



Mawlana Bhashani Science And Technology University

Lab-Report

Lab Report No: 02

Lab Report Name: Programming with Python

Course Title : Network Planning and Designing Lab

Group member ID: IT-18013 and IT-18028

Course Title: Network Planning and Designing Lab

Course Code: ICT-3208

Date of Performance: 18-01-2021

Date of Submission: 18-01-2021

Submitted by

Name: Anjom Nour Anika and
Fatema tuz jannat

ID: IT-18013 and IT-18028.

3rd Year 2nd Semester

Session: 2017-2018

Dept. of ICT MBSTU

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: In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Local Variables: In computer science, a local variable is a variable that is given local scope. Local variable references in the function or block in which it is declared override the same variable name in the larger scope. In programming languages with only two levels of visibility, local variables are contrasted with global variables.

are declared in starting from the point of definition of the name.

The global statement: Global statements generally provide information to SAS, request information or data, move between different modes of execution, or set values for system options. Other global statements (ODS statements) deliver output in a variety of formats, such as in Hypertext Markup Language (HTML).
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?

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

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 = 20 #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
```

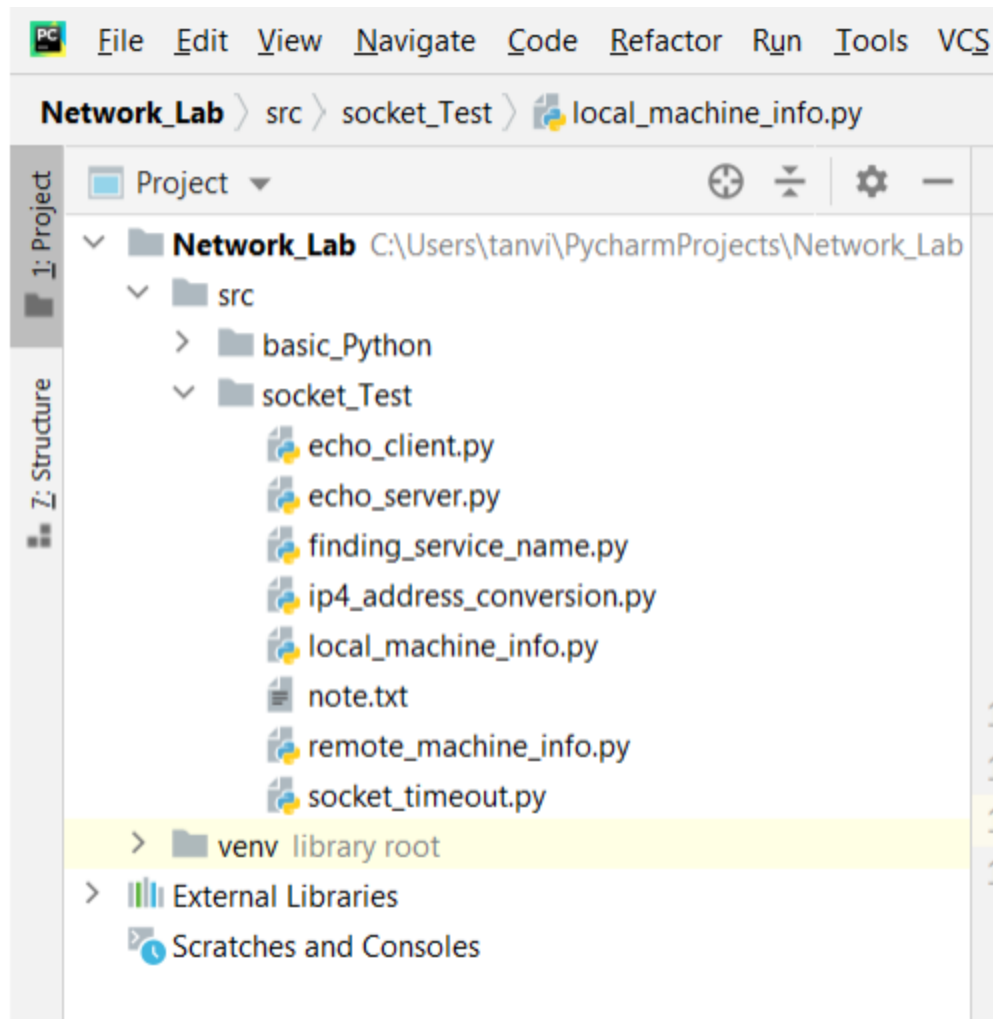
Using modules: A module can be imported by another program to make use of its functionality.

This is how we can use the Python standard library as well.

```
import xx_yy
```

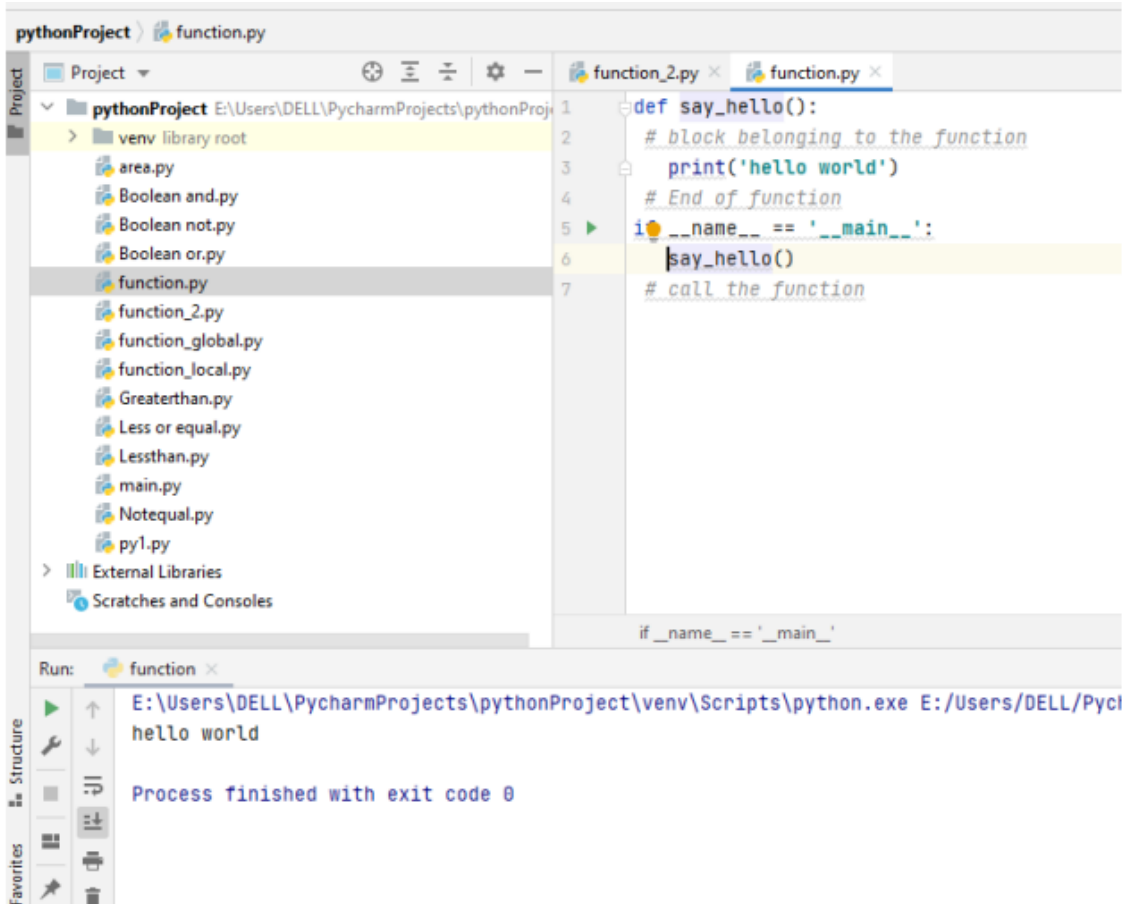
4. Exercises

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:


```
pythonProject  function_2.py
Project
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
module_demo.py
mymodule.py
Notequal.py
py1.py
> External Libraries
Scratches and Consoles

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
9 if __name__ == '__main__':
10     pass
11     print_max(3, 4)
12     # directly pass literal values
13     x = 5
14     y = 7
15     # pass variables as arguments
16     print_max(x, y)
17

Run: function_2
E:\Users\DELL\PycharmProjects\pythonProject\venv\Scripts\python.exe E:/Users/DELL/Pycharm
4 is maximum
7 is maximum

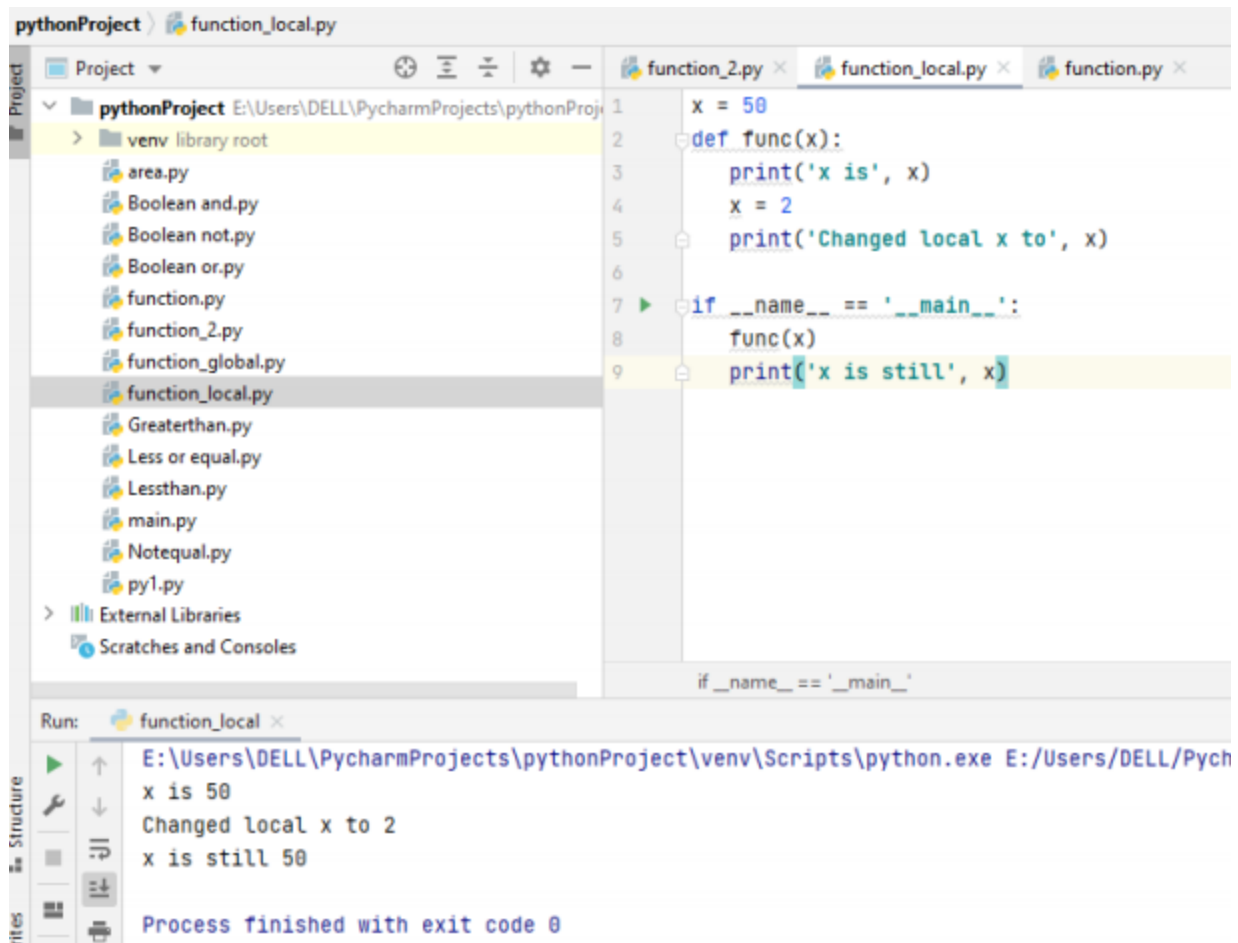
Process finished with exit code 0
```

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:

```
pythonProject > function_global.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

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)

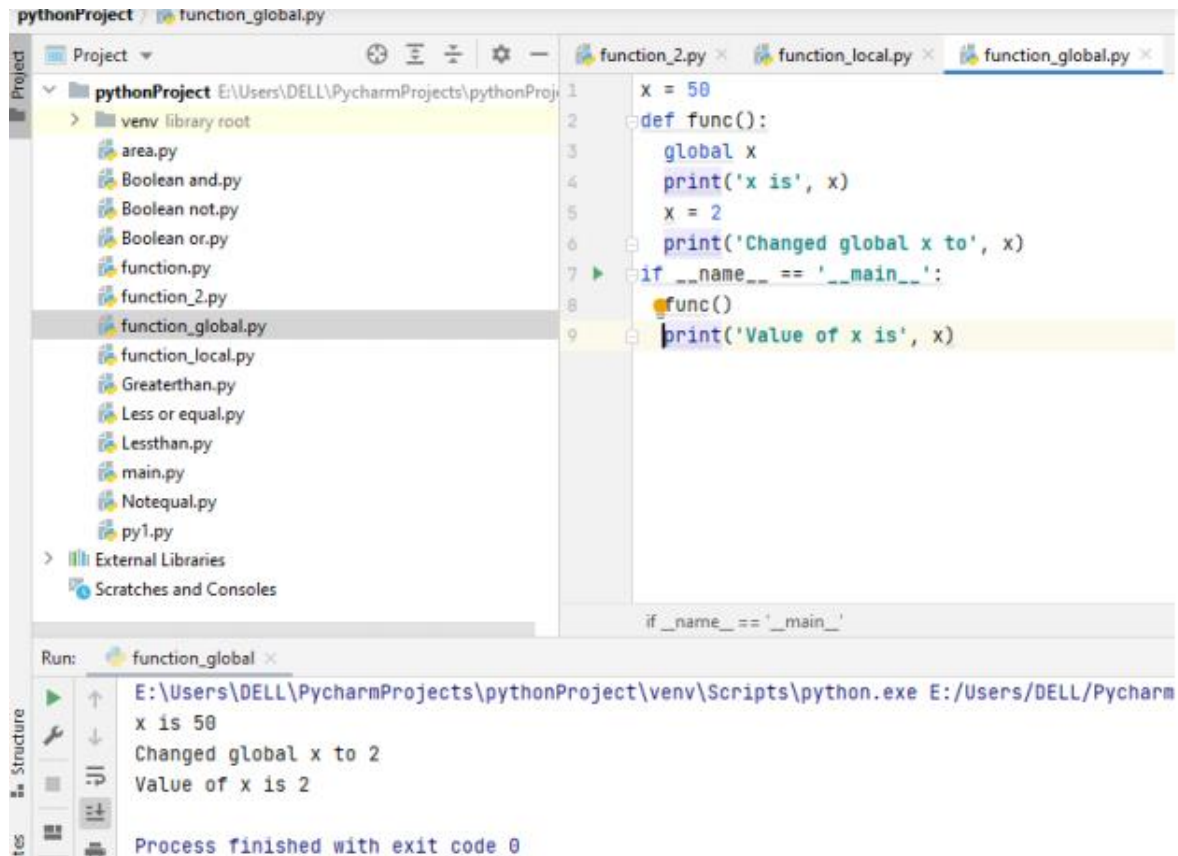
Run: function_global x
E:\Users\DELL\PycharmProjects\pythonProject\venv\Scripts\python.exe E:/Users/DELL/Pycharm
x is 50
Changed global x to 2
Value of x is 2
Process finished with exit code 0
```

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:

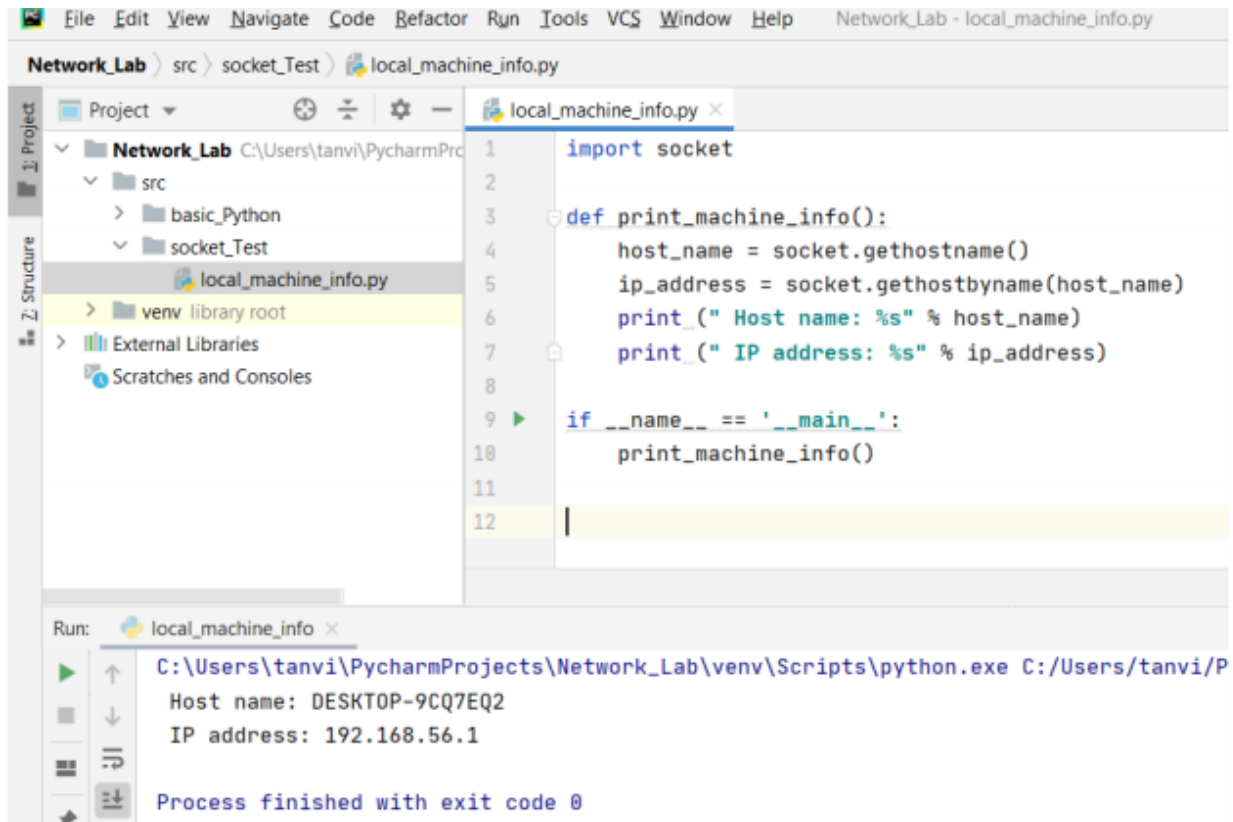


Section 4.2: Sockets, IPv4, and Simple Client/Server Programming

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 title bar indicates the project is 'Network_Lab' and the file is 'local_machine_info.py'. The left sidebar shows the Project and Structure tool windows. The Project window displays the file hierarchy: Network_Lab > src > socket_Test > local_machine_info.py. The Structure window shows the 'venv' library root and 'Scratches and Consoles'. The main editor 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
```

The bottom Run window shows the execution of the script. The command used is 'C:\Users\tanvi\PycharmProjects\Network_Lab\venv\Scripts\python.exe C:/Users/tanvi/P'. The output is:

```
Host name: DESKTOP-9CQ7EQ2
IP address: 192.168.56.1
```

The 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:

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

```
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 accessing %s: error number and detail %s" % (remote_host, err_msg))
9 if __name__ == '__main__':
10     get_remote_machine_info()
11
```

Modify the code for getting the RMIT website info.

Answer:

```
Network_Lab src socket_Test local_machine_info.py
Project C:\Users\tanvi\PycharmProjects\Network_Lab
src
basic_Python
socket_Test
echo_client.py
echo_server.py
finding_service_name.py
ip4_address_conversion.py
local_machine_info.py
note.txt
remote_machine_info.py
socket_timeout.py
venv library root
External Libraries
Scratches and Consoles
Run: local_machine_info
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
```

```
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
```

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:



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

Run

```
C:\Users\tanvi\PycharmProjects\Network_Lab\venv\Scripts\python.exe C:\Users\tanvi\PycharmProjects\Network_Lab\src\socket_Test\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'c0a00001', Unpacked: 192.168.0.1
```

How binascii works?

Answer:

I have what should be an array of Hex but always seen as a 'str' by binascii.crc32().

As an example:

```
data = ['aa', 'bb', 'cc'].
```

This is for frame building in order to put it in a txt file openable by Wireshark under specific format (not the problem here, this works fine).

As shown in documentation: `print(binascii.crc32(b"hello world"))` works.

I tried to convert data into binary with `bin()` which gave me

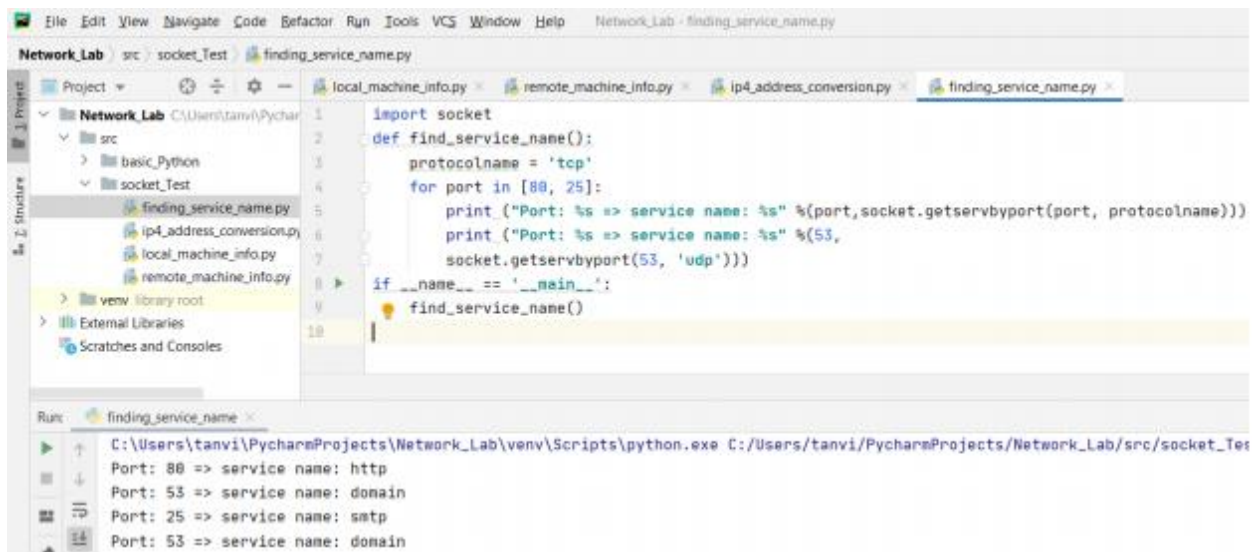
```
data = ['10101010', '10111011', '11001100']
```

However, it is never seen as binary.

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:



The screenshot shows a PyCharm IDE window titled 'Network_Lab - finding_service_name.py'. The left sidebar displays a project structure with folders 'src' and 'socket_Test'. The main editor shows the following Python code:

```
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
```

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

```
Run: finding_service_name x
C:\Users\tanvi\PycharmProjects\Network_Lab\venv\Scripts\python.exe C:/Users/tanvi/PycharmProjects/Network_Lab/src/socket_Test
Port: 80 => service name: http
Port: 53 => service name: domain
Port: 25 => service name: smtp
Port: 53 => service name: domain
```

Answer:

Exercise 4.2.5: Setting and getting the default socket timeout

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

Answer:

Port Protocol Name

21 ftp

22 ssh

110 pop3

Which is the role of socket timeout in real applications?

Answer:

A connection timeout is the maximum amount of time that the program is willing to wait to setup a connection to another process. You aren't getting or posting any application data at this point, just establishing the connection, itself.

A socket timeout is the timeout when waiting for individual packets. It's a common misconception that a socket timeout is the timeout to receive the full response. So, if you have a socket timeout of 1 second, and a response comprised of 3 IP packets, where each response packet takes 0.9 seconds to arrive, for a total response time of 2.7 seconds, then there will be no timeout.

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

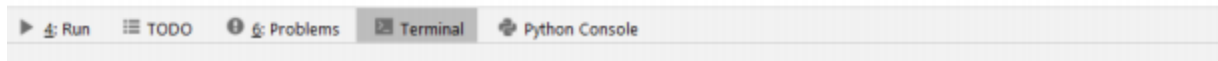
Run the script, which is the output?

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
```



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

Output:

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:

Functions:

Functions are an important step to cover, when introducing additional complexity into your programming. If a variable is a named container for data, then a function is a named container for a block of code that can be executed on demand. This is useful if you have a program that executes a certain operation lots of times. Instead of copying and pasting the code to do that operation each section where it's needed, you can simply write one single function to do it for you.

A function is a named container for a block of code.

There are two types of functions: the ones that you write yourself and include in your code, and the ones that are included in Python natively, which carry out common procedures, such as converting an integer to a string, or finding the length of a string.

Modules:

If functions are groups of code, then modules are groups of functions. There's more to them, but that's a good way of thinking about them for now.

As we discussed earlier, you can't make a complex program without sorting code into functions, and as your program continues to grow further, even they become unwieldy. You have to sort them into another hierarchical level. These are modules. So, we can summarize them as a structural and organizational tool for our code.

Modules are quite easy to create. They are simply Python files, like your regular scripts. To create a module, write one or more functions in a text file, then save it with a .py extension. Let's do that now with an example. Open up a new file your text editor or IDE, and make a function. I'm going to continue with the example of a shopping cart from earlier and make a function to calculate tax on the products.

We're going to look at writing a simple function now, and demonstrate how it can be useful in the real world code. Then, we'll have a look at some of the most important built-in functions.

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

Answer:

Local Variable:

A local variable is always declared inside a function block. In C, a local variable is declared at the start of a code block. In C++, they can be declared anywhere in the code block prior to their use.

Global Variable:

A global variable is declared outside all the functions present in a program. Unlike local variables, the global variable can be accessed by any function present in a program. Global variables are not much reliable as their value can be changed by any function present in the program

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.