

Lab-Report

Report No: 02

Course code: ICT-3207

Course title: Computer network lab

Date of Performance:

Date of Submission:

Submitted by

Name: Ashikur Rahman Miran &
Rafiul Hasan

ID: IT-18014 & IT-18016

3rd year 2nd semester

Session: 2017-2018

Dept. of ICT

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Lab Report No. : 02

Lab Report Name: Programming with python.

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.

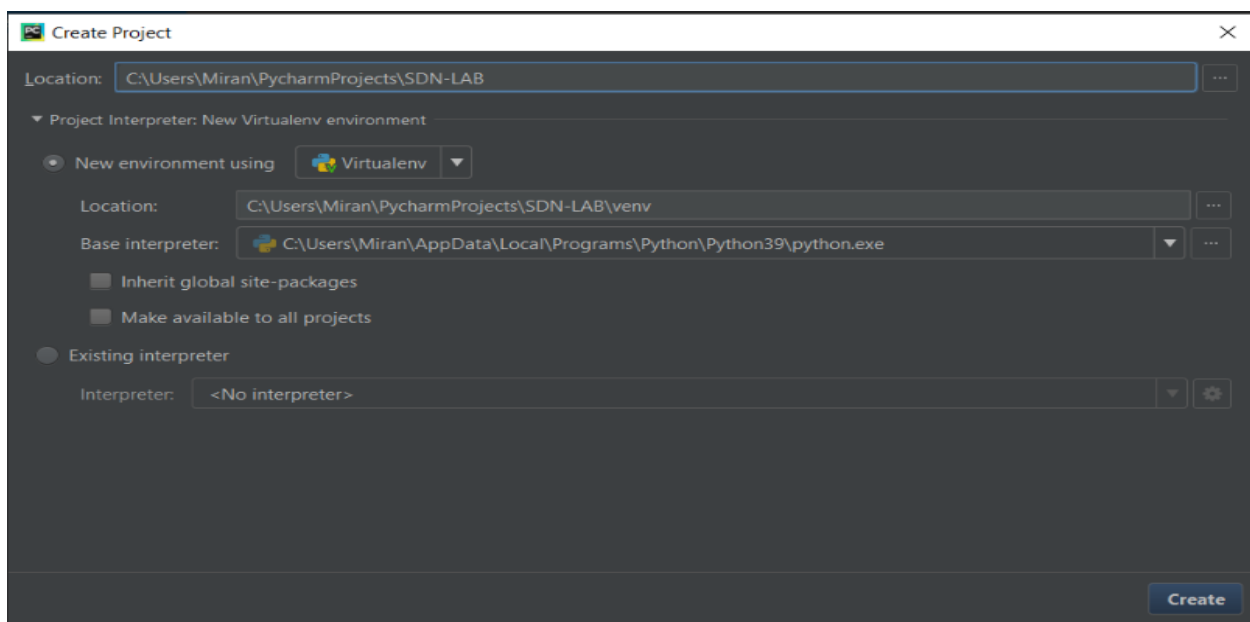
Local Variables: Variables declared inside a function definition are not related in any way to other variables with the same names used outside the function (variable names are local to the function). This is called the scope of the variable. All variables have the scope of the block they are declared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

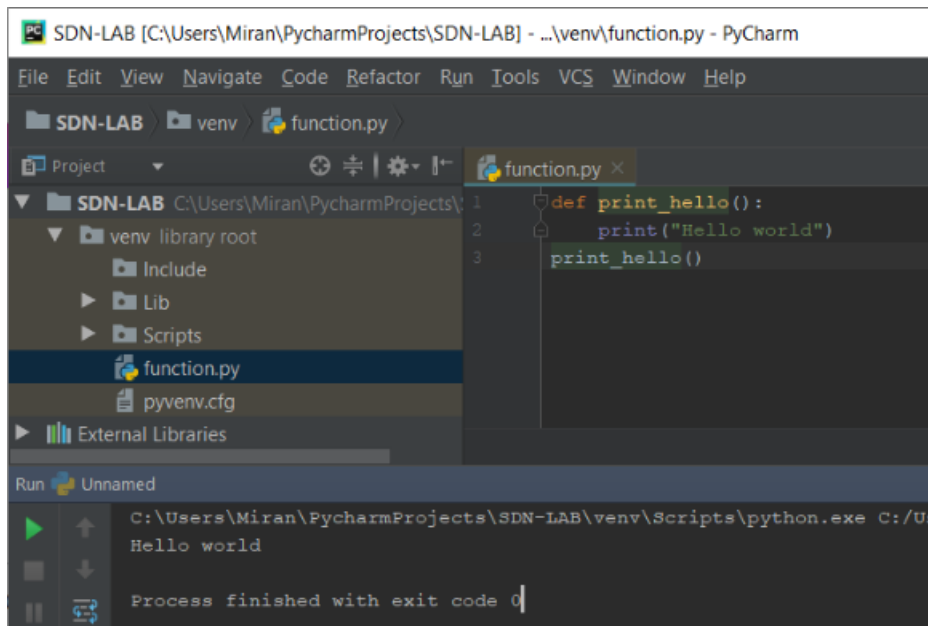
Modules: Modules allow reusing a number of functions in other programs.

Exercises:

Exercise 4.1.1: Create a Python project with SDN-LAB.



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

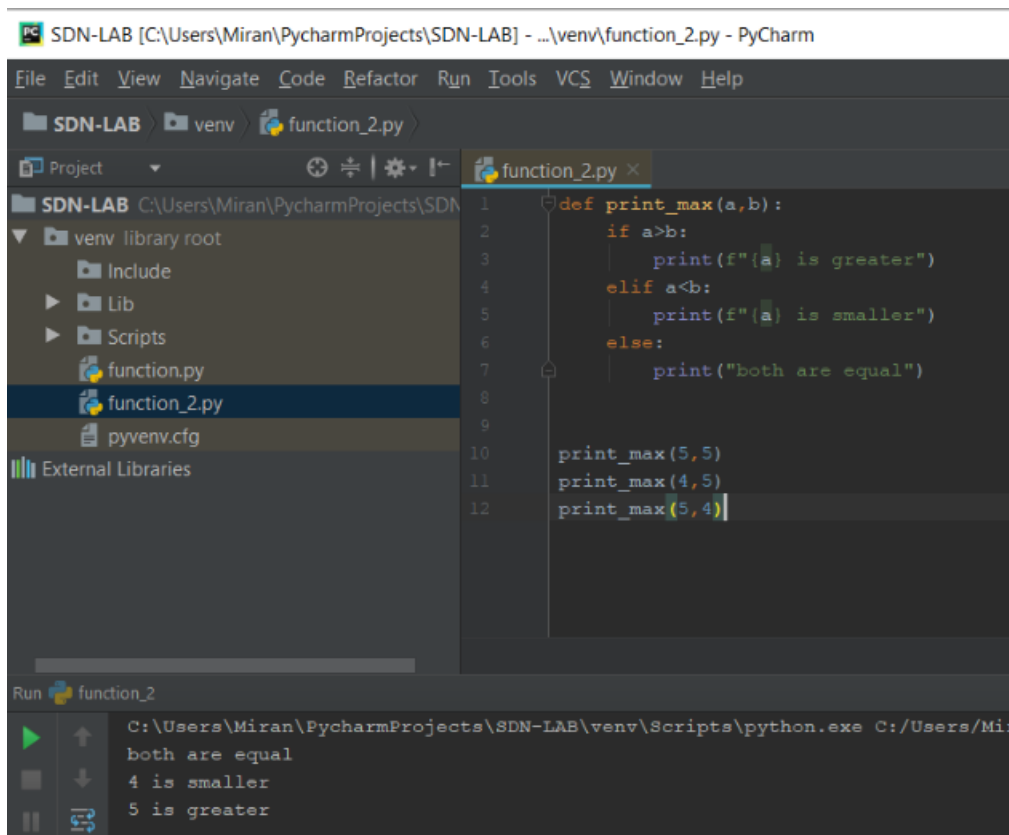


The screenshot shows the PyCharm IDE with a project named 'SDN-LAB'. The file explorer on the left shows the project structure, including a 'venv' directory with 'library root', 'Include', 'Lib', 'Scripts', 'function.py', and 'pyenv.cfg'. The main editor window displays the code for 'function.py':

```
1 def print_hello():  
2     print("Hello world")  
3     print_hello()
```

The Run window at the bottom shows the execution of the script using 'C:\Users\Miran\PycharmProjects\SDN-LAB\venv\Scripts\python.exe'. The output is 'Hello world' and the process finished with exit code 0.

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

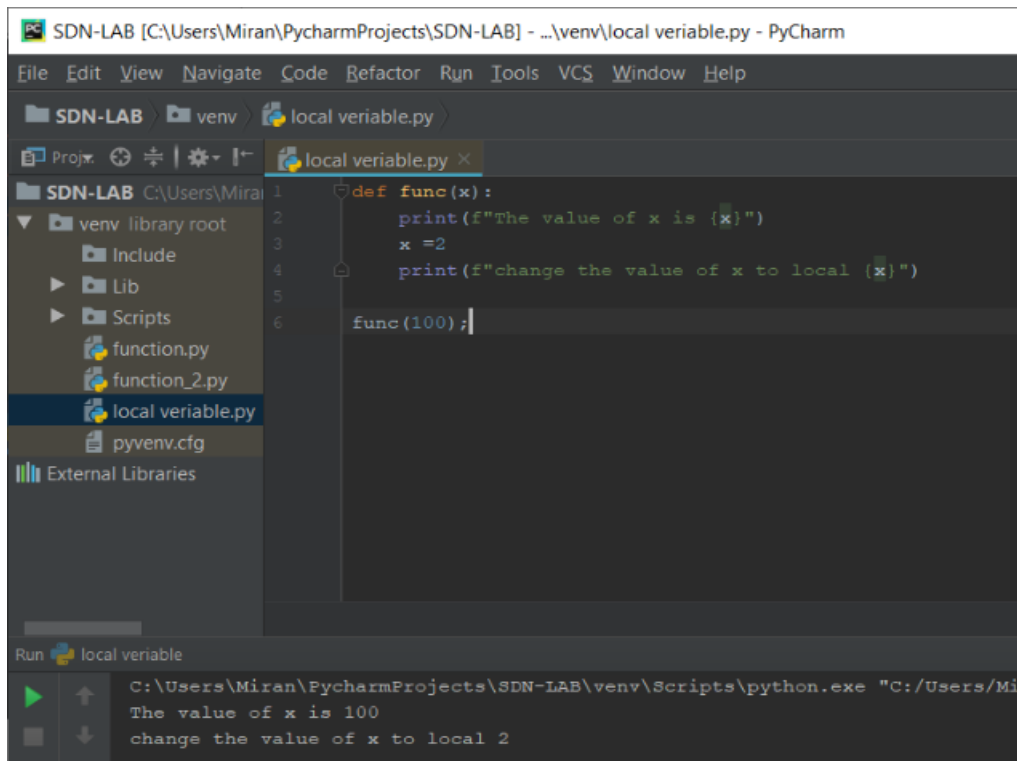


The screenshot shows the PyCharm IDE with a project named 'SDN-LAB'. The file explorer on the left shows the project structure, including a 'venv' directory with 'library root', 'Include', 'Lib', 'Scripts', 'function.py', 'function_2.py', and 'pyenv.cfg'. The main editor window displays the code for 'function_2.py':

```
1 def print_max(a,b):  
2     if a>b:  
3         print(f"{a} is greater")  
4     elif a<b:  
5         print(f"{a} is smaller")  
6     else:  
7         print("both are equal")  
8  
9  
10    print_max(5,5)  
11    print_max(4,5)  
12    print_max(5,4)
```

The Run window at the bottom shows the execution of the script using 'C:\Users\Miran\PycharmProjects\SDN-LAB\venv\Scripts\python.exe'. The output is 'both are equal', '4 is smaller', and '5 is greater'.

Exercise 4.1.4: Local variable



```
def func(x):
    print(f"The value of x is {x}")
    x = 2
    print(f"change the value of x to local {x}")

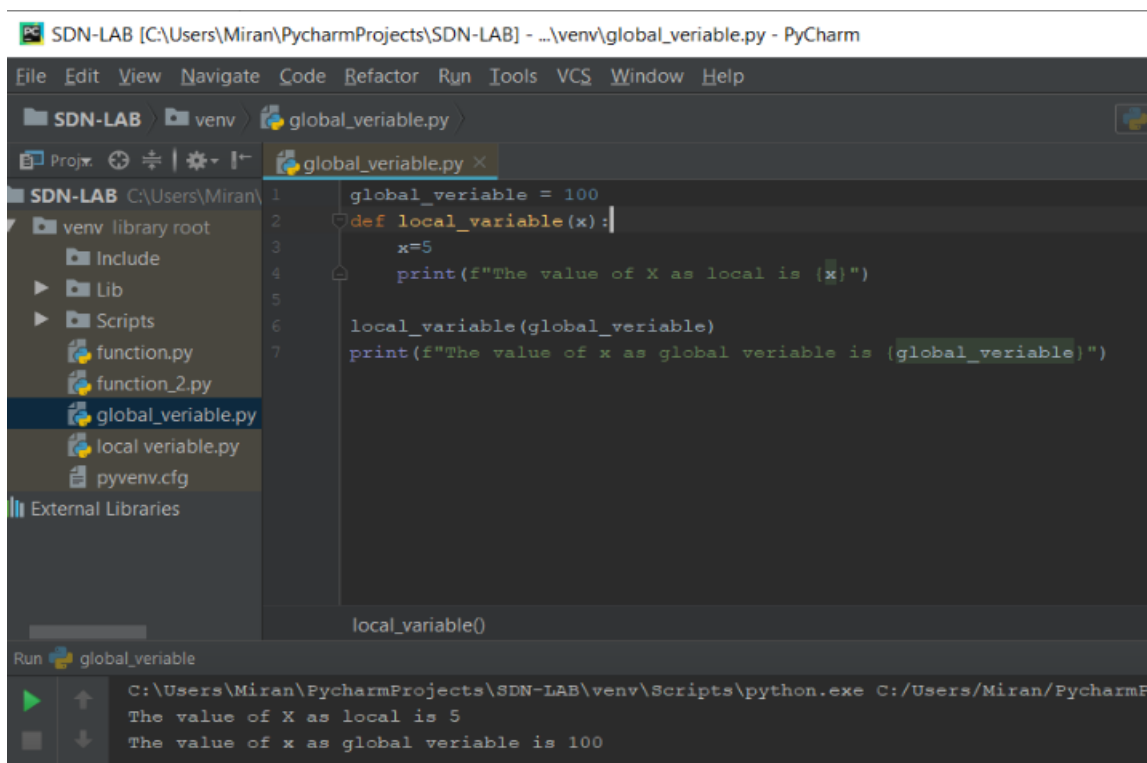
func(100);
```

Run local variable

C:\Users\Miran\PycharmProjects\SDN-LAB\venv\Scripts\python.exe "C:/Users/Miran/PycharmProjects/SDN-LAB/venv/local_variable.py"

The value of x is 100
change the value of x to local 2

Exercise 4.1.5: Global variable



```
global_variable = 100

def local_variable(x):
    x = 5
    print(f"The value of X as local is {x}")

local_variable(global_variable)
print(f"The value of x as global variable is {global_variable}")

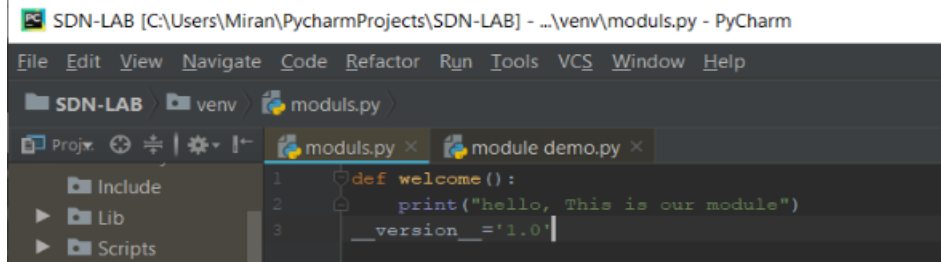
local_variable()
```

Run global_variable

C:\Users\Miran\PycharmProjects\SDN-LAB\venv\Scripts\python.exe C:/Users/Miran/PycharmProjects/SDN-LAB/venv/global_variable.py

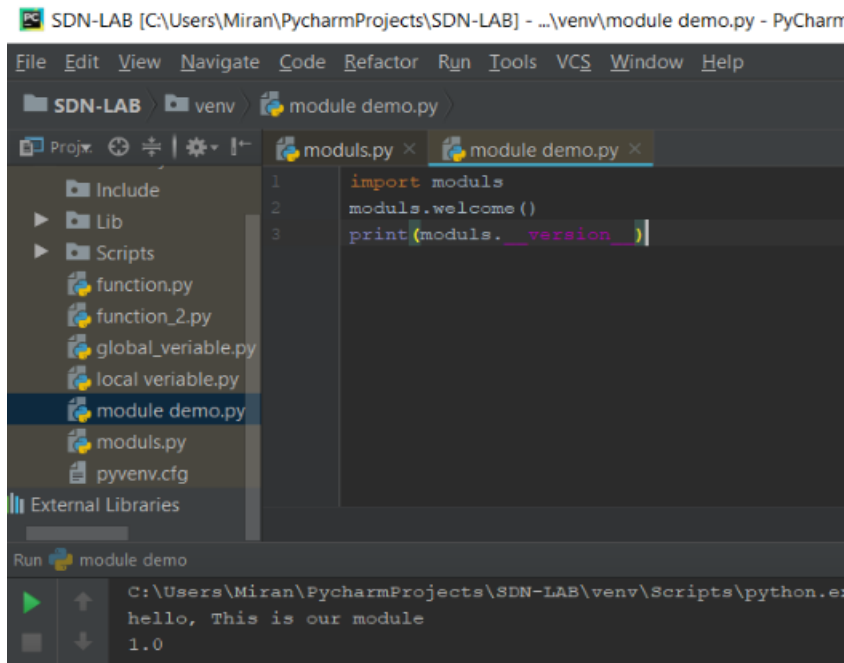
The value of X as local is 5
The value of x as global variable is 100

Exercise 4.1.6: python modules



The screenshot shows the PyCharm IDE with the 'SDN-LAB' project open. The file explorer on the left shows the 'venv' directory. The main editor window displays the 'modules.py' file with the following code:

```
1 def welcome():
2     print("hello, This is our module")
3     __version__ = '1.0'
```



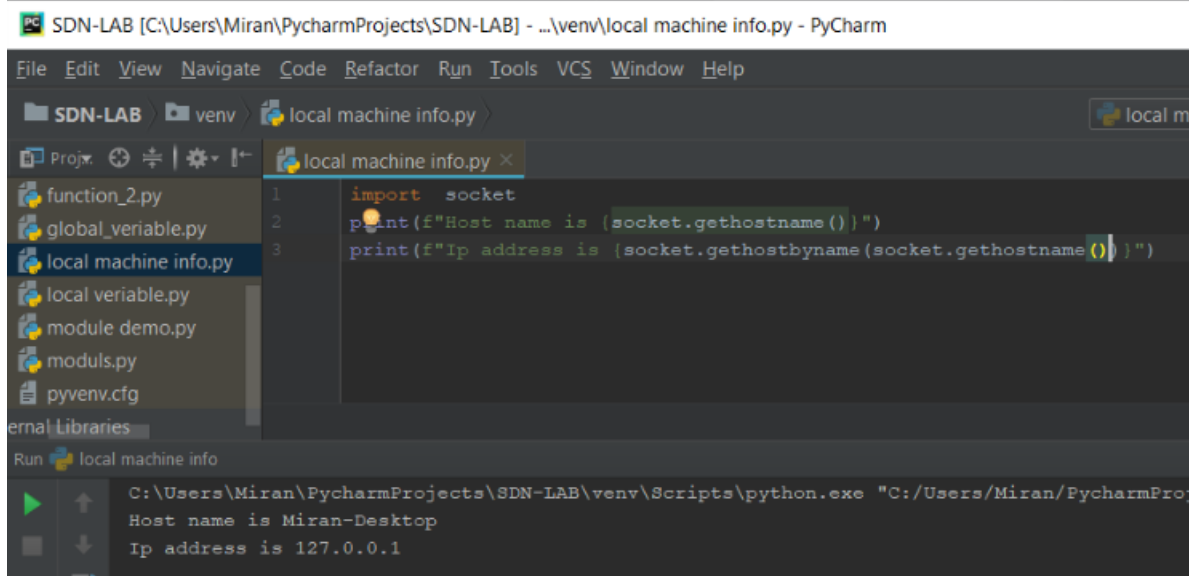
The screenshot shows the PyCharm IDE with the 'SDN-LAB' project open. The file explorer on the left shows the 'venv' directory. The main editor window displays the 'module demo.py' file with the following code:

```
1 import modules
2 modules.welcome()
3 print(modules.__version__)
```

The Run console at the bottom shows the output of the 'module demo.py' script:

```
hello, This is our module
1.0
```

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



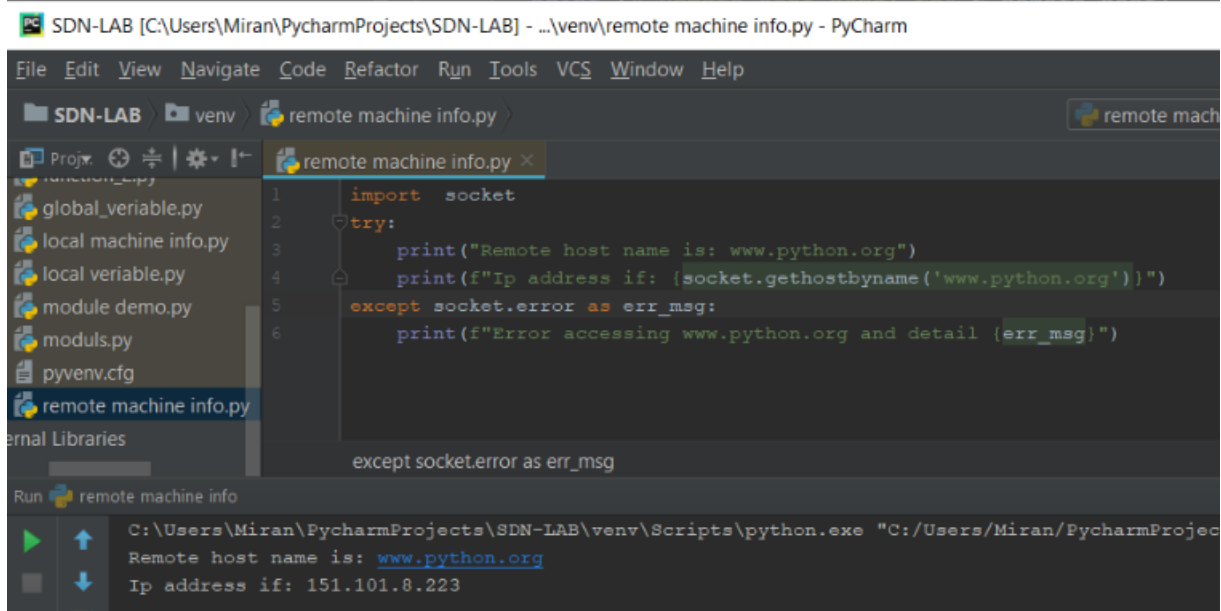
The screenshot shows the PyCharm IDE with the 'SDN-LAB' project open. The file explorer on the left shows the 'venv' directory. The main editor window displays the 'local machine info.py' file with the following code:

```
1 import socket
2 print(f"Host name is {socket.gethostname()}")
3 print(f"Ip address is {socket.gethostbyname(socket.gethostname())}")
```

The Run console at the bottom shows the output of the 'local machine info.py' script:

```
Host name is Miran-Desktop
Ip address is 127.0.0.1
```

Exercise 4.2.2: Retrieving a remote machine's IP address



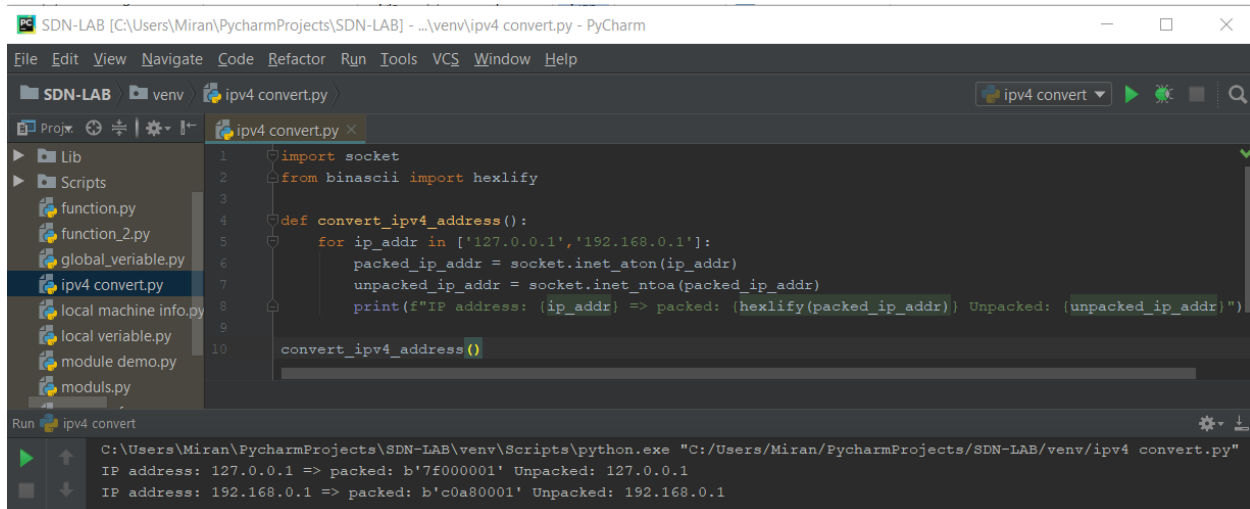
The screenshot shows the PyCharm IDE with a project named 'SDN-LAB'. The file 'remote machine info.py' is open in the editor. The code in the file is as follows:

```
1 import socket
2 try:
3     print("Remote host name is: www.python.org")
4     print(f"Ip address if: {socket.gethostbyname('www.python.org')}")
5 except socket.error as err_msg:
6     print(f"Error accessing www.python.org and detail {err_msg}")
```

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

```
Remote host name is: www.python.org
Ip address if: 151.101.8.223
```

Exercise 4.2.3: Converting an IPv4 address to different formats.



The screenshot shows the PyCharm IDE with a project named 'SDN-LAB'. The file 'ipv4 convert.py' is open in the editor. The code in the file is as follows:

```
1 import socket
2 from binascii import hexlify
3
4 def convert_ipv4_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(f"IP address: {ip_addr} => packed: {hexlify(packed_ip_addr)} Unpacked: {unpacked_ip_addr}")
9
10    convert_ipv4_address()
```

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

```
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
```

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

The screenshot shows a PyCharm IDE window titled "SDN-LAB [C:\Users\Miran\PycharmProjects\SDN-LAB] - ...\venv\finding service name.py - PyCharm". The code in "finding service name.py" is as follows:

```
1 import socket
2 def find_service_name():
3     for port in [80, 25]:
4         print(f"Port: {port} => service name: {socket.getservbyport(port, 'tcp')}")
5         print(f"Port: {53} => service name: {socket.getservbyport(53, 'udp')}")
6
7 find_service_name()
```

The Run console shows the following output:

```
C:\Users\Miran\PycharmProjects\SDN-LAB\venv\Scripts\python.exe "C:/Users/Miran/PycharmProjects/SDN-LAB/venv/finding service name.py"
Port: 80 => service name: http
Port: 53 => service name: domain
Port: 25 => service name: smtp
Port: 53 => service name: domain
```

Exercise 4.2.5: Setting and getting the default socket timeout.

The screenshot shows a PyCharm IDE window titled "SDN-LAB [C:\Users\Miran\PycharmProjects\SDN-LAB] - ...\venv\socket timeout.py - PyCharm". The code in "socket timeout.py" is as follows:

```
1 import socket
2 def test_socket_timeout():
3     s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4     print(f"Default socket timeout: {s.gettimeout()}")
5     s.settimeout(1000)
6     print(f"Current socket timeout: {s.gettimeout()}")
7
8 test_socket_timeout()
```

The Run console shows the following output:

```
C:\Users\Miran\PycharmProjects\SDN-LAB\venv\Scripts\python.exe "C:/Users/Miran/PycharmProjects/SDN-LAB/venv/socket timeout.py"
Default socket timeout: None
Current socket timeout: 1000.0
```

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

Server code:

```
SDN-LAB [C:\Users\Miran\PycharmProjects\SDN-LAB] - ...\.venv\echo server.py - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help

SDN-LAB C:\Users\Miran\Py
venv library root
Include
Lib
Scripts
echo server.py
finding service name.
function.py
function_2.py
global_variable.py
ipv4 convert.py
local machine info.py
local variable.py
module demo.py
moduls.py
pyvenv.cfg
remote machine info.
socket timeout.py
External Libraries

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     sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
11     sock.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
12     server_address = (host,port)
13     print(f"Starting up echo server on {server_address} port {sock.bind(server_address)}")
14     sock.listen(backlog)
15     while True:
16         print("Willing to receive message from client")
17         client,address = sock.accept()
18         data = client.recv(data_payload)
19         if data:
20             print(f>Data: {data}")
21             client.send(data)
22             print(f"sent {data} bytes back to {address}")
23             client.close();
24
25 parser = argparse.ArgumentParser(description='Socket Server Example')
26 parser.add_argument('--port',action='store',dest="port",type=int,required=True)
27 given_args = parser.parse_args()
28 port = given_args.port
29 echo_server(port)
```

Client code:

```
SDN-LAB [C:\Users\Miran\PycharmProjects\SDN-LAB] - ...\.venv\echo server.py - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help

SDN-LAB C:\Users\Miran\Py
venv library root
Include
Lib
Scripts
echo client.py
echo server.py
finding service name.
function.py
function_2.py
global_variable.py
ipv4 convert.py
local machine info.py
local variable.py
module demo.py
moduls.py
pyvenv.cfg
remote machine info.
socket timeout.py
External Libraries

1 import socket
2 import sys
3 import argparse
4 import codecs
5 from codecs import encode, decode
6 host='localhost'
7 data_payload=2048
8 backlog =5
9 def echo_server(port):
10     sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
11     sock.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
12     server_address = (host,port)
13     print(f"Starting up echo server on {server_address} port {server_address}")
14     sock.bind(server_address)
15     sock.listen(backlog)
16     while True:
17         print("Willing to receive message from client")
18         client,address = sock.accept()
19         data = client.recv(data_payload)
20         if data:
21             print(f>Data: {data}")
22             client.send(data)
23             print(f"sent {data} bytes back to {address}")
24             client.close();
25
26 parser = argparse.ArgumentParser(description="Computer Network lab")
27 parser.add_argument('--port',action="store",dest="port",type=int,required=True)
28 given_args = parser.parse_args()
29 port = given_args.port
30 echo_server(port)
```


Conclusion:

Python plays an essential role in network programming. The standard library of Python has full support for network protocols, encoding, and decoding of data and other networking concepts, and it is simpler to write network programs in Python than that of C++. There are two levels of network service access in Python. These are:

- Low-Level Access
- High-Level Access

In the first case, programmers can use and access the basic socket support for the operating system using Python's libraries, and programmers can implement both connection-less and connection-oriented protocols for programming.

Application-level network protocols can also be accessed using high-level access provided by Python libraries. These protocols are HTTP, FTP, etc.

A socket is the end-point in a flow of communication between two programs or communication channels operating over a network. They are created using a set of programming requests called socket API (Application Programming Interface). Python's socket library offers classes for handling common transports as a generic interface.

Sockets use protocols for determining the connection type for port-to-port communication between client and server machines. The protocols are used for:

- Domain Name Servers (DNS)
- IP addressing
- E-mail
- FTP (File Transfer Protocol) etc