MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

Santosh,Tangail – 1902

Course Title	**: Computer Networks Lab**
Lab Report Name	**: Programming with Python**
Lab Report No.	**: 05**

Submitted by,

Name : ANIKA JAHIN

ID: IT-17056

Session: 2016-17

Dept. of ICT,MBSTU.

Submitted to,

NAZRUL ISLAM

Assistant Professor

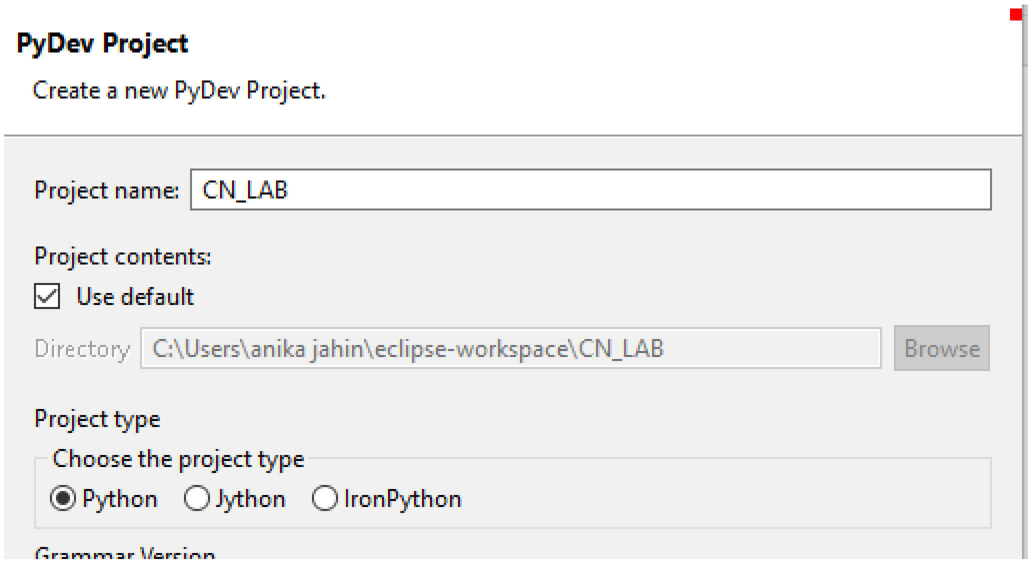
Dept. of ICT,MBSTU.

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 using with CN_LAB



PyDev Project

Create a new PyDev Project.

Project name: CN_LAB

Project contents:

☑ Use default

Directory C:\Users\anika jahin\eclipse-workspace\CN_LAB

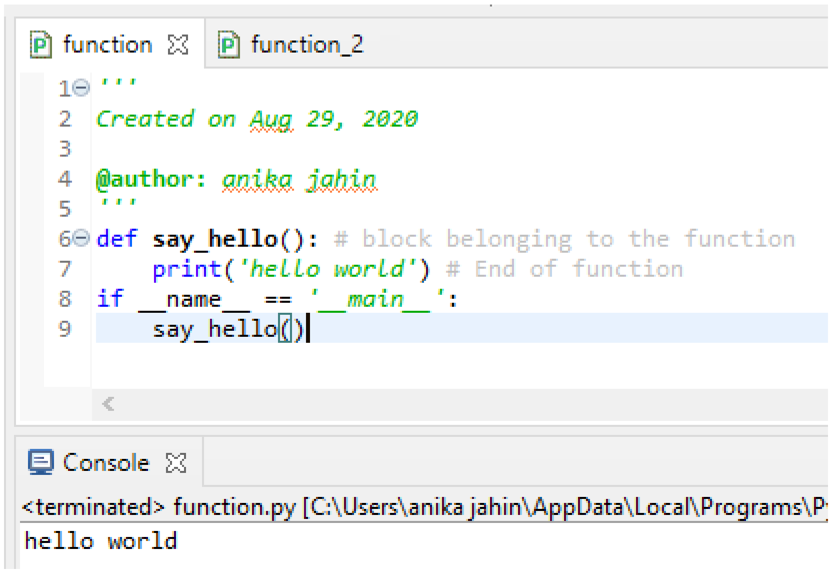
Project type

Choose the project type

[✓ Python] Jython IronPython

Grammar Version

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

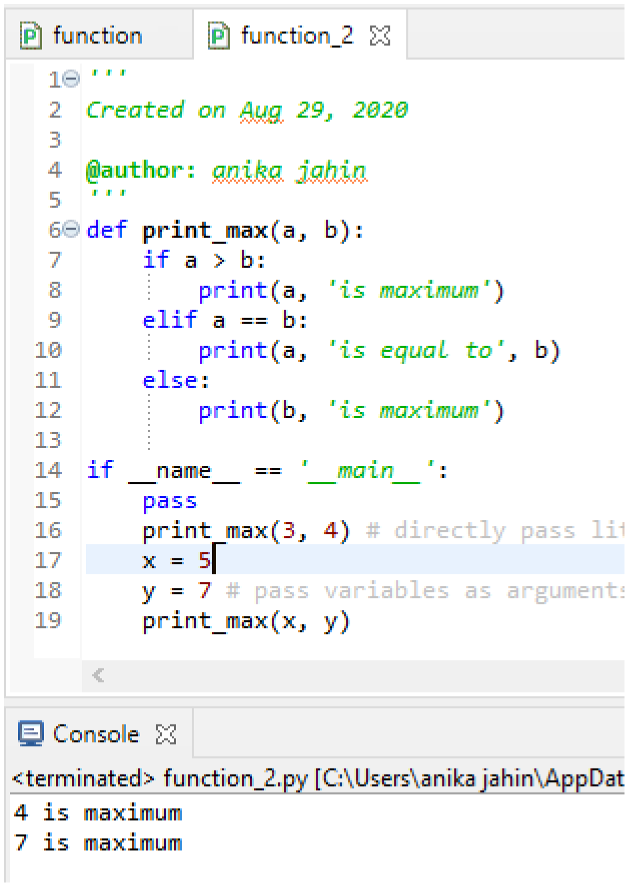

function ⌧ function_2

```python
'''
Created on Aug 29, 2020

@author: anika jahin
'''
def say_hello(): # block belonging to the function
    print('hello world') # End of function
if __name__ == '__main__':
    say_hello()
```

Console ⌧

```
<terminated> function.py [C:\Users\anika jahin\AppData\Local\Programs\P
hello world
```

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

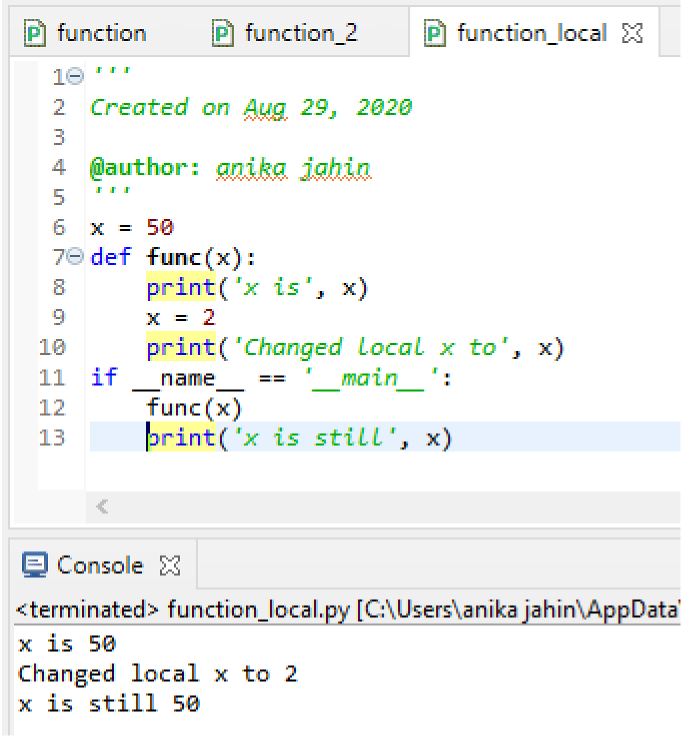

function function_2 ⌧

```python
'''
Created on Aug 29, 2020

@author: anika jahin
'''
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 lit
    x = 5
    y = 7 # pass variables as argument
    print_max(x, y)
```

Console ⌧

```
<terminated> function_2.py [C:\Users\anika jahin\AppDat
4 is maximum
7 is maximum
```

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

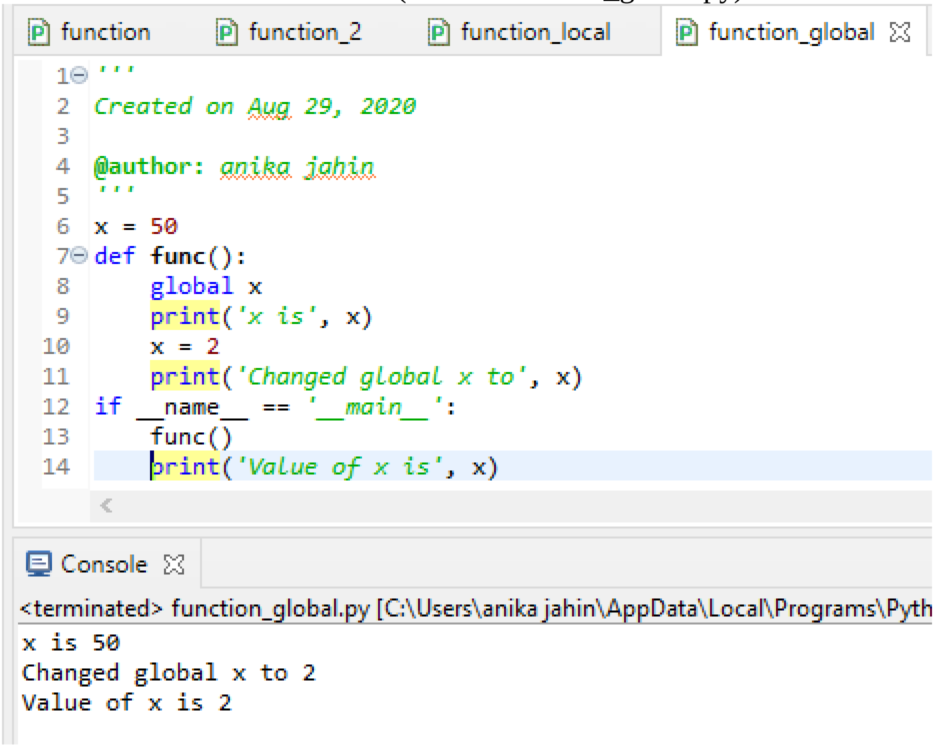

function   function_2   function_local

```python
'''
Created on Aug 29, 2020

@author: anika jahin
'''
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)
```

Console

```
<terminated> function_local.py [C:\Users\anika jahin\AppData
x is 50
Changed local x to 2
x is still 50
```

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

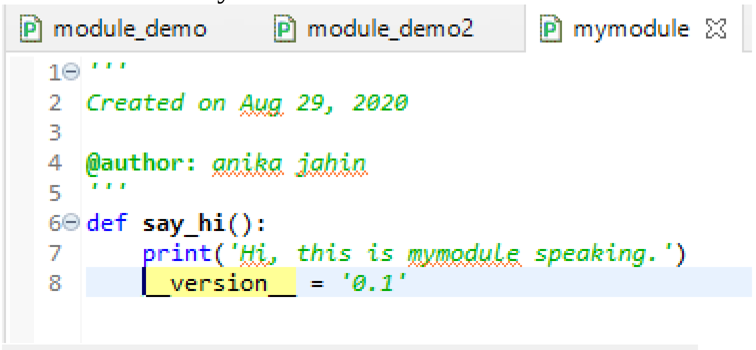

function   function_2   function_local   function_global

```python
'''
Created on Aug 29, 2020

@author: anika jahin
'''
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)
```

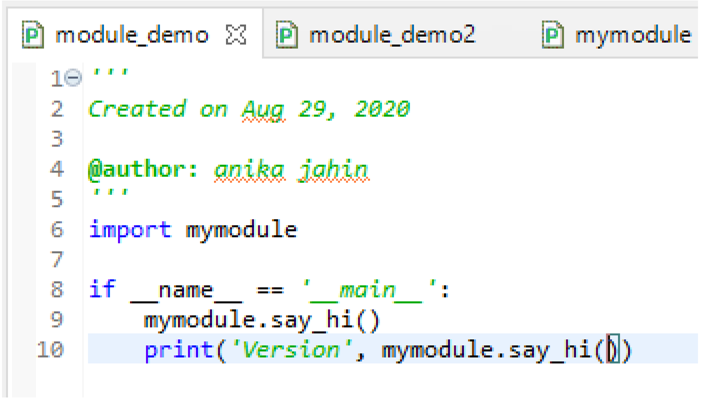
Console

```
<terminated> function_global.py [C:\Users\anika jahin\AppData\Local\Programs\Pyth
x is 50
Changed global x to 2
Value of x is 2
```

Exercise 4.1.6: Python modules

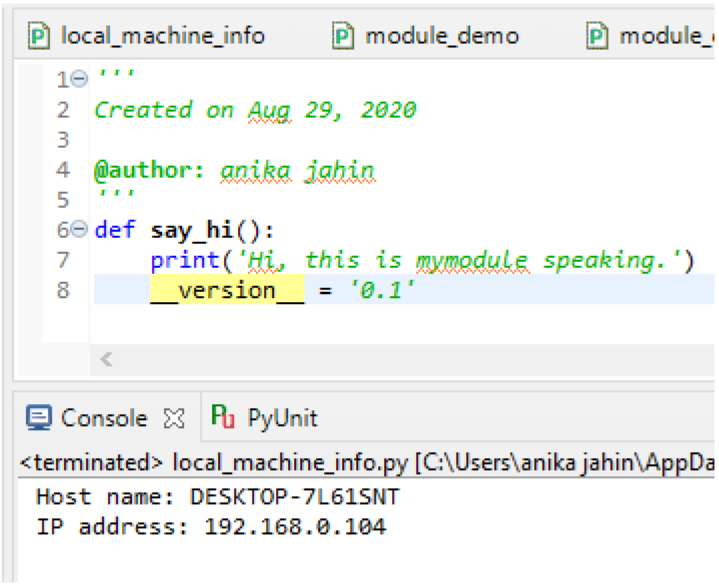

module_demo module_demo2 mymodule

```
'''
Created on Aug 29, 2020

@author: anika jahin
'''
def say_hi():
    print('Hi, this is mymodule speaking.')
    __version__ = '0.1'
```


module_demo module_demo2 mymodule

```
'''
Created on Aug 29, 2020

@author: anika jahin
'''
import mymodule

if __name__ == '__main__':
    mymodule.say_hi()
    print('Version', mymodule.say_hi())
```

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

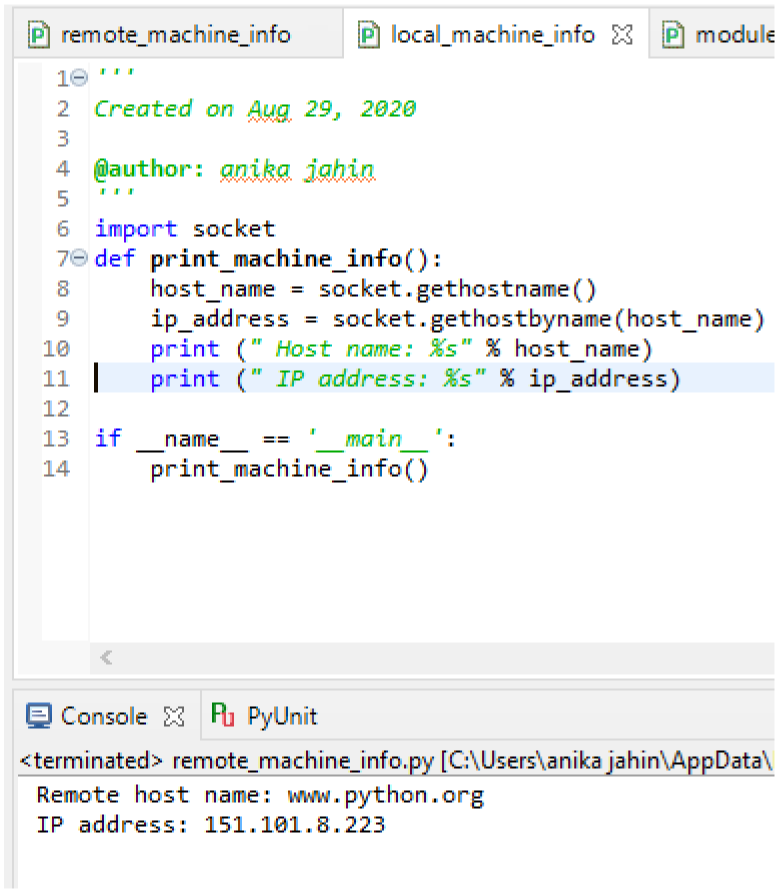

local_machine_info module_demo module_

```
'''
Created on Aug 29, 2020

@author: anika jahin
'''
def say_hi():
    print('Hi, this is mymodule speaking.')
    __version__ = '0.1'
```

Console PyUnit

```
<terminated> local_machine_info.py [C:\Users\anika jahin\AppDa
Host name: DESKTOP-7L61SNT
IP address: 192.168.0.104
```

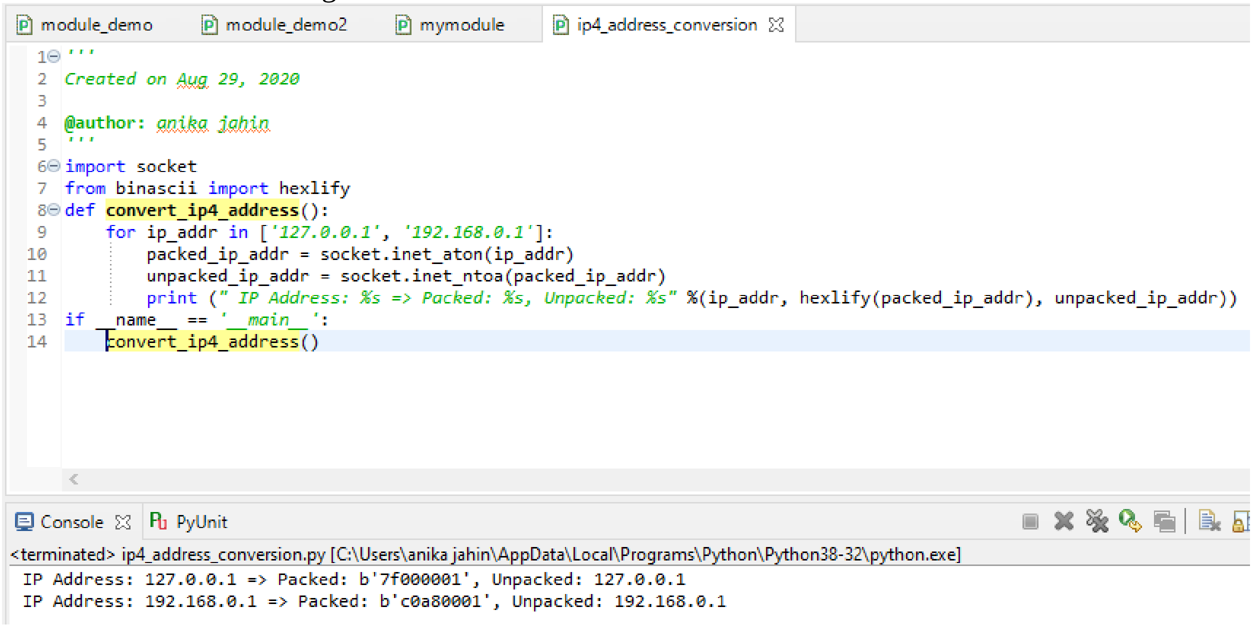
Exercise 4.2.2: Retrieving a remote machine's IP address



```
'''
Created on Aug 29, 2020

@author: anika jahin
'''
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()
```

```
<terminated> remote_machine_info.py [C:\Users\anika jahin\AppData\
 Remote host name: www.python.org
 IP address: 151.101.8.223
```

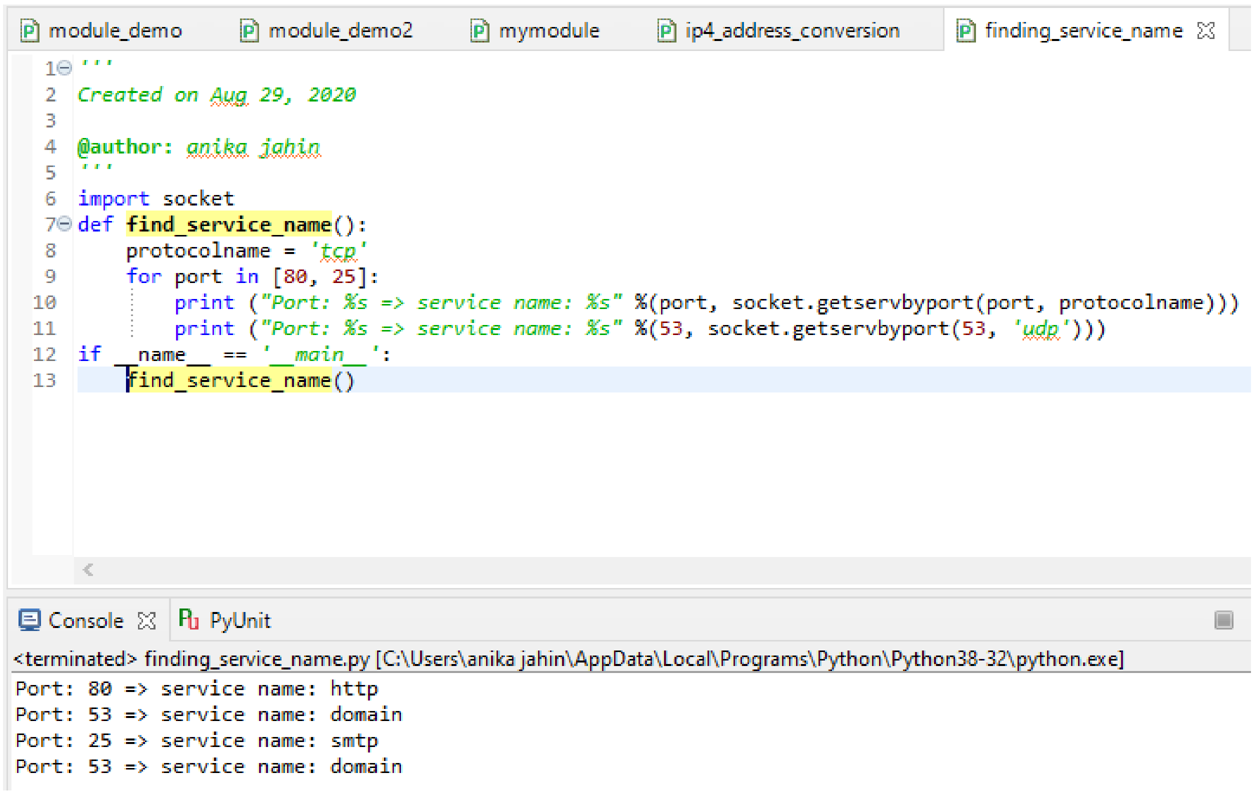
Exercise 4.2.3: Converting an IPv4 address to different formats



```
'''
Created on Aug 29, 2020

@author: anika jahin
'''
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()
```

```
<terminated> ip4_address_conversion.py [C:\Users\anika jahin\AppData\Local\Programs\Python\Python38-32\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
```

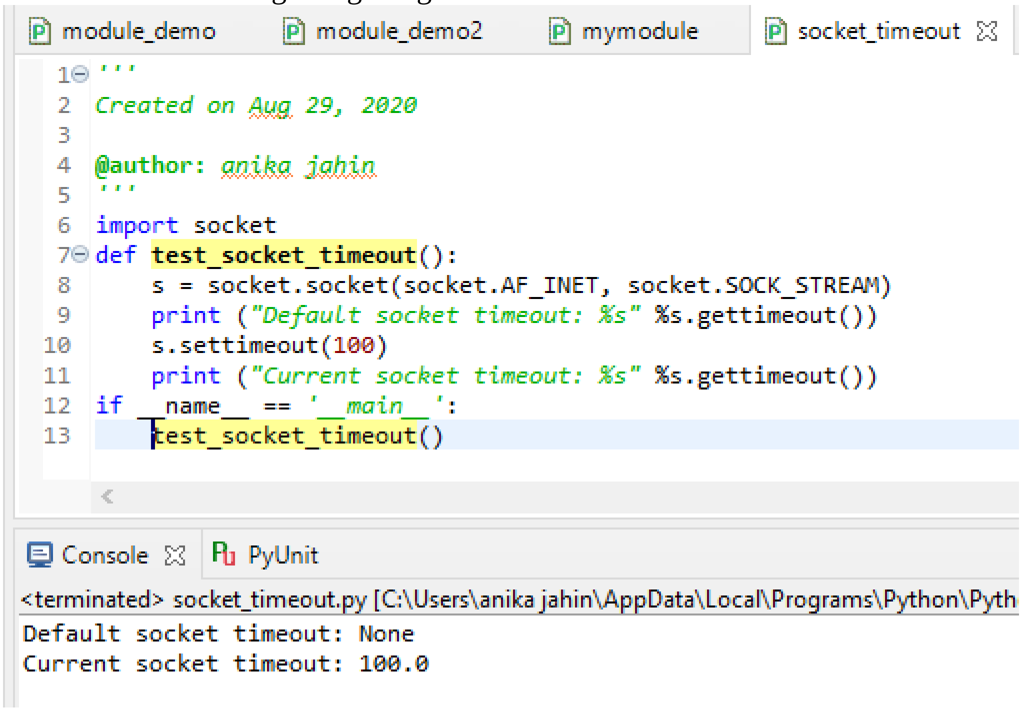
Exercise 4.2.4: Finding a service name, given the port and protocol



```python
'''
Created on Aug 29, 2020

@author: anika jahin
'''
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()
```

```
<terminated> finding_service_name.py [C:\Users\anika jahin\AppData\Local\Programs\Python\Python38-32\python.exe]
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

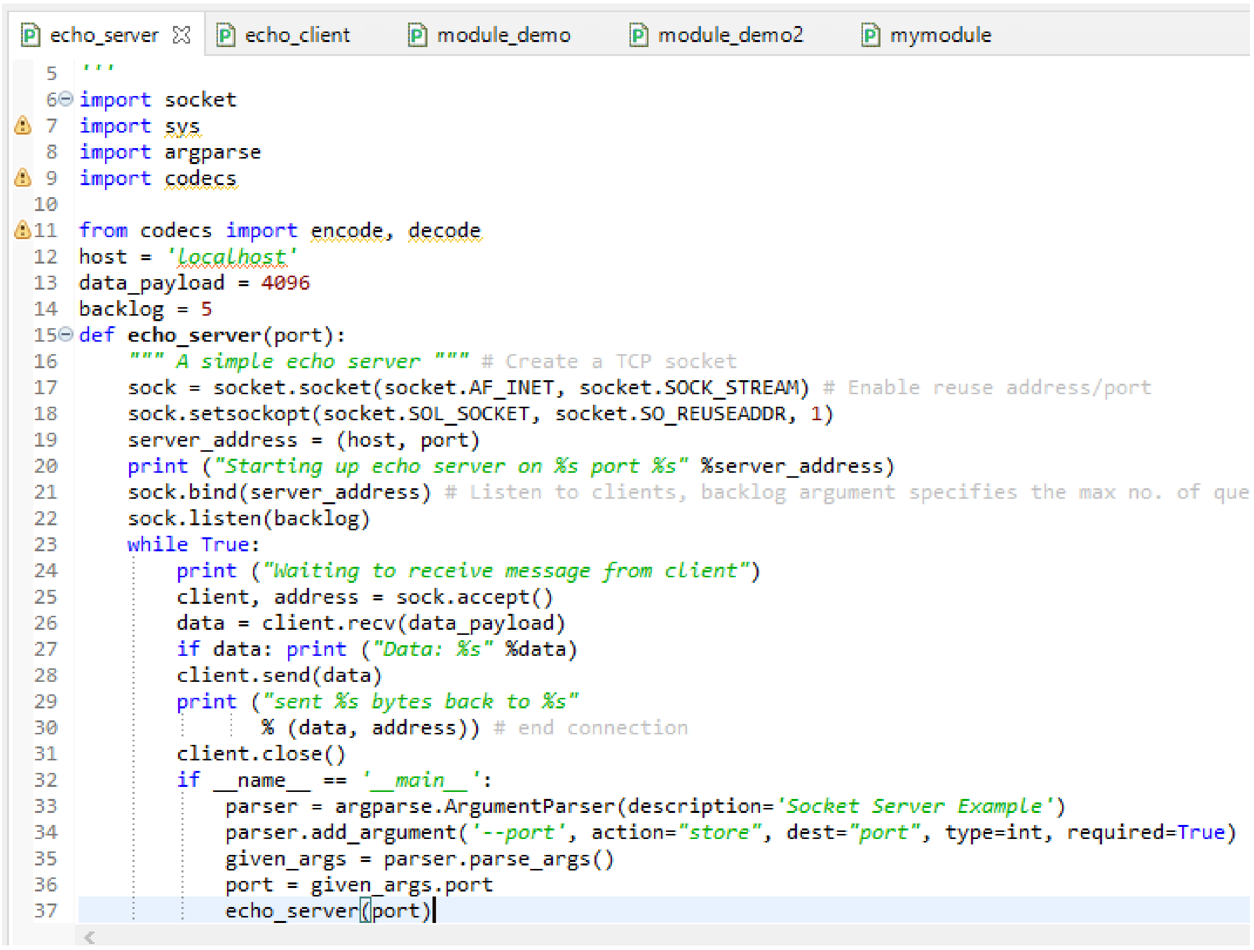


```python
'''
Created on Aug 29, 2020

@author: anika jahin
'''
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()
```

```
<terminated> socket_timeout.py [C:\Users\anika jahin\AppData\Local\Programs\Python\Pyth
Default socket timeout: None
Current socket timeout: 100.0
```

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

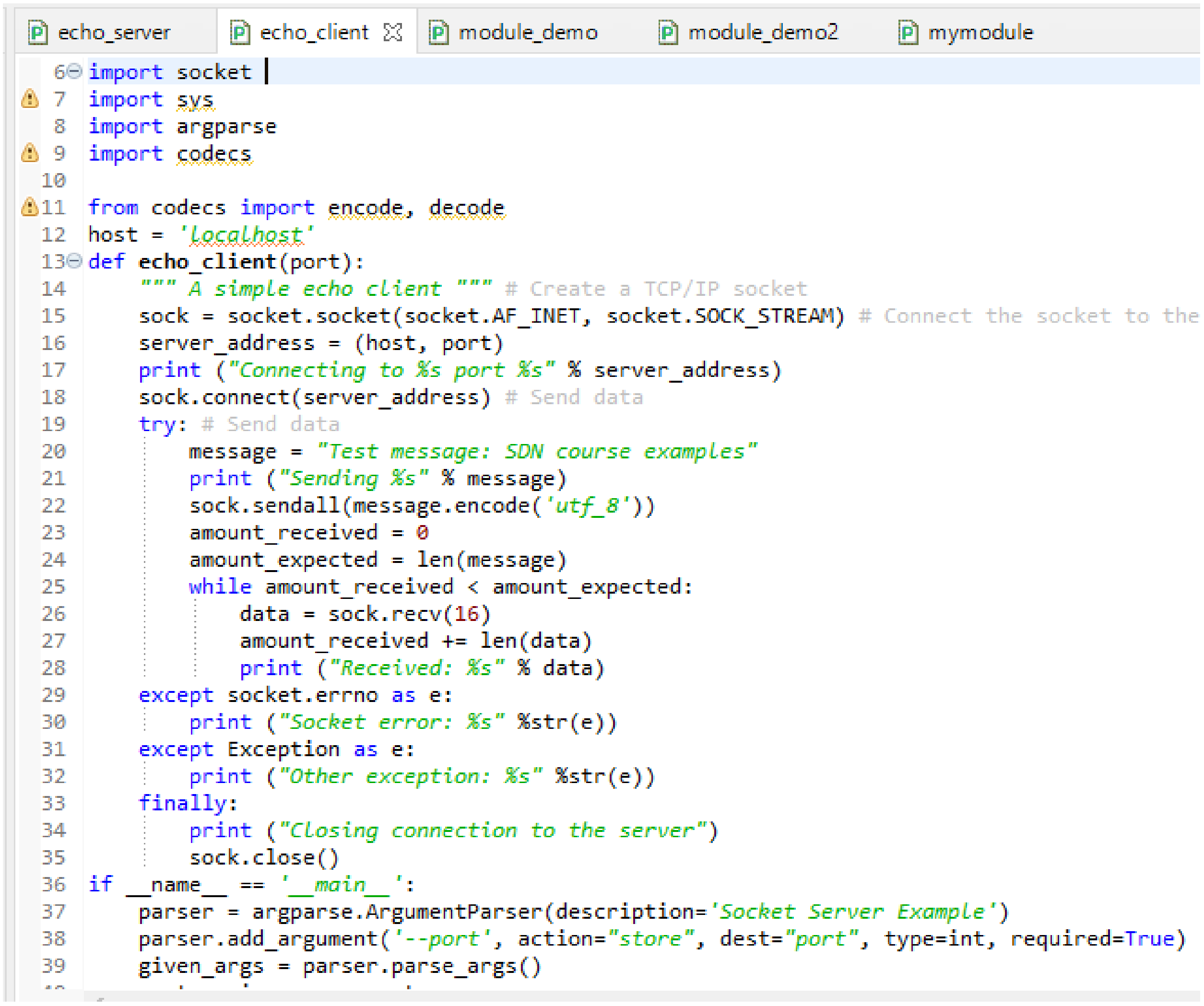
Server code:



echo_server | echo_client | module_demo | module_demo2 | mymodule

```
'''
import socket
import sys
import argparse
import codecs

from codecs import encode, decode
host = 'localhost'
data_payload = 4096
backlog = 5
def echo_server(port):
    """ A simple echo server """ # Create a TCP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Enable reuse address/port
    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.bind(server_address) # Listen to clients, backlog argument specifies the max no. of que
    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)) # end connection
        client.close()
        if __name__ == '__main__':
            parser = argparse.ArgumentParser(description='Socket Server Example')
            parser.add_argument('--port', action="store", dest="port", type=int, required=True)
            given_args = parser.parse_args()
            port = given_args.port
            echo_server(port)
```

Client code:



```python
import socket
import sys
import argparse
import codecs

from codecs import encode, decode
host = 'localhost'
def echo_client(port):
    """ A simple echo client """ # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Connect the socket to the
    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    sock.connect(server_address) # Send data
    try: # Send data
        message = "Test message: SDN course examples"
        print ("Sending %s" % message)
        sock.sendall(message.encode('utf_8'))
        amount_received = 0
        amount_expected = len(message)
        while amount_received < amount_expected:
            data = sock.recv(16)
            amount_received += len(data)
            print ("Received: %s" % data)
    except socket.errno as e:
        print ("Socket error: %s" %str(e))
    except Exception as e:
        print ("Other exception: %s" %str(e))
    finally:
        print ("Closing connection to the server")
        sock.close()
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    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...