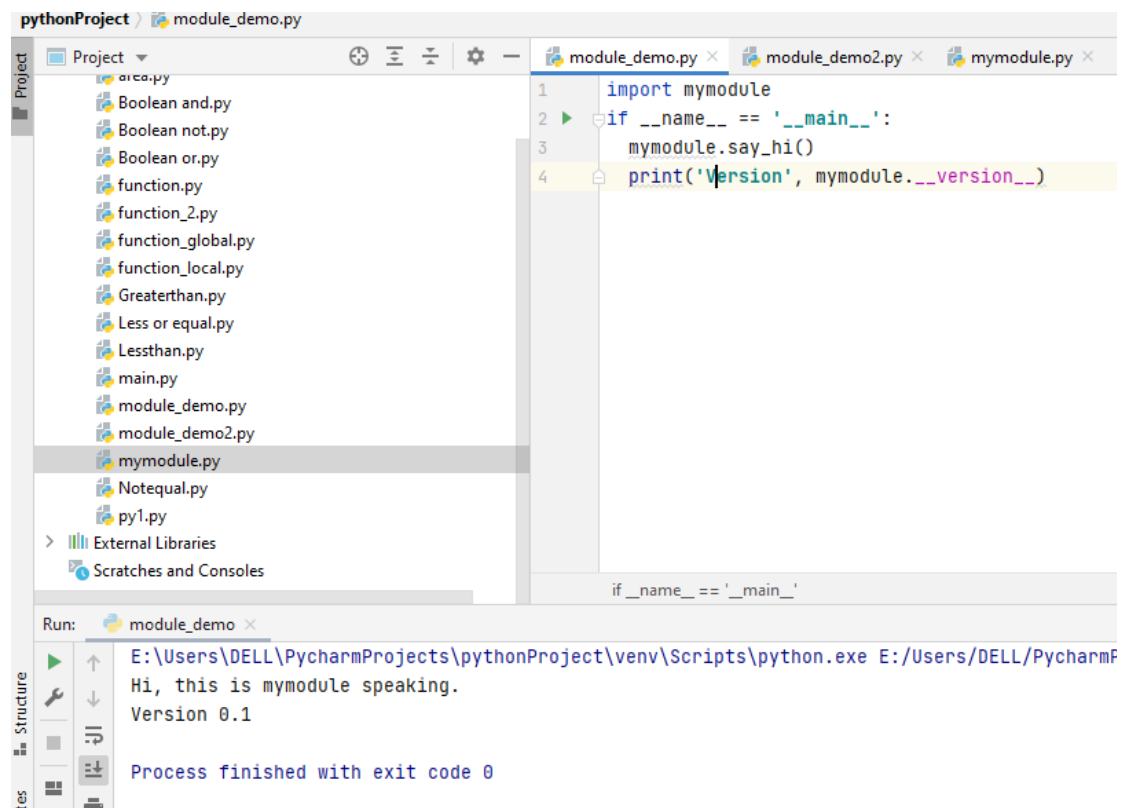
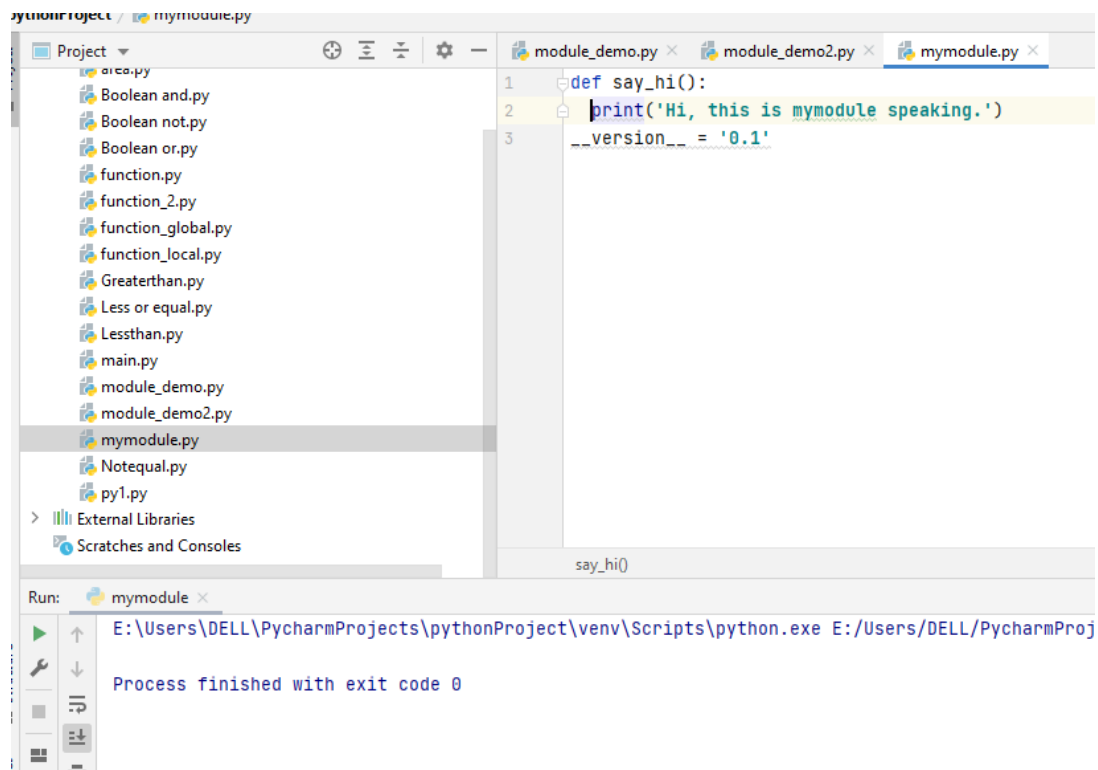
Create python script using the syntax provided below (save as module_demo.py).

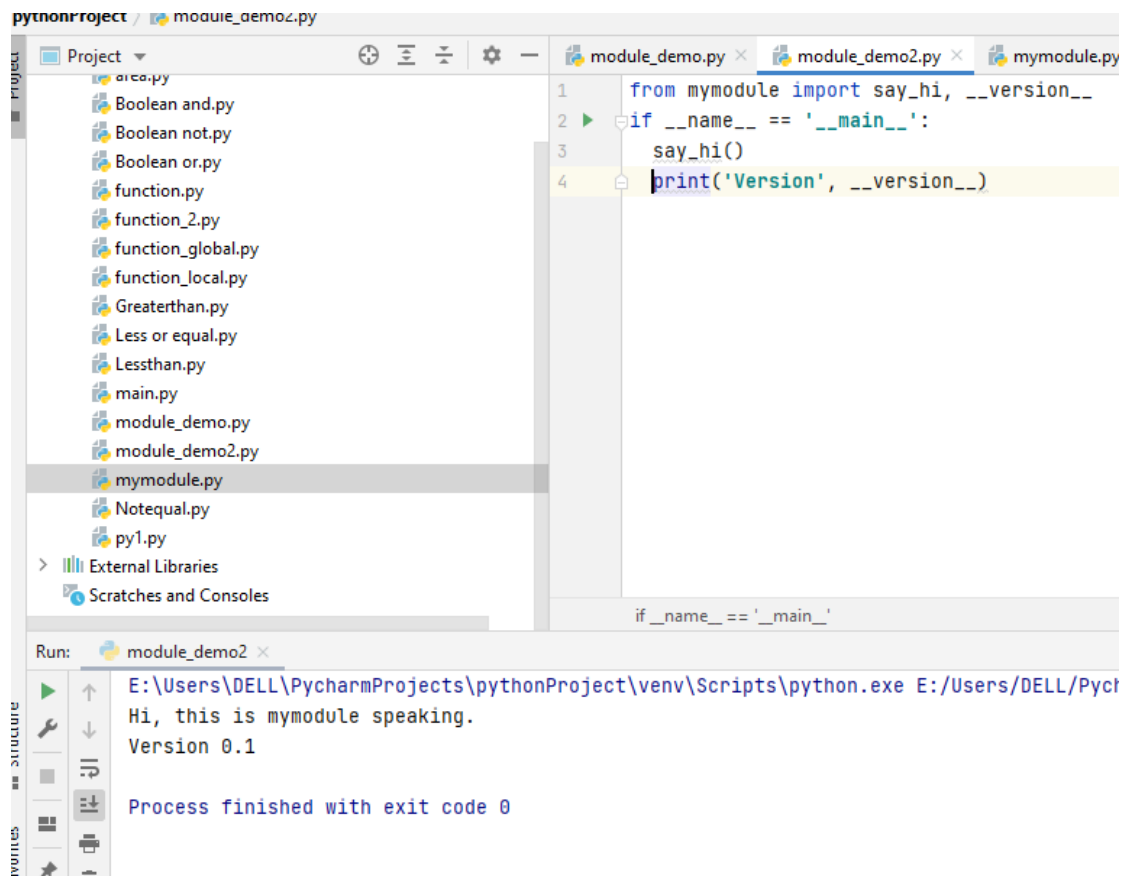
Run the script, which is the role of import?

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

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

Answer:





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

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

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

Answer:

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The breadcrumb navigation shows the path: Network_Lab > src > socket_Test > local_machine_info.py. The left sidebar displays the Project and Structure tool windows. The Project window shows the hierarchy: Network_Lab (C:\Users\tanvi\PycharmProjects) > src > socket_Test > local_machine_info.py. The Structure window shows the file local_machine_info.py. The main editor window displays the code for local_machine_info.py:

```
1 import socket
2
3 def print_machine_info():
4     host_name = socket.gethostname()
5     ip_address = socket.gethostbyname(host_name)
6     print(" Host name: %s" % host_name)
7     print(" IP address: %s" % ip_address)
8
9 if __name__ == '__main__':
10     print_machine_info()
11
12
```

Below the editor, the Run window shows the execution command and output:

```
Run: local_machine_info
C:\Users\tanvi\PycharmProjects\Network_Lab\venv\Scripts\python.exe C:/Users/tanvi/P
Host name: DESKTOP-9CQ7EQ2
IP address: 192.168.56.1
Process finished with exit code 0
```

Run the script, which module the program uses? Provide two additional functions of Socket.

Answer:

The module name is **print_machine_info()**

Two functions are :

- gethostname() , gethostbyname()

Exercise 4.2.2: Retrieving a remote machine's IP address

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

Answer:

The screenshot shows the PyCharm IDE with the file `remote_machine_info.py` open. The code defines a function `get_remote_machine_info()` that uses the `socket` module to retrieve the IP address of `www.python.org`. The `__main__` block calls this function. The Run window at the bottom shows the output: Remote host name: www.python.org and IP address: 151.101.8.223.

```
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_msg:
8         print("Error accesing %s: error number and detail %s" %(remote_host, err_msg))
9
10 if __name__ == '__main__':
11     get_remote_machine_info()
```

Run: remote_machine_info x
C:\Users\tanvi\PycharmProjects\Network_Lab\venv\Scripts\python.exe C:/Users/tanvi/PycharmProjects/Network_Lab/src/socket_test/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.

Answer:

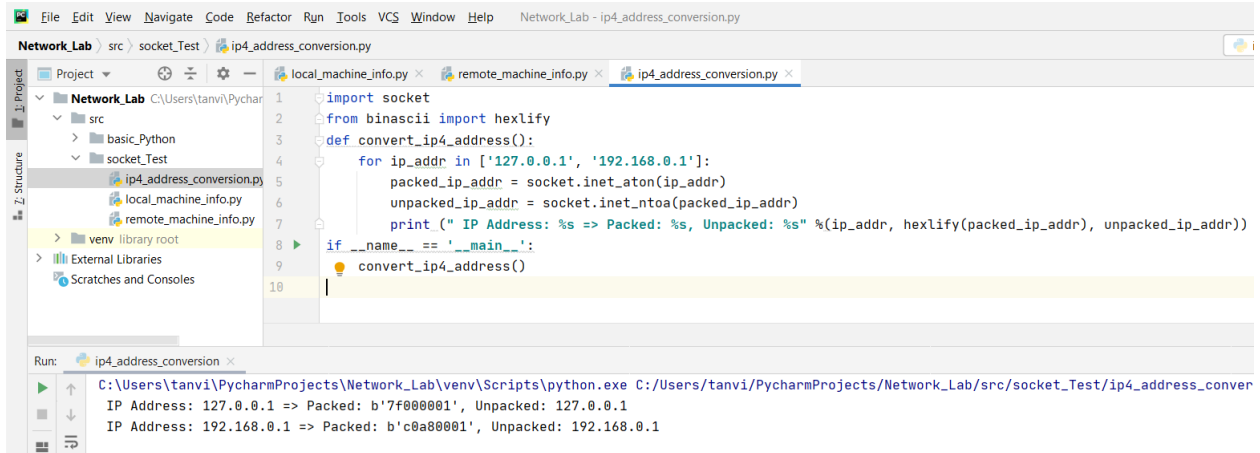
The screenshot shows the PyCharm IDE with the file `local_machine_info.py` open. The code defines a function `print_machine_info()` that uses the `socket` module to retrieve the IP address of `www.rmit.edu.au`. The `__main__` block calls this function. The Run window at the bottom shows the output: Host name: www.rmit.edu.au and IP address: 54.66.185.131.

```
1 import socket
2
3 def print_machine_info():
4     #host_name = socket.gethostname()
5     host_name = "www.rmit.edu.au"
6     ip_address = socket.gethostbyname(host_name)
7     print(" Host name: %s" % host_name)
8     print(" IP address: %s" % ip_address)
9
10 if __name__ == '__main__':
11     print_machine_info()
12
13
```

Run: local_machine_info x
C:\Users\tanvi\PycharmProjects\Network_Lab\venv\Scripts\python.exe C:/Users/tanvi/PycharmProjects/Network_Lab/src/socket_test/local_machine_info.py
Host name: www.rmit.edu.au
IP address: 54.66.185.131

Exercise 4.2.3: Converting an IPv4 address to different formats
Create python script using the syntax below (save as ip4_address_conversion.py):

Answer:



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help Network_Lab - ip4_address_conversion.py
Network_Lab > src > socket_Test > ip4_address_conversion.py
Project: Network_Lab C:\Users\tanvi\PycharmProjects\Network_Lab
  - src
    - basic_Python
    - socket_Test
      - ip4_address_conversion.py
      - local_machine_info.py
      - remote_machine_info.py
  - venv library root
  - External Libraries
  - Scratches and Consoles

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, Unpacked: %s" %(ip_addr, hexlify(packed_ip_addr), unpacked_ip_addr))
8     if __name__ == '__main__':
9         convert_ip4_address()
10

Run: ip4_address_conversion
C:\Users\tanvi\PycharmProjects\Network_Lab\venv\Scripts\python.exe C:/Users/tanvi/PycharmProjects/Network_Lab/src/socket_Test/ip4_address_conver
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
```

How binascii works?

Answer:

The [binascii](#) module contains a number of methods to convert between binary and various ASCII-encoded binary representations. Normally, you will not use these functions directly but use wrapper modules like [uu](#), [base64](#), or [binhex](#) instead. The binascii module contains low-level functions written in C for greater speed that are used by the higher-level modules.

Exercise 4.2.4: Finding a service name, given the port and protocol

Create python script using the syntax below (save as finding_service_name.py):

Answer:

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

Run: finding_service_name

C:\Users\tanvi\PycharmProjects\Network_Lab\venv\Scripts\python.exe C:/Users/tanvi/PycharmProjects/Network_Lab/src/socket_Tes

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

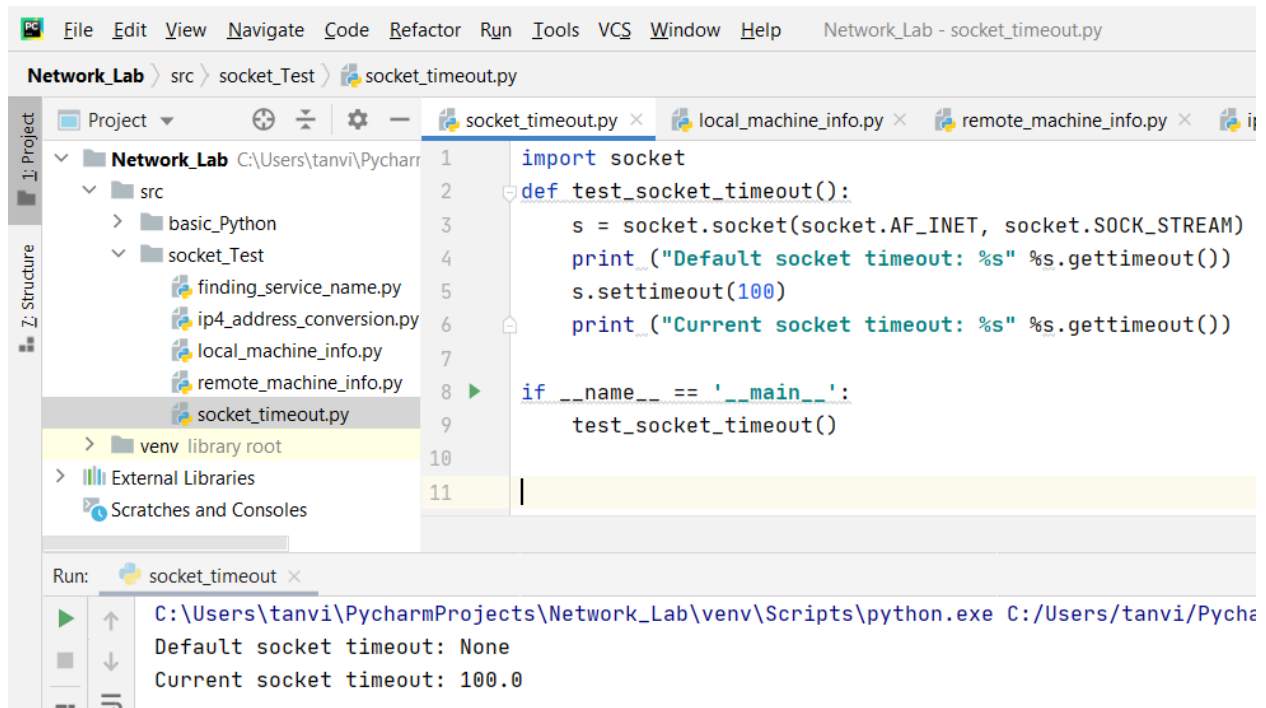
Answer:

Port	Protocol Name
21	ftp
22	ssh
110	pop3

Exercise 4.2.5: Setting and getting the default socket timeout

Create python script using the syntax below (save as socket_timeout.py):

Answer:



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The title bar indicates the project is 'Network_Lab - socket_timeout.py'. The left sidebar shows the Project and Structure toolbars. The Project toolbars shows the 'Network_Lab' project structure with folders 'src' and 'socket_Test'. The Structure toolbars shows the file 'socket_timeout.py' selected. The main editor window displays the following Python code:

```
1 import socket
2 def test_socket_timeout():
3     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4     print("Default socket timeout: %s" %s.gettimeout())
5     s.settimeout(100)
6     print("Current socket timeout: %s" %s.gettimeout())
7
8 if __name__ == '__main__':
9     test_socket_timeout()
10
11
```

The bottom status bar shows the Run configuration for 'socket_timeout'. The command line is 'C:\Users\tanvi\PycharmProjects\Network_Lab\venv\Scripts\python.exe C:/Users/tanvi/PycharmProjects/Network_Lab/socket_timeout.py'. The output is:

```
Default socket timeout: None
Current socket timeout: 100.0
```

Which is the role of socket timeout in real applications?

Answer:

In complex networks and in consumer computers, there is a digital component called a socket that connects two different platforms. When there is a problem with the socket connection, such as the network being unavailable or there being no Internet, the socket will keep trying to connect. A socket timeout stops this connection after a specified amount of time. The socket timeout command is usually created in object-oriented programming (OOP) or network programming, and keeps the socket from creating inflated problems by severing the connection.

A socket timeout is a designated amount of time from when the socket connects until the connection breaks. Many users believe the timeout itself is a problem, but the timeout is actually made to keep further problems from manifesting. The amount of time between the connection and the timeout is set by programmers of the software or **operating system** (OS). Without a timeout command, the socket will continue to attempt the connection indefinitely.

If the socket timeout is not programmed, then the socket will remain open as it waits for the other side to connect. Allowing it to remain open opens the computer up to potential malicious attacks; more commonly, the computer just uses excess memory to connect to a network that is not responding. This also keeps the socket from being used for anything else, which makes the entire computer slow down.

OS and software programmers have to specify the socket timeout wait time. This is most commonly seen in OOP or network programming, because these are the programs that use sockets the most; most website programming does not use sockets as often and has no timeout commands. The timeout amount is generally measured in milliseconds, but the programmer can make the timeout take several minutes or even hours if he or she desires.

Most programmers have two socket timeout messages, one for a connection that is not responding and another for when the server or network program is closed. A socket timeout is not always needed for a socket to stop the connection. When a server or computer is about to close the connection, it sends a signal to the socket to do the same and close the connection between the two systems. This signal is not always received, including when the Internet suddenly crashes or the **Ethernet** cable is removed during the connection time. In these instances, the socket will just keep waiting for data.

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

Run the script, which is the output?

Answer:

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

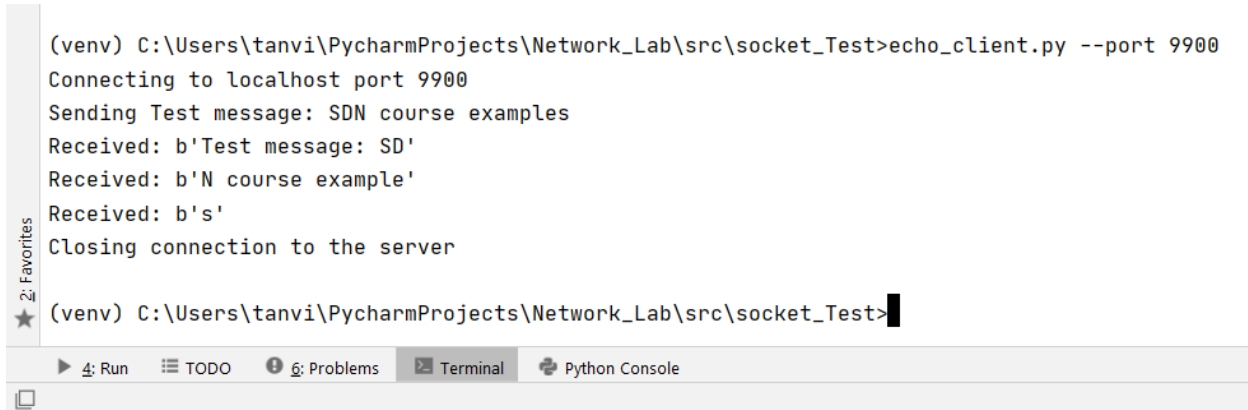
Output:

```
(venv) C:\Users\tanvi\PycharmProjects\Network_Lab\src\socket_Test>echo_server.py --port 9900
Starting up echo server on localhost port 9900
Waiting to receive message from client
Data: b'Test message: SDN course examples'
sent b'Test message: SDN course examples' bytes back to ('127.0.0.1', 20250)
Waiting to receive message from client
```

4: Run TODO 6: Problems Terminal Python Console

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

Output:



```
(venv) C:\Users\tanvi\PycharmProjects\Network_Lab\src\socket_Test>echo_client.py --port 9900
Connecting to localhost port 9900
Sending Test message: SDN course examples
Received: b'Test message: SD'
Received: b'N course example'
Received: b's'
Closing connection to the server
(venv) C:\Users\tanvi\PycharmProjects\Network_Lab\src\socket_Test>
```

What you need to do for running the program?

Answer: I did following steps :

1. First I started the server at 9900 port
`echo_server.py --port 9900`
2. Secondly I sent the request to the server from the client.
`echo_client.py --port 9900`

Which program do you need to run the first client of the server?

Answer: I ran first server program to start the server.

5. Questions

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

Answer:

Python Function : A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Python Module : A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

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

Answer:

Local variable is created when the function starts execution and lost when the functions terminate. Global variable is created before the program's global execution starts and lost when the program terminates.

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

Answer:

A socket is one endpoint of a two way communication link between two programs running on the network. The socket mechanism provides a means of inter-process communication (IPC) by establishing named contact points between which the communication takes place. Like 'Pipe' is used to create pipes and sockets is created using 'socket' system call. The socket provides bidirectional FIFO Communication facility over the network. A socket connecting to the network is created at each end of the communication. Each socket has a specific address. This address is composed of an IP address and a port number. Sockets are generally employed in client server applications. The server creates a socket, attaches it to a network port addresses then waits for the client to contact it. The client creates a socket and then attempts to connect to the server socket. When the connection is established, transfer of data takes place.

Several types of Internet socket are available:

Datagram sockets

[Connectionless](#) sockets, which use [User Datagram Protocol](#) (UDP).^[3] Each packet sent or received on a datagram socket is individually addressed and routed. Order and reliability are not guaranteed with datagram sockets, so multiple packets sent from one machine or process to another may arrive in any order or might not arrive at all. Special configuration may be required to send [broadcasts](#) on a datagram socket.^[4] In order to receive broadcast packets, a datagram socket should not be bound to a specific address, though in some implementations, broadcast packets may also be received when a datagram socket is bound to a specific address.^[5]

Stream sockets

[Connection-oriented](#) sockets, which use [Transmission Control Protocol](#) (TCP), [Stream Control Transmission Protocol](#) (SCTP) or [Datagram Congestion Control Protocol](#) (DCCP). A stream socket provides a [sequenced](#) and unique flow of error-free data without record boundaries, with well-defined mechanisms for creating and destroying connections and reporting errors. A stream socket transmits data [reliably](#), in order, and with [out-of-band](#) capabilities. On the Internet, stream sockets are typically implemented using TCP so that applications can run across any networks using TCP/IP protocol.

Raw sockets

Allow direct sending and receiving of IP packets without any protocol-specific transport layer formatting. With other types of sockets, the [payload](#) is automatically [encapsulated](#) according to the chosen transport layer protocol (e.g. TCP, UDP), and the socket user is unaware of the existence of protocol [headers](#) that are broadcast with the payload. When reading from a raw socket, the headers are usually included. When transmitting packets from a raw socket, the automatic addition of a header is optional.

Most socket [application programming interfaces](#) (APIs), for example those based on [Berkeley sockets](#), support raw sockets. [Windows XP](#) was released in 2001 with raw socket support implemented in the [Winsock](#) interface, but three years later, Microsoft limited Winsock's raw socket support because of security concerns.^[6]

Raw sockets are used in security-related applications like [Nmap](#). One use case for raw sockets is the implementation of new transport-layer protocols in [user space](#).^[7] Raw sockets are typically available in network equipment, and used for [routing protocols](#) such as the [Internet Group Management Protocol](#) (IGMP) and [Open Shortest Path First](#) (OSPF), and in the [Internet Control Message Protocol](#) (ICMP) used, among other things, by the [ping utility](#).^[8]

Other socket types are implemented over other transport protocols, such as [Systems Network Architecture](#)^[9] and [Unix domain sockets](#) for internal inter-process communication.

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

Answer:

The IPv4 address is a 32-bit number that uniquely identifies a network interface on a machine. An IPv4 address is typically written in decimal digits, formatted as four 8-bit fields that are separated by periods. Each 8-bit field represents a byte of the IPv4 address. This form of representing the bytes of an IPv4 address is often referred to as the dotted-decimal format.

When a user opens a web browser and types `www.tutorialpoints.com` which is a domain name and a PC does not understand how to communicate with the server using domain names, then the PC sends a DNS query out on the network in order to obtain the IP address pertaining to the domain name. The pre-configured DNS server responds to the query with the IP address of the domain name specified.