

MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

Santosh, Tangail -1902



Lab Report No : 05
Lab Report Name : Programing with Python
Course Name : Computer Networks Lab
Course Code : ICT - 3208

Submitted by,

Name : Md. Imtyaz Ahmed

ID : IT-17017

Session : 2016-17

Dept. of ICT, MBSTU.

Submitted to,

Nazrul Islam

Assistant Professor

Dept. of ICT, MBSTU

Objective :

The objective of this lab is to:

- Understand how python function works
- Understand the use of global and local variables
- Understand how python modules works

Learning the basis of networking programming with python.

Theory: Python is an exceptionally basic programming language so regardless of whether you are new to programming, you can learn python without confronting any issues. Python is allowed to download and utilize. This implies you can download it for nothing and use it in your application.

Python is a broadly useful language now and then alluded to as utilitarian which is intended to be easy to peruse and compose. The point that it is anything but a mind boggling language is significant. The fashioners set less of an accentuation on regular grammar, which makes it simpler to work with, in any event, for non-software engineers or designers.

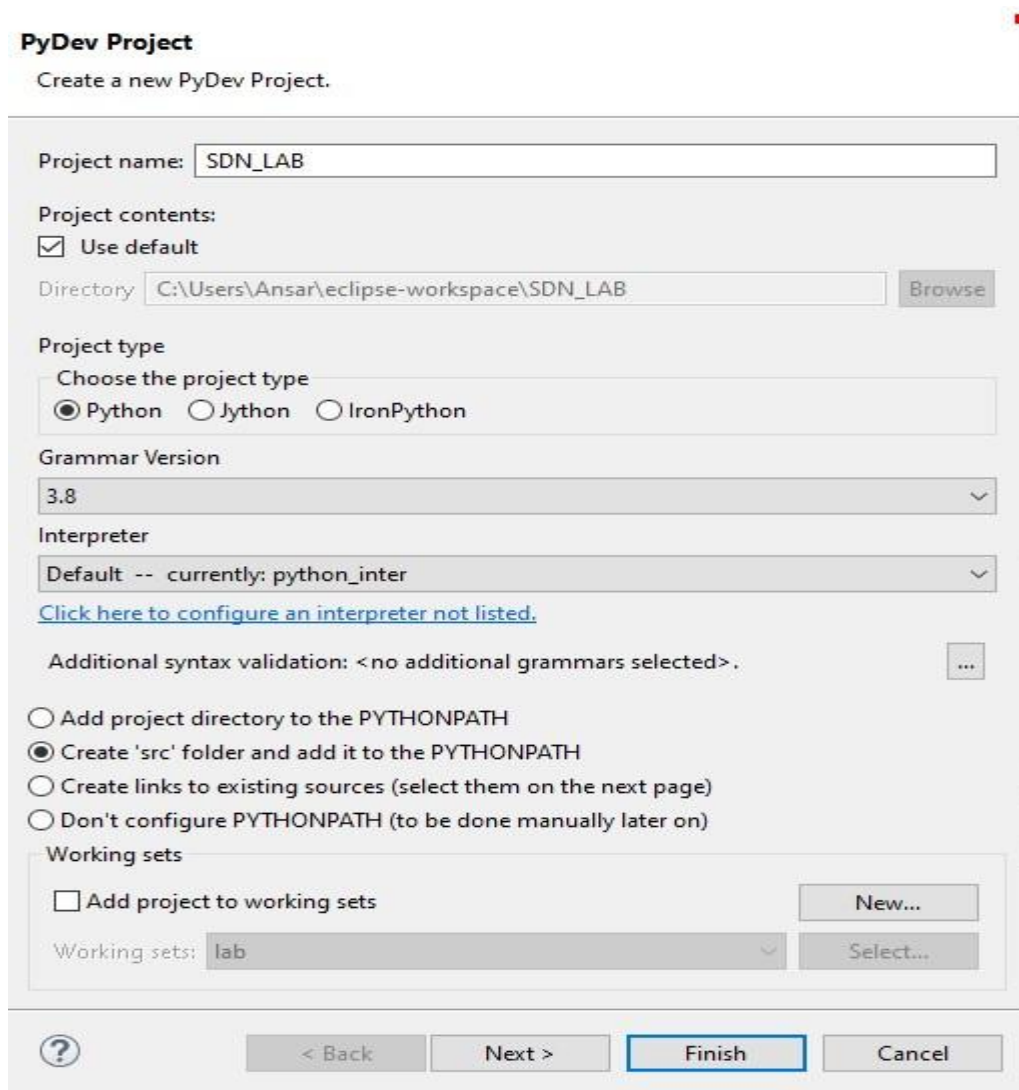
Moreover, in light of the fact that it's considered really all inclusive and used to meet different improvement needs, it's a language that offers a ton of choices to software engineers all in all. On the off chance that they start working with Python for one work or vocation, they can undoubtedly bounce to another, regardless of whether it's in a disconnected industry. The language is utilized for framework tasks, web improvement, worker and authoritative instruments, sending, logical demonstrating and considerably more.

In any case, shockingly, numerous engineers don't get Python as their essential language. Since it's so natural to utilize and learn, they pick it as a second or third language. This might be another motivation behind why it's so well known among engineers.

Additionally, for reasons unknown one of the greatest tech organizations on the planet Google utilizes the language for some of their applications. They even have a designer entry gave to Python, with free classes offered including works out, address recordings and then some.

What's more, the ascent in the utilization of the Django system for web improvement and a decrease in notoriety of PHP has additionally added to Python's prosperity, at the same time, at last, it's the ideal tempest the perfect measure of designer and authority uphold, just as request.

Exercise 4.1.1: Create a python project using with SDN_LAB Ans:



The image shows the 'PyDev Project' dialog box in Eclipse. The 'Project name' field is set to 'SDN_LAB'. Under 'Project contents', the 'Use default' checkbox is checked. The 'Directory' field shows 'C:\Users\Ansar\eclipse-workspace\SDN_LAB'. In the 'Project type' section, 'Python' is selected. The 'Grammar Version' is set to '3.8'. The 'Interpreter' is set to 'Default -- currently: python_inter'. There is a link to 'Click here to configure an interpreter not listed.' Under 'Additional syntax validation', it says '<no additional grammars selected>'. In the 'Working sets' section, 'Add project to working sets' is unchecked, and the 'Working sets' dropdown is set to 'lab'. At the bottom, there are buttons for '< Back', 'Next >', 'Finish' (which is highlighted), and 'Cancel'.

PyDev Project
Create a new PyDev Project.

Project name: SDN_LAB

Project contents:
☒ Use default

Directory: C:\Users\Ansar\eclipse-workspace\SDN_LAB Browse

Project type
Choose the project type
☒ Python ☐ Jython ☐ IronPython

Grammar Version
3.8

Interpreter
Default -- currently: python_inter

[Click here to configure an interpreter not listed.](#)

Additional syntax validation: <no additional grammars selected>.

☐ Add project directory to the PYTHONPATH
☒ Create 'src' folder and add it to the PYTHONPATH
☐ Create links to existing sources (select them on the next page)
☐ Don't configure PYTHONPATH (to be done manually later on)

Working sets
☐ Add project to working sets New...

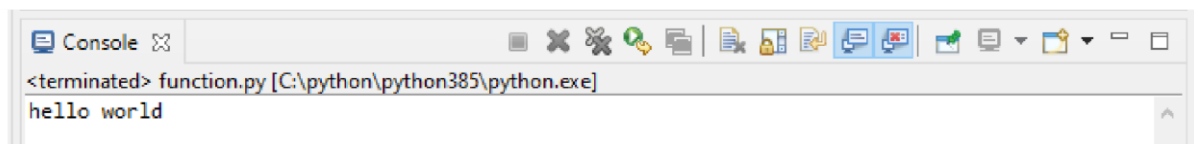
Working sets: lab Select...

? < Back Next > Finish Cancel

Exercise 4.1.2: Python function (save as function.py) Create python scrip using the syntax provided below. `def say_hello(): print('hello world')`

`if __name__ == '__main__': say_hello()`

The output of this code is:



The image shows the Eclipse Console window. The title bar says 'Console'. The output text is: `<terminated> function.py [C:\python\python385\python.exe]` followed by `hello world` on the next line.

```
<terminated> function.py [C:\python\python385\python.exe]
hello world
```

Exercise 4.1.3: Python function (save as function_2.py) Create python scrip using the syntax provided below. `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__':`

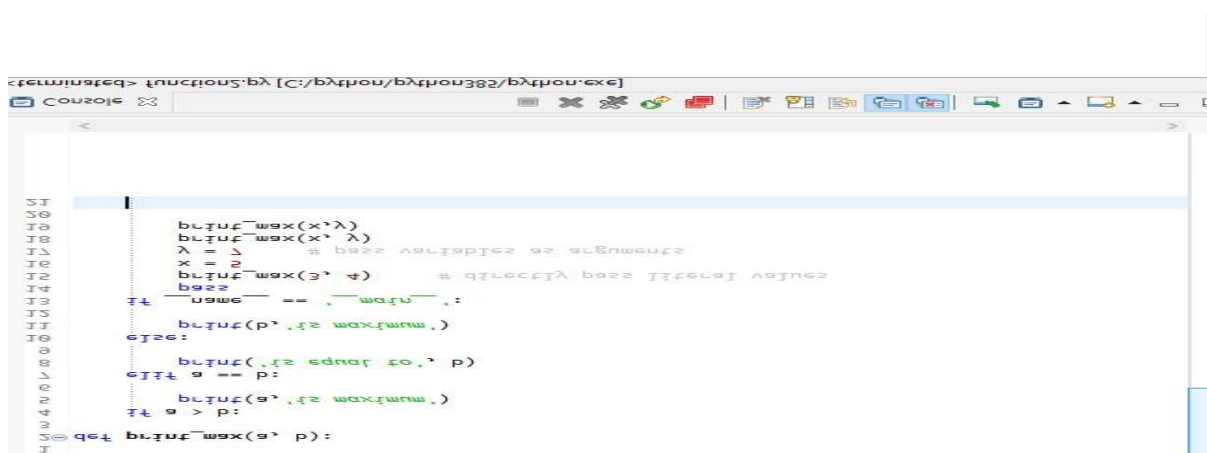
`pass print_max(3, 4) # directly pass literal values`

`x = 5`

`y = 7 # pass variables as arguments print_max(x, y)`

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

Ans: The code does not show any output. May be there is some problem in this code.



```
<terminated> function_2.py [C:/bpython/bpython382/bpython.exe]
Console
def print_max(a, b):
    """Prints the maximum of two numbers"""
    if a > b:
        print(a, 'is maximum')
    elif a == b:
        print(a, 'is equal to', b)
    else:
        print(b, 'is maximum')
if __name__ == '__main__':
    pass
    print_max(3, 4) # directly pass literal values
    x = 5
    y = 7 # pass variables as arguments
    print_max(x, y)
```

This function does not need any parameter .

Exercise 4.1.4: Local variable (save as function_local.py) Create python scrip using the syntax provided below.

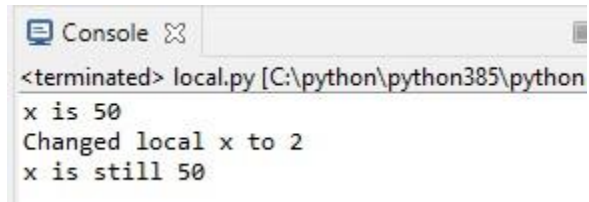
```
x = 50 def func(x):    print('x is', x)
x = 2    print('Changed local x to', x)
if __name__ == '__main__':

    func(x)

    print('x is still', x)
```

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

Ans: Output is:



```
<terminated> local.py [C:\python\python385\python
x is 50
Changed local x to 2
x is still 50
```

The final value of variable x is 50. It does not change because it is a global variable.

Exercise 4.1.5: Global variable (save as function_global.py) Create python scrip using the syntax provided below.

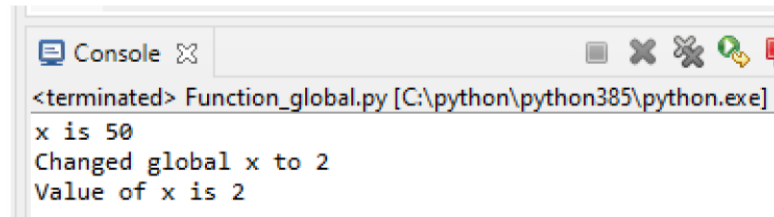
```
x = 50 def func():
global x    print('x
is', x)    x = 2

print('Changed global x to', x) if
__name__ == '__main__':

func()    print('Value of x
is', x)
```

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

Ans: Output is:



```
<terminated> Function_global.py [C:\python\python385\python.exe]
x is 50
Changed global x to 2
Value of x is 2
```

This time variable x is declared as global inside the function .So the variable x is changed.

Exercise 4.1.6: Python modules Create python scrip using the syntax provided below (save as mymodule.py).

```
def say_hi():
```

```
    print('Hi, this is mymodule speaking.')
```

```
__version__ = '0.1'
```

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

```
import mymodule if
```

```
__name__ == '__main__':
```

```
    mymodule.say_hi()    print('Version',
```

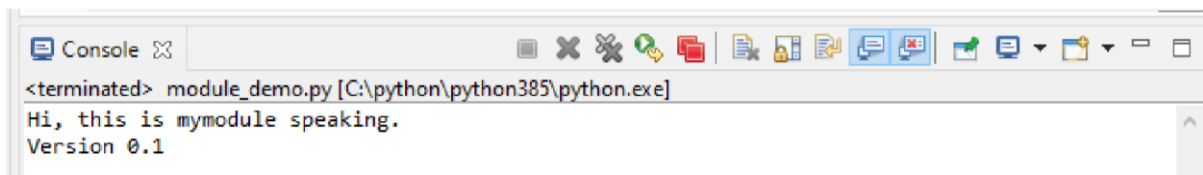
```
    mymodule.__version__)
```

Run the script, which is the role of

import?

Ans:

Output is:



```
<terminated> module_demo.py [C:\python\python385\python.exe]
Hi, this is mymodule speaking.
Version 0.1
```

Python modules can get access to code from another module by importing the file/function using import. The import statement is the most common way of invoking the import machinery, but it is not the only way. When import is used, it searches for the module initially in the local scope by calling `__import__()` function

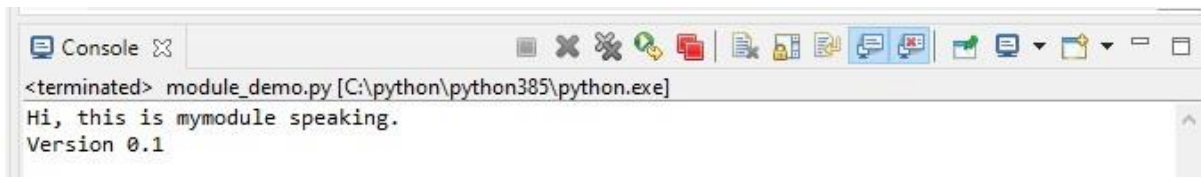
Create python scrip using the syntax provided below (save as module_demo2.py). from mymodule import say_hi, __version__

```
if __name__ == '__main__':    say_hi()
```

```
print('Version', __version__)
```

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

Ans: Output is :

A screenshot of a Python console window. The title bar says "Console". The text inside the console shows the execution of a script: "<terminated> module_demo.py [C:\python\python385\python.exe]" followed by the output "Hi, this is mymodule speaking." and "Version 0.1".

```
<terminated> module_demo.py [C:\python\python385\python.exe]
Hi, this is mymodule speaking.
Version 0.1
```

Using **‘from’** we say the module name and then using **‘import’** we say what we are importing from the module.

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): `import socket`

```
def print_machine_info():
```

```
    host_name = socket.gethostname()    ip_address =
```

```
    socket.gethostbyname(host_name) print (" Host
```

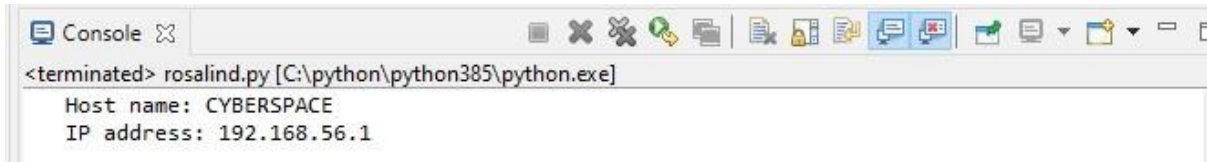
```
name: %s" % host_name)    print (" IP address: %s"
```

```
% ip_address) if __name__ == '__main__':
```

```
    print_machine_info()
```

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

Ans:



```
<terminated> rosaling.py [C:\python\python385\python.exe]
Host name: CYBERSPACE
IP address: 192.168.56.1
```

The *type* argument specifies the socket type, which determines the semantics of communication over the socket. The following socket types are defined; implementations may specify additional socket types:

SOCK_STREAM

Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.

SOCK_DGRAM

Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.

SOCK_SEQPACKET

Provides sequenced, reliable, bidirectional, connection-mode transmission paths for records. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.

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`):

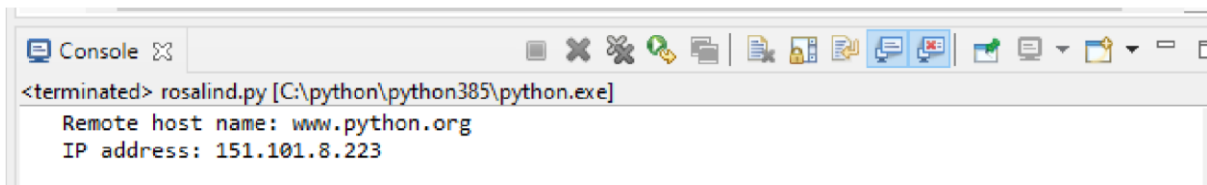
```
import socket
def
get_remote_machine_info():
remote_host = 'www.python.org'

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()
```


Run the script, which is the output? Modify the code for getting the RMIT website info.

Ans:



```
<terminated> rosaland.py [C:\python\python385\python.exe]
Remote host name: www.python.org
IP address: 151.101.8.223
```

RMIT website info:

Code:

```
import socket
def
get_remote_machi
ne_info():
remote_host =
'www.rmit.org'
try:
    print
    (" Remote host
    name: %s" %
    remote_host)
    print (" IP address:
    %s"
    %socket.gethostbyn
    ame(remote_host))
except socket.error
as err_msg:
    print ("Error
    accsesing %s: error
```

number and detail

%s"

%(remote_host,

err_msg)) if

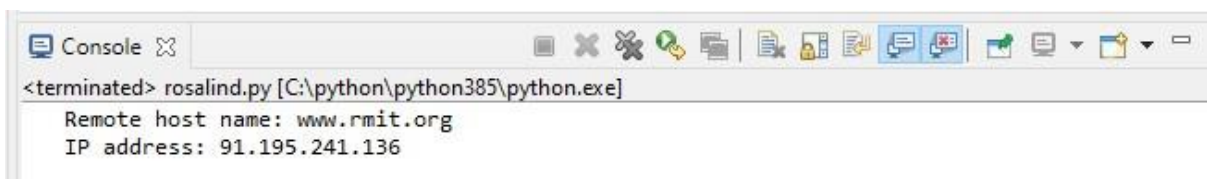
__name__ ==

'__main__':

get_remote_machi

ne_info()

Output:



```
<terminated> rosaland.py [C:\python\python385\python.exe]
Remote host name: www.rmit.org
IP address: 91.195.241.136
```

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

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)

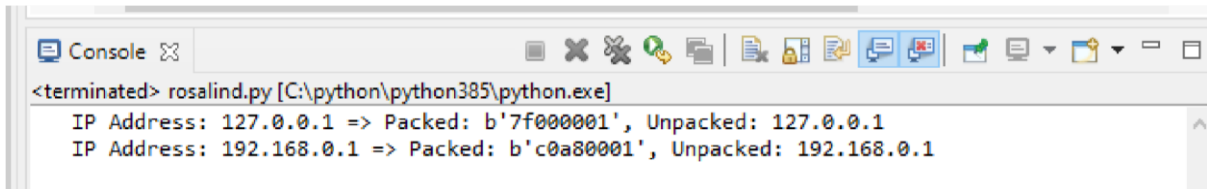
print (" IP Address: %s => Packed: %s, Unpacked: %s" %(ip_addr, hexlify(packed_ip_addr),

unpacked_ip_addr)) if __name__ == '__main__':

convert_ip4_address()

Run the script, which is the output? How binascii works?

Ans:



```
<terminated> rosaland.py [C:\python\python385\python.exe]
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
```

Binascii:

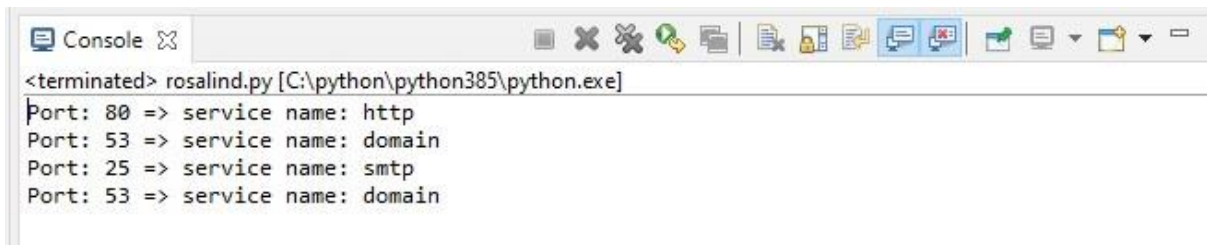
The **binascii** module contains a number of methods to convert between binary and various ASCIIencoded binary representations. ... Convert binary data to a line of ASCII characters, the return value is the converted line, including a newline char. The length of data should be at most 45.

Exercise 4.2.4: 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()
```

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

Output:



```
<terminated> rosaland.py [C:\python\python385\python.exe]
Port: 80 => service name: http
Port: 53 => service name: domain
Port: 25 => service name: smtp
Port: 53 => service name: domain
```

For the given port the code will be:

```
import socket def find_service_name():
    protocolname = 'tcp' for port in [21,22,110]:
        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:
```

```
Console [C:\python\python385\python.exe]
<terminated>
Port: 21 => service name: ftp
Port: 53 => service name: domain
Port: 22 => service name: ssh
Port: 53 => service name: domain
Port: 110 => service name: pop3
Port: 53 => service name: domain
```

Exercise 4.2.5: 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()
```

Run the script, which is the role of socket timeout in real applications?

Output:

```
Console [C:\python\python385\python.exe]
<terminated>
Default socket timeout: None
Current socket timeout: 100.0
```

A **socket timeout** implementation should allow for setting the timeout at ... For **example**, this is how we connect to a local HTTP server on port 80 ... It can be implemented as a method that we add to `IO::Socket::INET` class, possibly by using a **Role**. ...

The real version handles `EINTR` and other corner cases. **Exercise 4.2.6: Writing a simple echo client/server application (Tip: Use port 9900)**

Ans:

Server:

```
import socket

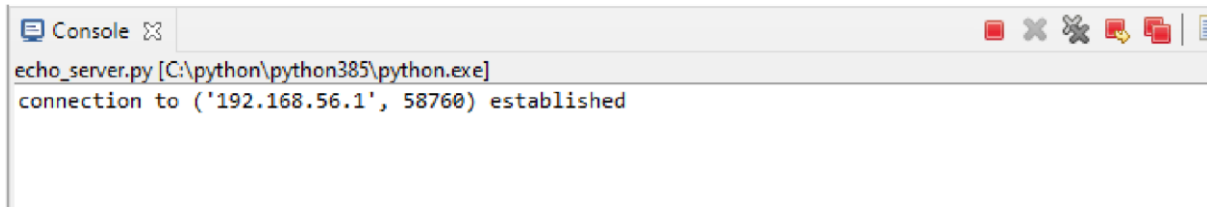
s= socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((socket.gethostname(),1022))
s.listen(5) while True:  clt,adr =
s.accept()
    print(f"connection to {adr} established")
    clt.send(bytes("Socket programming in python", "utf-8"))
```

Client:

```
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

```
s.connect((socket.gethostname(),1022)) msg = s.recv(1022)
print(msg.decode("utf-8"))
```

Output:



```
echo_server.py [C:\python\python385\python.exe]
connection to ('192.168.56.1', 58760) established
```

Conclusion : In this lab we have taken in the fundamental idea of python programming and attachment programming in python. In Python we have figured out how to characterize a capacity, neighborhood and worldwide factors. Utilizing the Python program, we figured out how to recuperate our machine's hostname, IP address, and how to recoup the IP address of a far off machine. We realize how to interface clientside and worker side utilizing the Python program.

The End