



Mawlana Bhashani Science and Technology University

Lab-Report

Course Title : Computer Networks Lab

Lab Report No: 02

Lab Report Name: Programming with Python

Submitted by

Name: Ali Ashadullah Arif &
Ahadul Haque
ID: IT-18031 & IT-18045
3rd Year 2nd Semester
Session: 2017-2018
Dept. of ICT,
MBSTU.

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 a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable, reuse it and save some time. Also functions are a key way to define interfaces so programmers can share their code. 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.

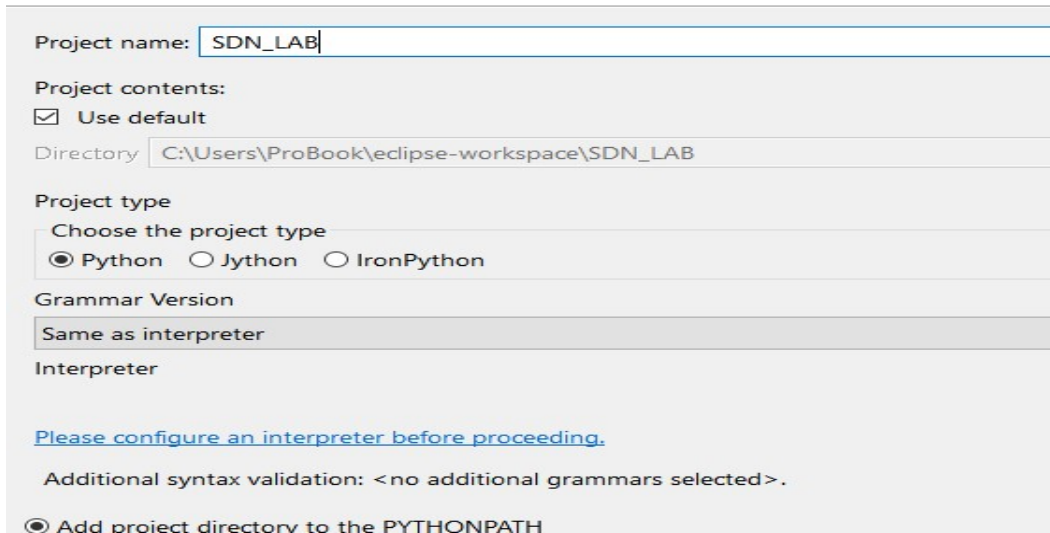
Local Variables: Variables declared inside a function definition are not related in any way to other variables with the same names used outside the function (variable names are local to the function). This is called the scope of the variable. All variables have the scope of the block they are declared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



The screenshot shows the 'New Python Project' dialog in Eclipse. The 'Project name' field is set to 'SDN_LAB'. Under 'Project contents', the 'Use default' checkbox is checked. The 'Directory' is 'C:\Users\ProBook\eclipse-workspace\SDN_LAB'. Under 'Project type', 'Python' is selected. Under 'Grammar Version', 'Same as interpreter' is selected. Under 'Interpreter', there is a link 'Please configure an interpreter before proceeding.' and the text 'Additional syntax validation: <no additional grammars selected>'. At the bottom, the checkbox 'Add project directory to the PYTHONPATH' is checked.

Project name:

Project contents:

☒ Use default

Directory

Project type

Choose the project type

☒ Python ☐ Jython ☐ IronPython

Grammar Version

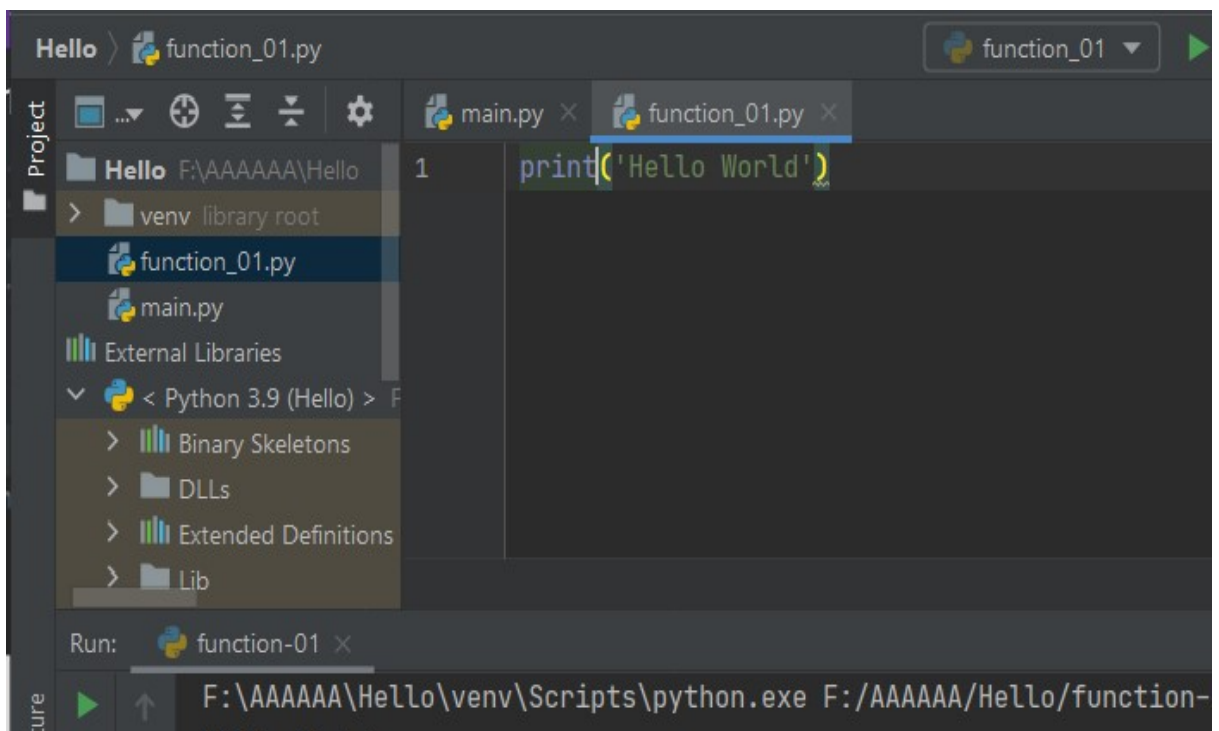
Interpreter

[Please configure an interpreter before proceeding.](#)

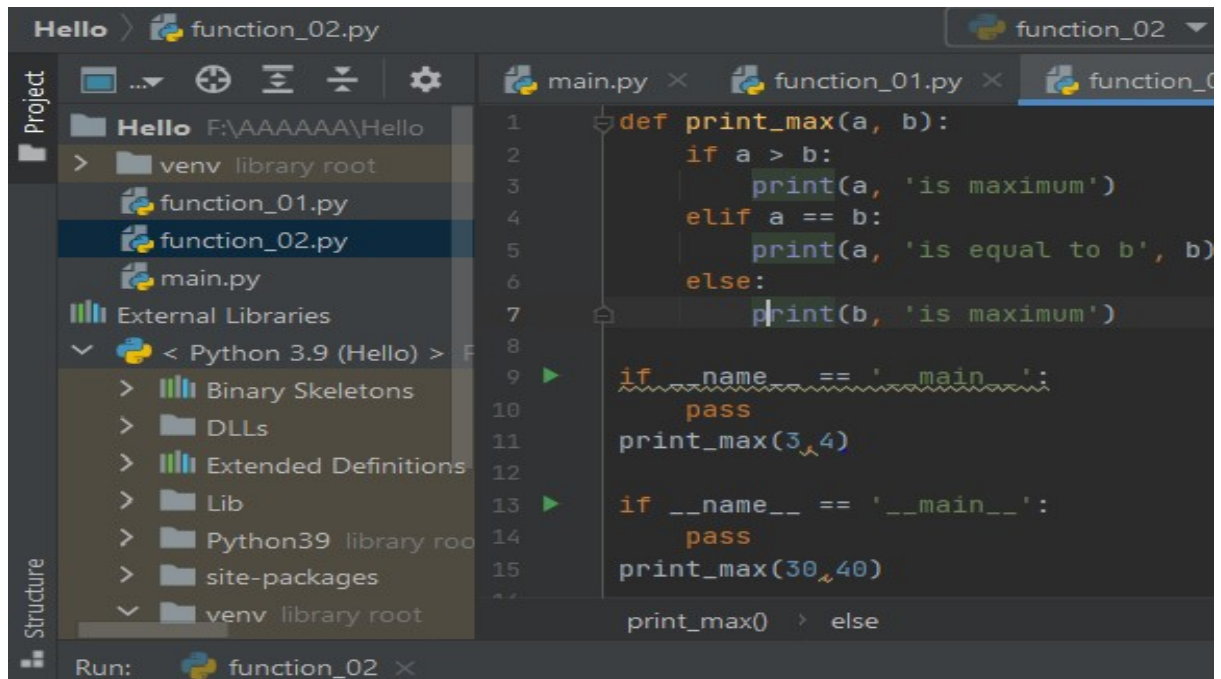
Additional syntax validation: <no additional grammars selected>.

☒ Add project directory to the PYTHONPATH

Exercise 4.1.2: Python function (save as function_01.py)



Exercise 4.1.3: Python function (save as function_02.py)

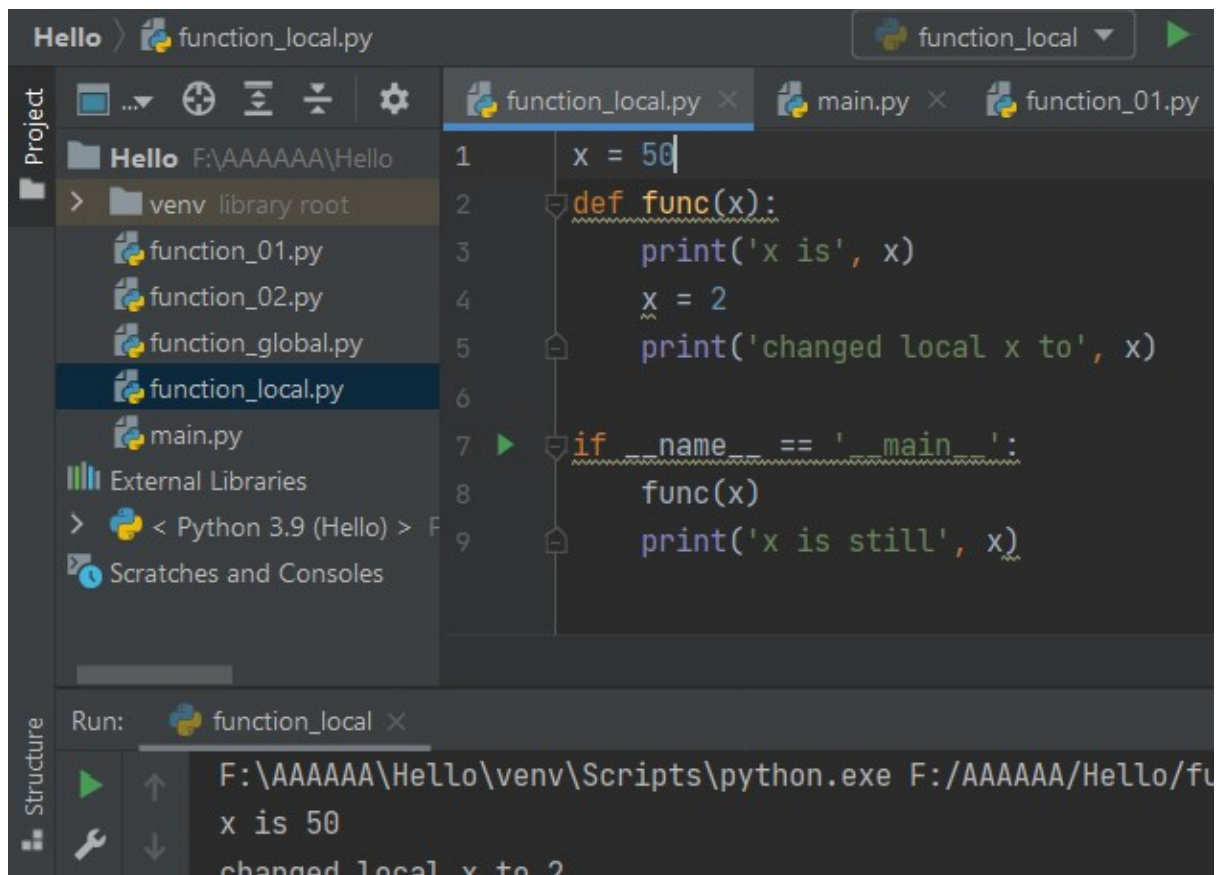


```
def print_max(a, b):
    if a > b:
        print(a, 'is maximum')
    elif a == b:
        print(a, 'is equal to b', b)
    else:
        print(b, 'is maximum')

if __name__ == '__main__':
    pass
print_max(3, 4)

if __name__ == '__main__':
    pass
print_max(30, 40)
```

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



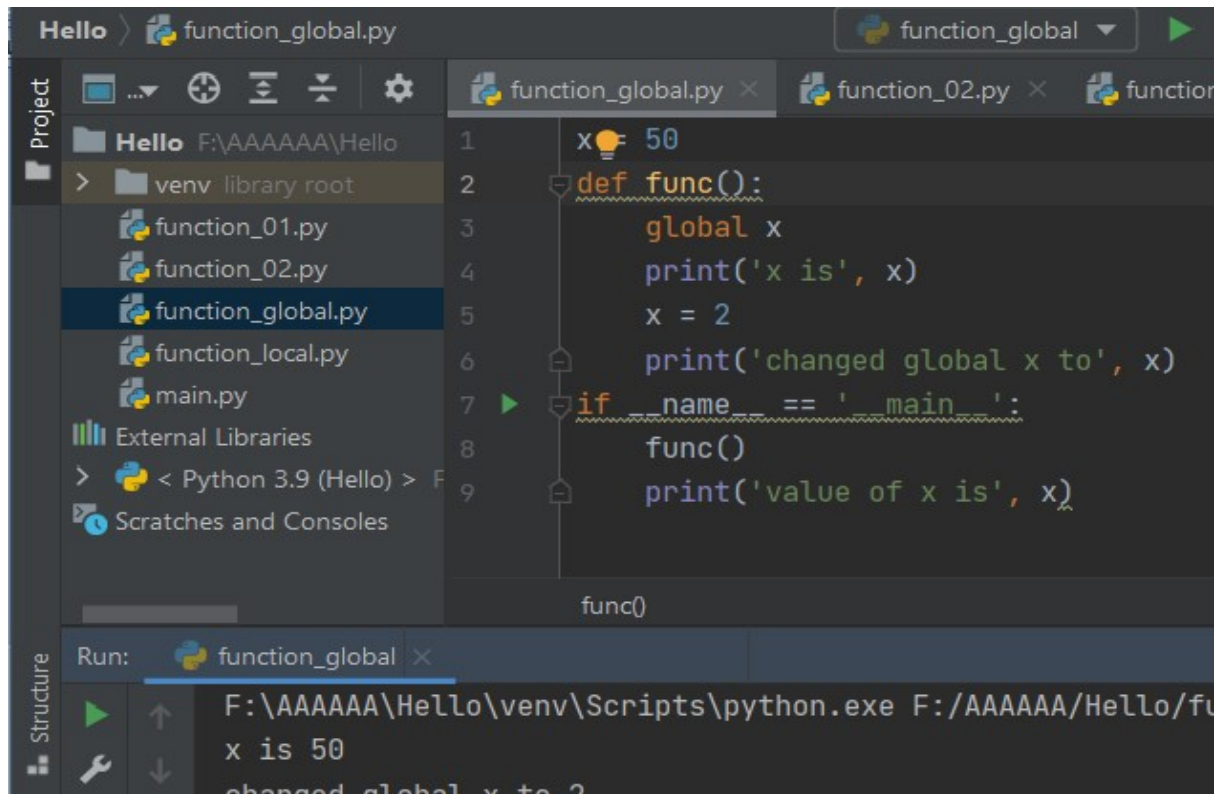
```
x = 50
def func(x):
    print('x is', x)
    x = 2
    print('changed local x to', x)

if __name__ == '__main__':
    func(x)
    print('x is still', x)
```

Run: function_local ×

F:\AAAAAA\Hello\venv\Scripts\python.exe F:/AAAAAA/Hello/fu
x is 50
changed local x to 2

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

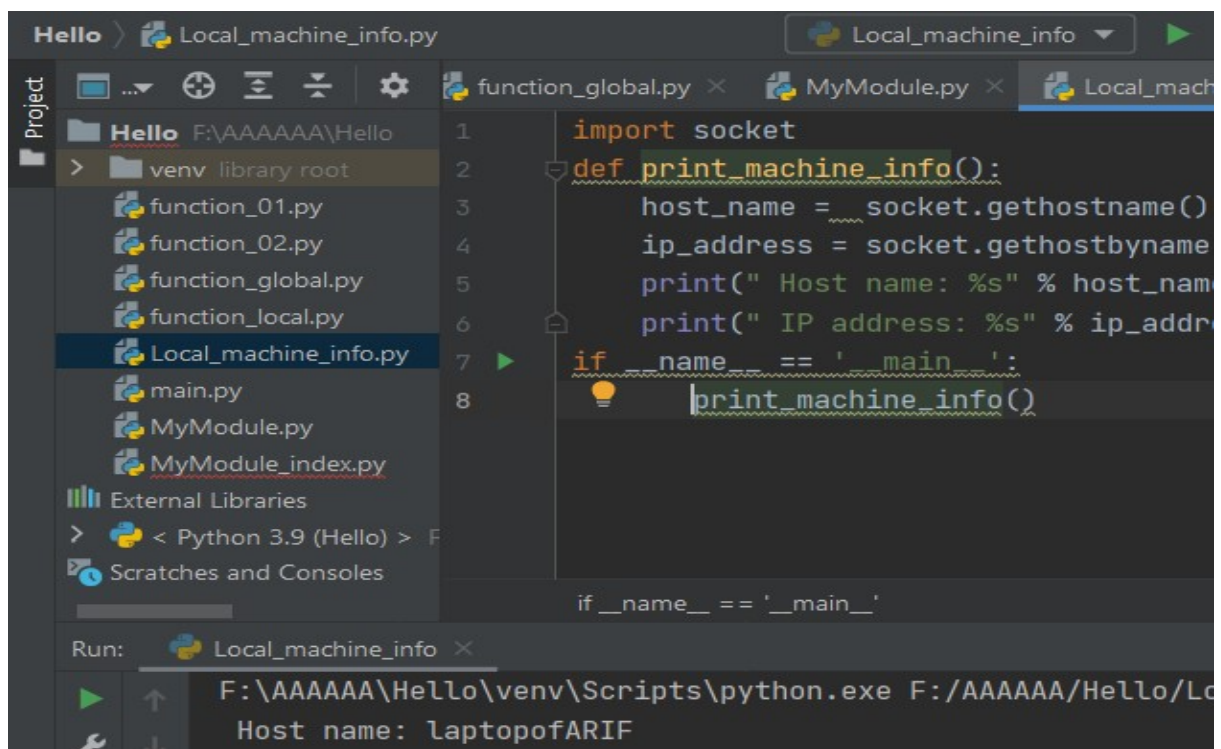


```
1 x = 50
2 def func():
3     global x
4     print('x is', x)
5     x = 2
6     print('changed global x to', x)
7 if __name__ == '__main__':
8     func()
9     print('value of x is', x)
```

Run: function_global

F:\AAAAAA\Hello\venv\Scripts\python.exe F:/AAAAAA/Hello/f
x is 50
changed global x to 2

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

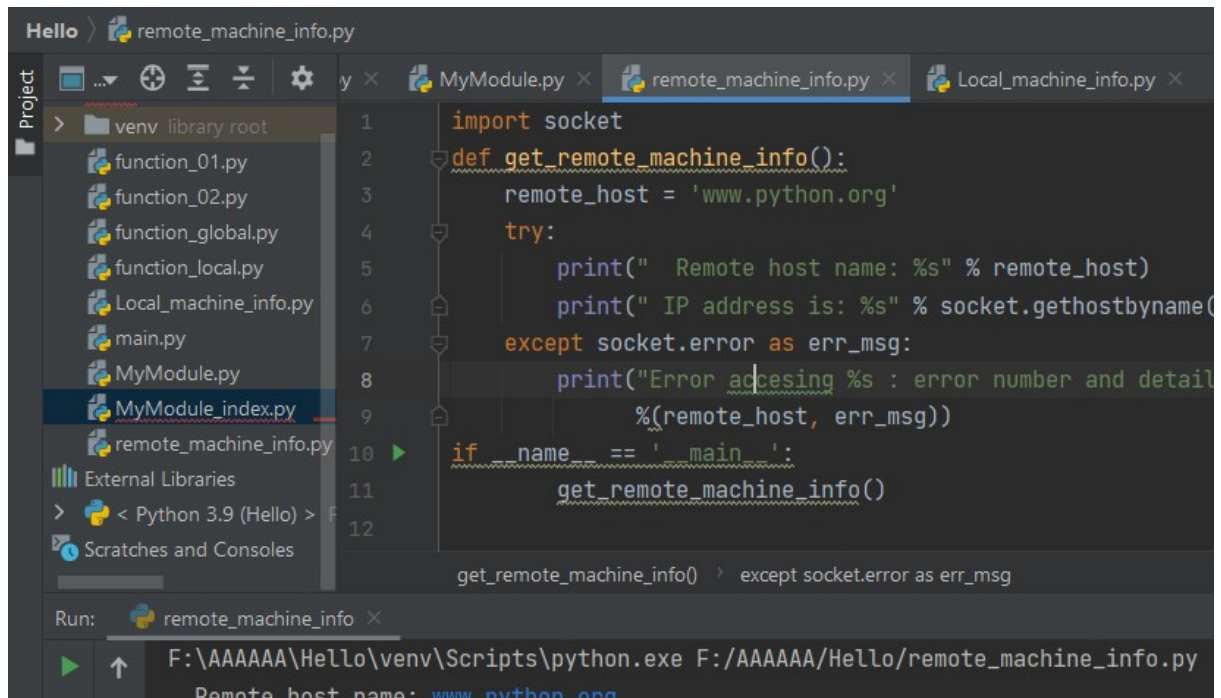


```
1 import socket
2 def print_machine_info():
3     host_name = socket.gethostname()
4     ip_address = socket.gethostbyname(host_name)
5     print(" Host name: %s" % host_name)
6     print(" IP address: %s" % ip_address)
7 if __name__ == '__main__':
8     print_machine_info()
```

Run: Local_machine_info

F:\AAAAAA\Hello\venv\Scripts\python.exe F:/AAAAAA/Hello/Lc
Host name: laptopofARIF

Exercise 4.2.2: Retrieving a remote machine's IP address



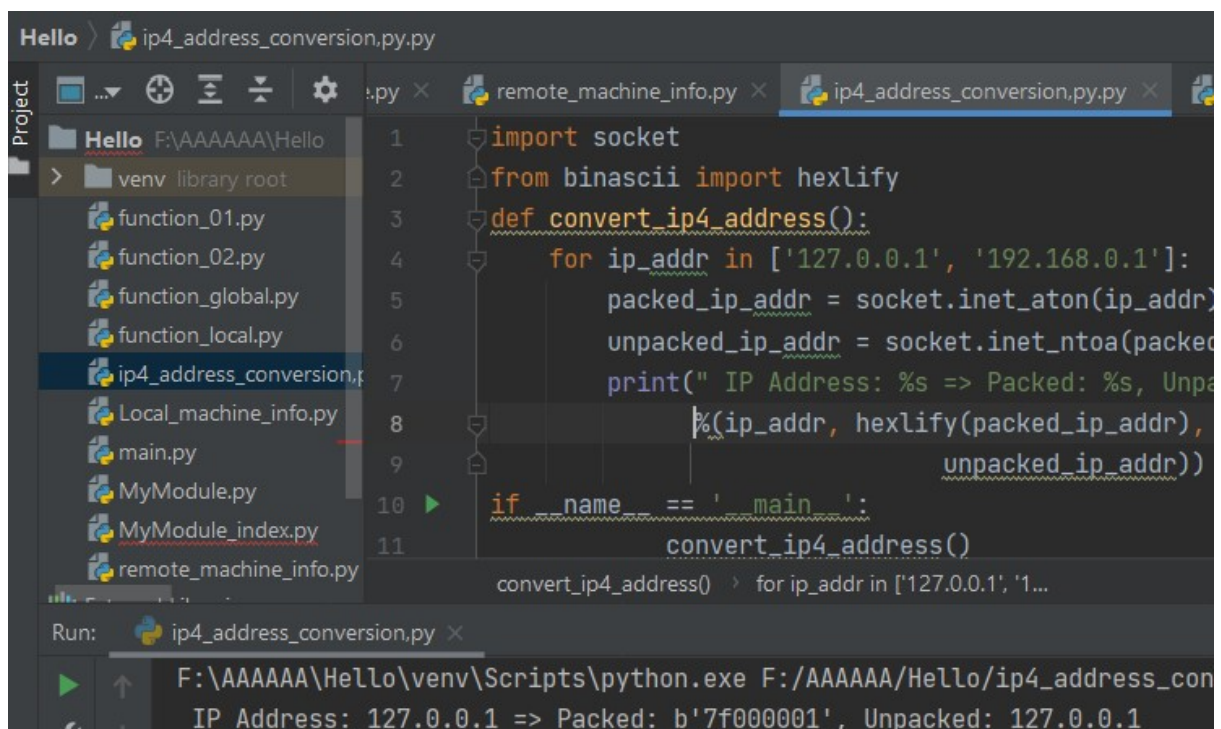
```
1 import socket
2 def get_remote_machine_info():
3     remote_host = 'www.python.org'
4     try:
5         print(" Remote host name: %s" % remote_host)
6         print(" IP address is: %s" % socket.gethostbyname(remote_host))
7     except socket.error as err_msg:
8         print("Error accessing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

Run: remote_machine_info

F:\AAAAAA\Hello\venv\Scripts\python.exe F:/AAAAAA/Hello/remote_machine_info.py

Remote host name: www.python.org

Exercise 4.2.3: Converting an IPv4 address to different formats



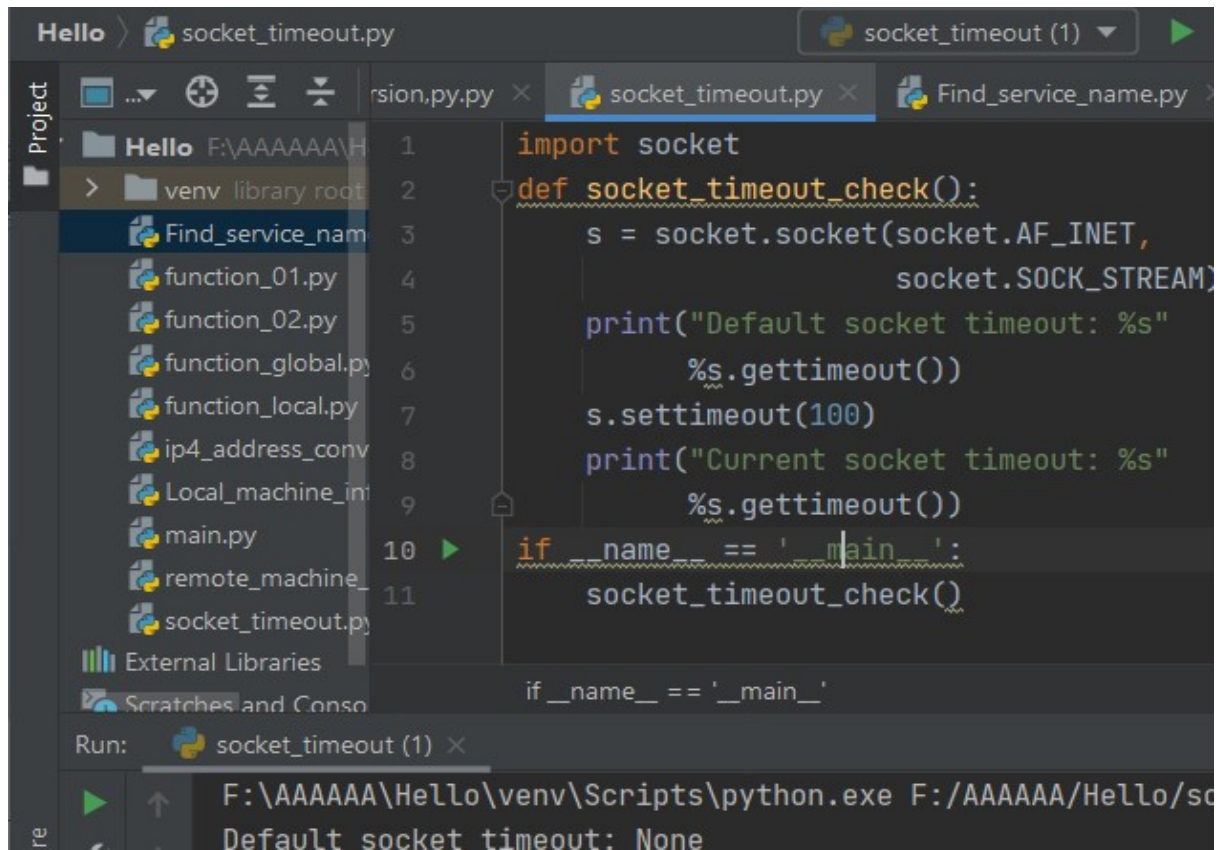
```
1 import socket
2 from binascii import hexlify
3 def convert_ip4_address():
4     for ip_addr in ['127.0.0.1', '192.168.0.1']:
5         packed_ip_addr = socket.inet_aton(ip_addr)
6         unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
7         print(" IP Address: %s => Packed: %s, Unpacked: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

Run: ip4_address_conversion.py

F:\AAAAAA\Hello\venv\Scripts\python.exe F:/AAAAAA/Hello/ip4_address_conversion.py

IP Address: 127.0.0.1 => Packed: b'7f000001', Unpacked: 127.0.0.1

Exercise 4.2.4: Setting and getting the default socket timeout



The screenshot shows an IDE with a project named 'Hello' located at 'F:\AAAAAA\H'. The file explorer on the left lists several files, including 'socket_timeout.py'. The main editor window displays the code for 'socket_timeout.py'.

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

if __name__ == '__main__':
```

The Run console at the bottom shows the command: `F:\AAAAAA\Hello\venv\Scripts\python.exe F:/AAAAAA/Hello/sc` and the output: `Default socket timeout: None`.

Exercise 4.2.4: Writing a simple echo client/server application (Tip: Use port 9900)

Server Code:

```
import socket
import sys
import argparse
import codecs
from codecs import encode, decode
host = 'localhost'
data_payload = 4096
backlog = 5
def echo_server(port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_address = (host, port)
    print('Starting up echo server on %s port %s' % server_address)
    sock.listen(backlog)
    while True:
        print('Waiting to receive message from client')
        client, address = sock.accept()
        data = client.recv(data_payload)
        if data:
            print('Data : %s' % data)
            client.send(data)
            print('Sent %s bytes back to %s' % (data, address))
            client.close()
if __name__ == '__main__':
```


Client Code:

```
ess_conversion.py.py x socket_timeout.py x server_echo.py x server_echo.py
2 import sys
3 import argparse
4 import codecs
5 from codecs import encode, decode
6 host = 'localhost'
7 def echo_client(port):
8     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9     server_address = (host, port)
10    print('Connecting to %s port %s' % server_address)
11    sock.connect(server_address)
12    try:
13        message = "Test message: SDN course examples"
14        print("Sending %s" % message)
15        sock.sendall(message.encode('utf_8'))
16        amount_received = 0
17        amount_expected = len(message)
18        while
19
22    print("Received: %s" % data) except socket.error
23    as e:
24        print("Socket error: %s" % str(e)) except Exception:
25        print("Other exception: %s" % str(e)) finally:
26        print("Closing connection to the server")
27        sock.close()
28 if __name__ == '__main__':
29     parser = argparse.ArgumentParser(description='Socket
30     parser.add_argument('-p', action="store",
31                         dest="port", type = int, required
32     given_args = parser.parse_args()
```

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.