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.

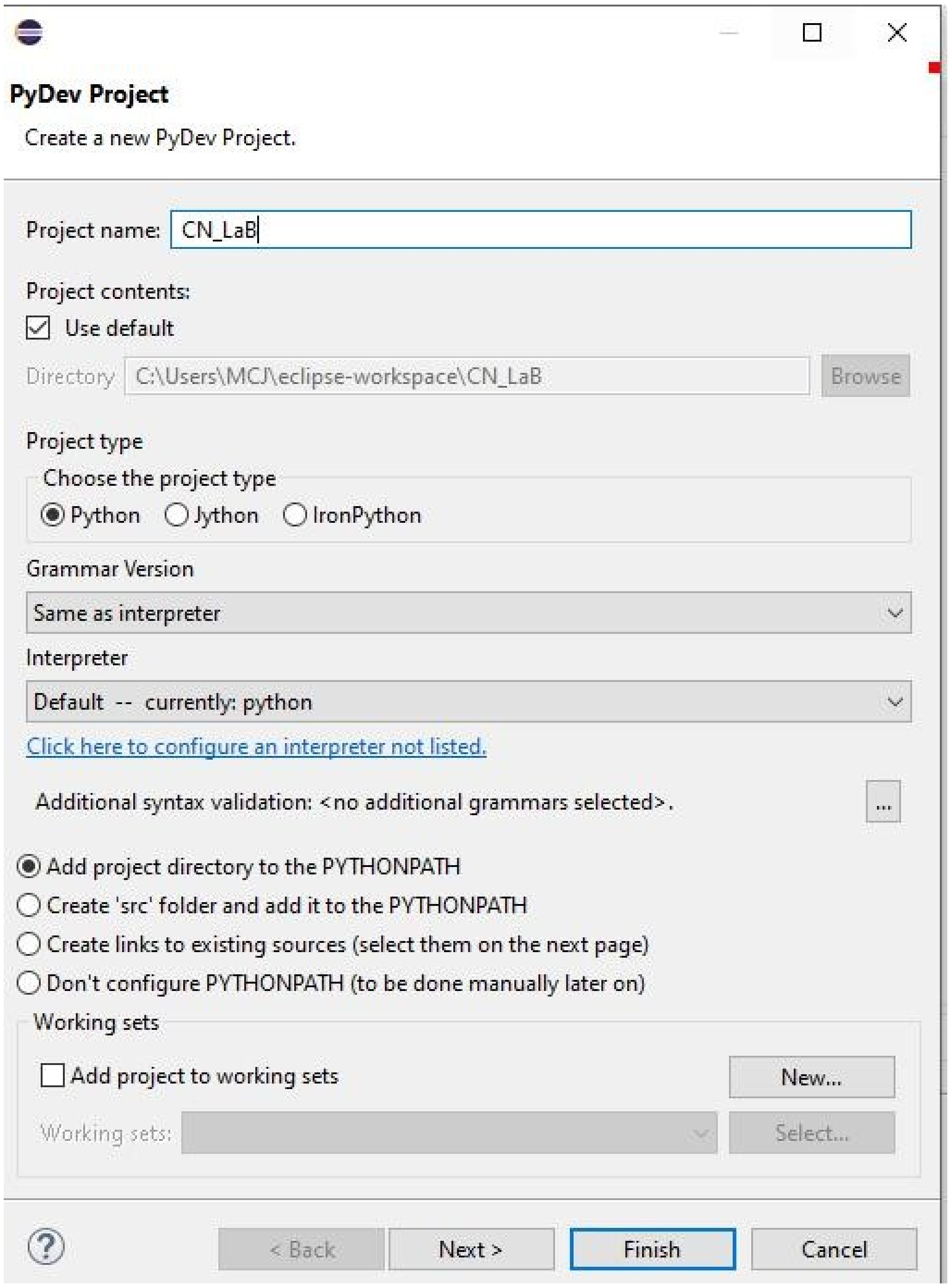
• **TCP**: TCP stands for transmission control protocol. It is implemented in the transport layer of the IP/TCP model and is used to establish reliable connections. TCP is one of the protocols that encapsulate data into packets. It then transfers these to the remote end of the connection using the methods 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.

• **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.

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\MCJ\eclipse-workspace\CN_LaB

Project type

Choose the project type

[✓ Python] Jython IronPython

Grammar Version

Same as interpreter

Interpreter

Default -- currently: python

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)

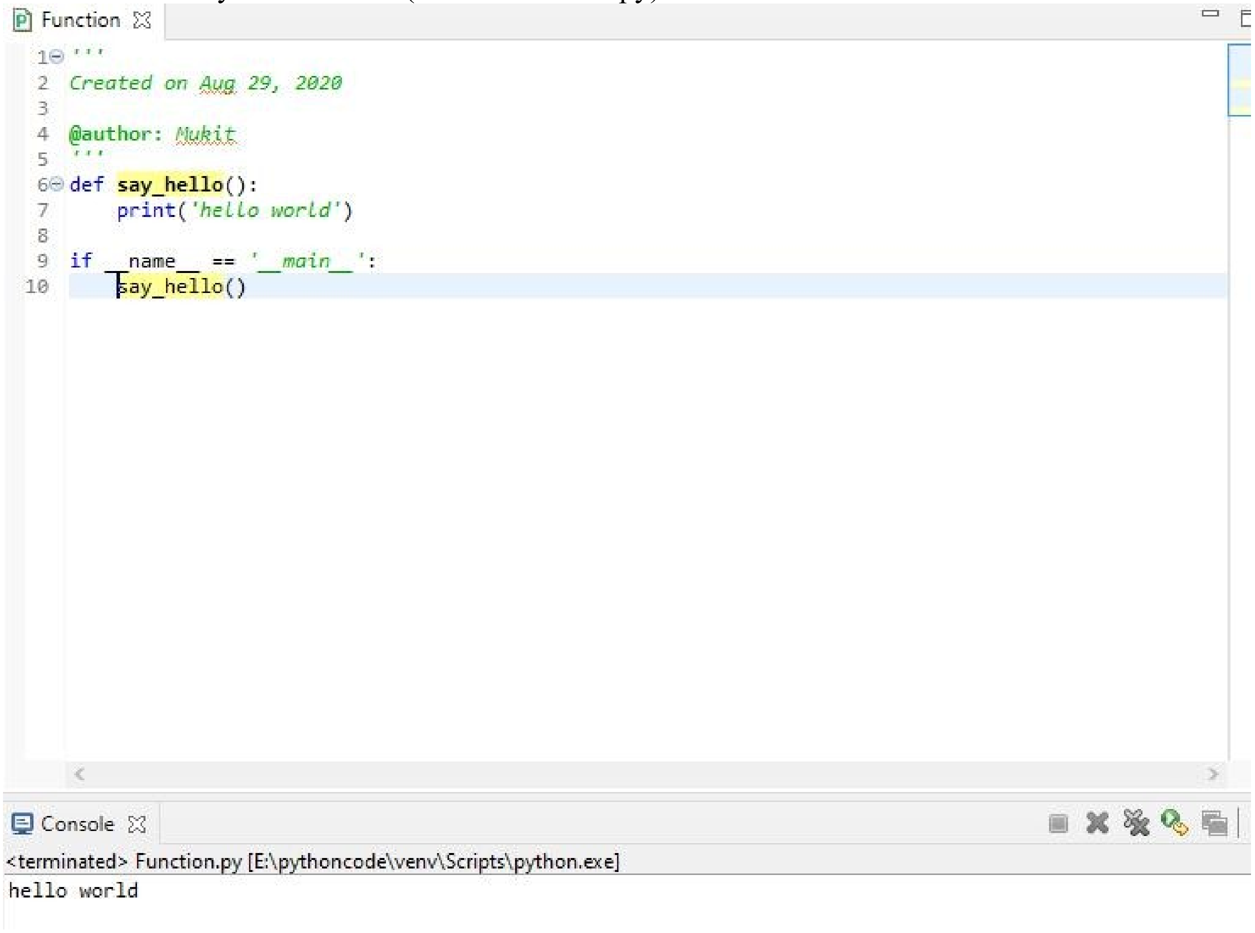
Don't configure PYTHONPATH (to be done manually later on)

Working sets

☐ Add project to working sets

Working sets:

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



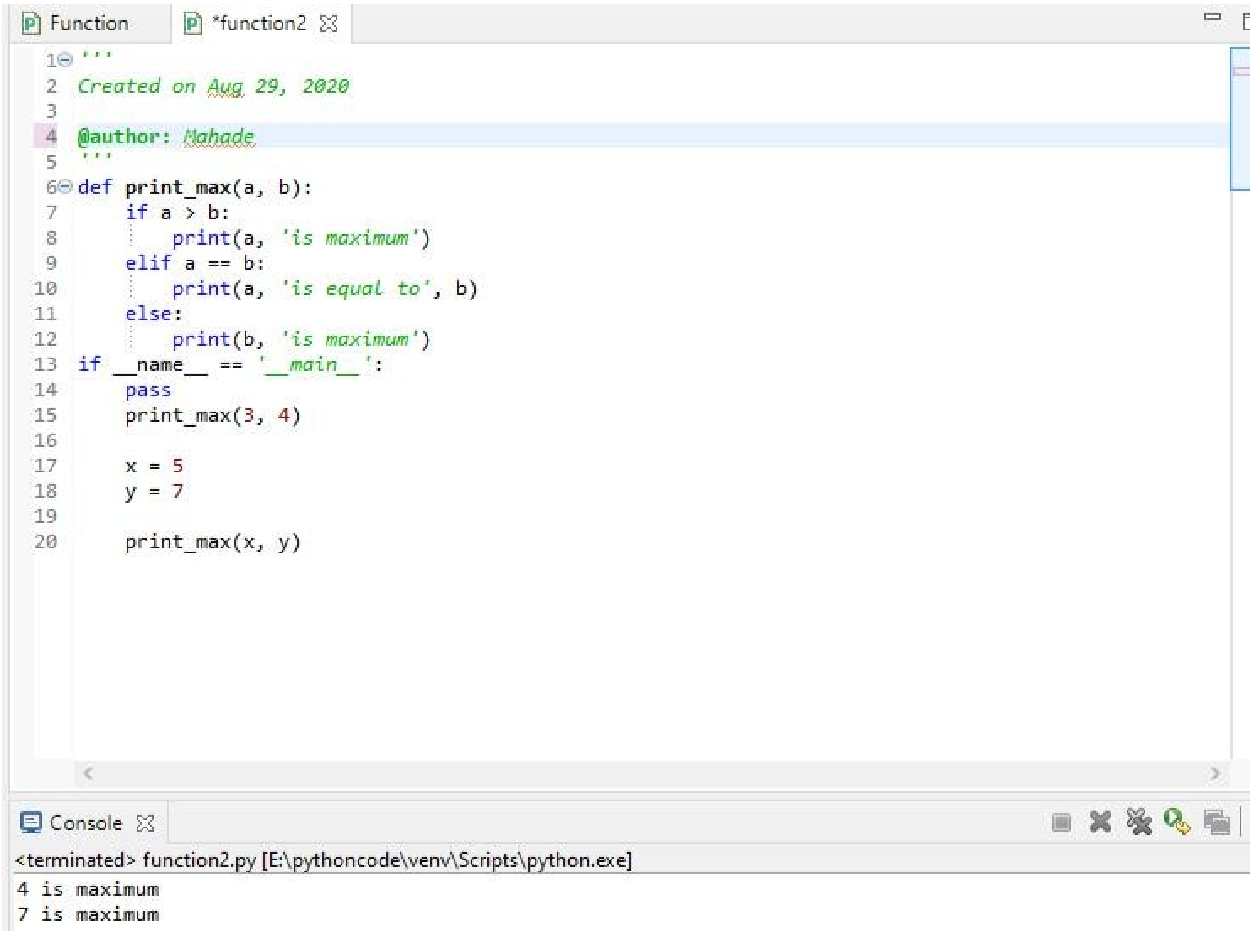
Function

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

@author: Mukit
'''
def say_hello():
    print('hello world')

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

Console

```
<terminated> Function.py [E:\pythoncode\venv\Scripts\python.exe]
hello world
```

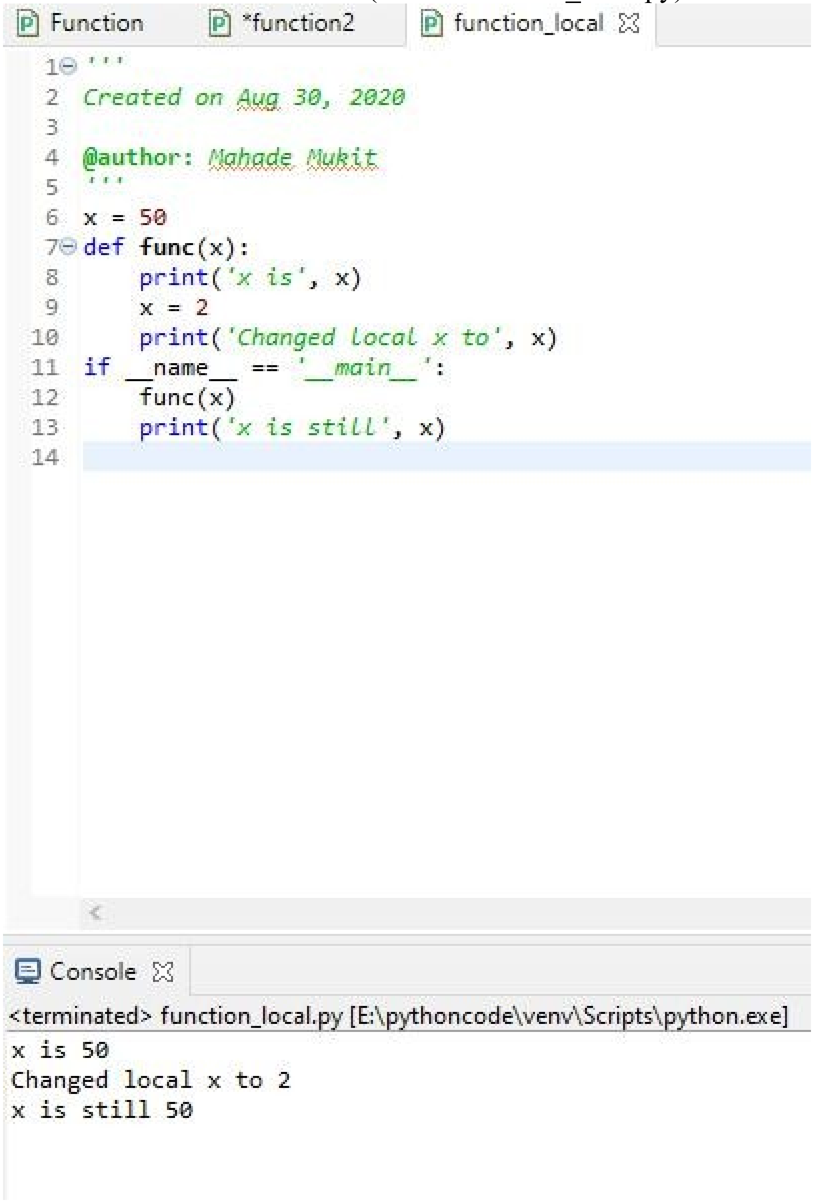
Exercise 4.1.3: Python function (save as function_2.py)



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

@author: Mahade
'''
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)

    x = 5
    y = 7

    print_max(x, y)
```

```
<terminated> function2.py [E:\pythoncode\venv\Scripts\python.exe]
4 is maximum
7 is maximum
```

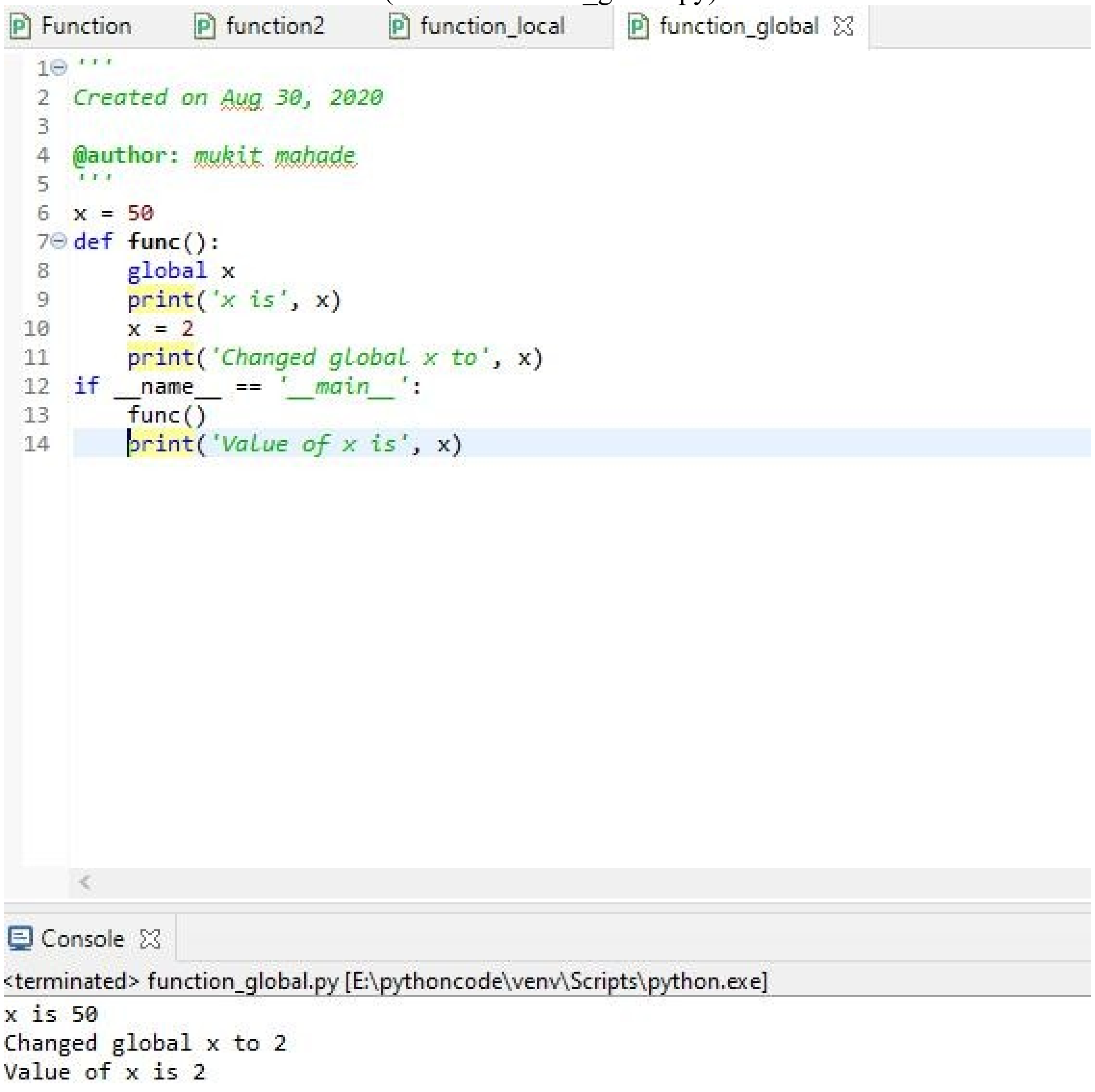
Exercise 4.1.4: Local variable (save as function_local.py)



```
'''
Created on Aug 30, 2020

@author: Mahade Mukit
'''
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)

```

```
<terminated> function_local.py [E:\pythoncode\venv\Scripts\python.exe]
x is 50
Changed local x to 2
x is still 50
```

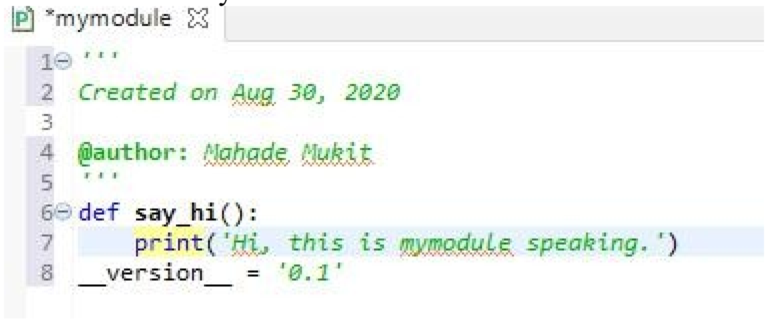
Exercise 4.1.5: Global variable (save as function_global.py)



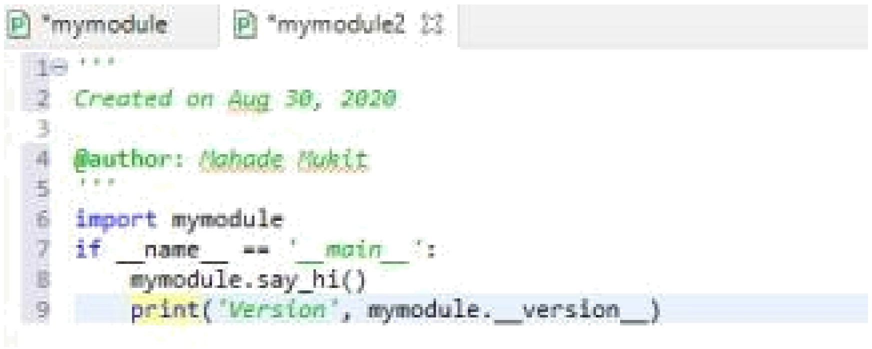
```
'''
Created on Aug 30, 2020

@author: mukit mahade
'''
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)
```

```
<terminated> function_global.py [E:\pythoncode\venv\Scripts\python.exe]
x is 50
Changed global x to 2
Value of x is 2
```

Exercise 4.1.6: Python modules

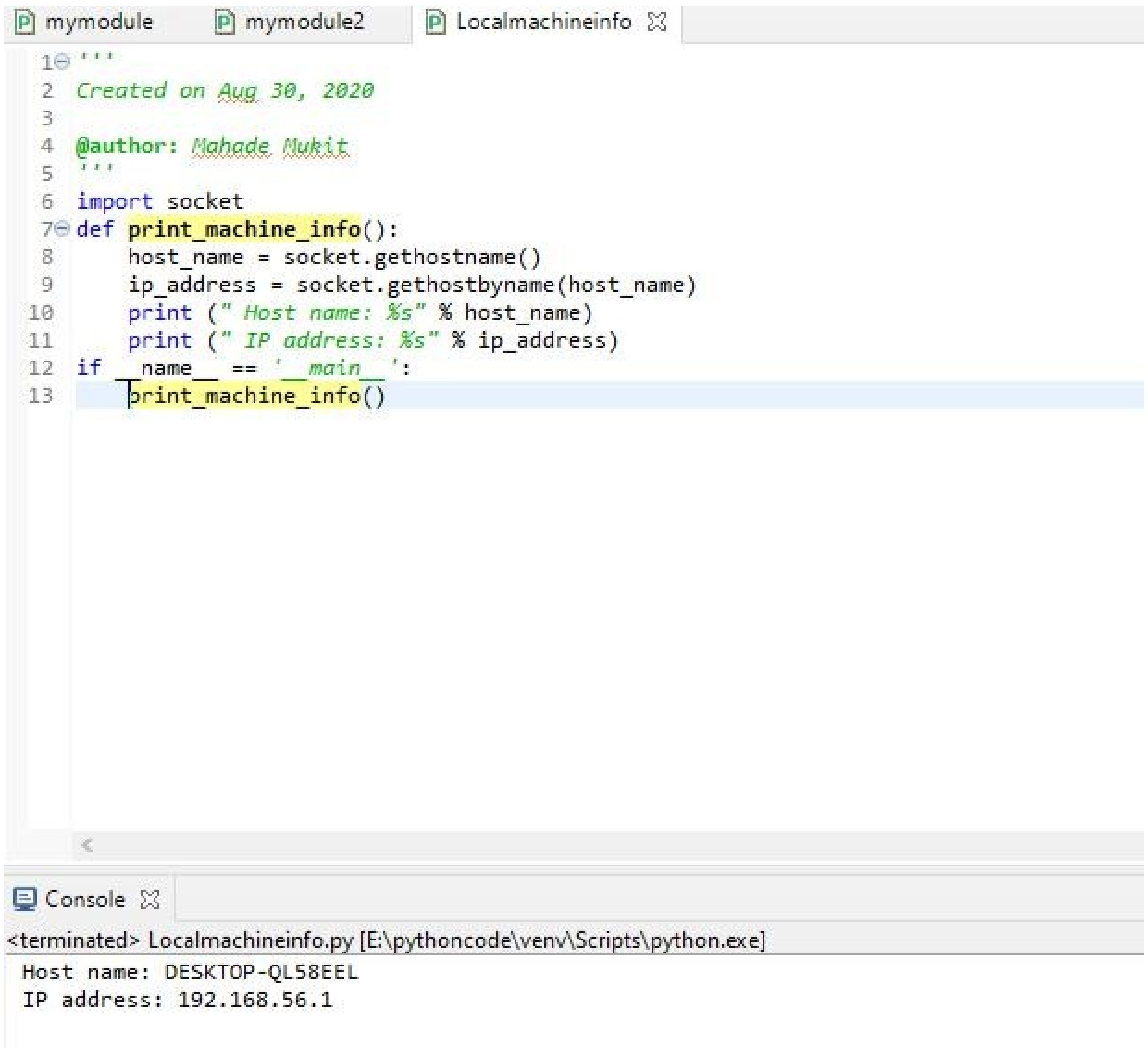


```
'''
Created on Aug 30, 2020

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


*mymodule   *mymodule2

```
'''
Created on Aug 30, 2020

@author: Mahade Mukit
'''
import mymodule
if __name__ == '__main__':
    mymodule.say_hi()
    print('Version', mymodule.__version__)
```

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

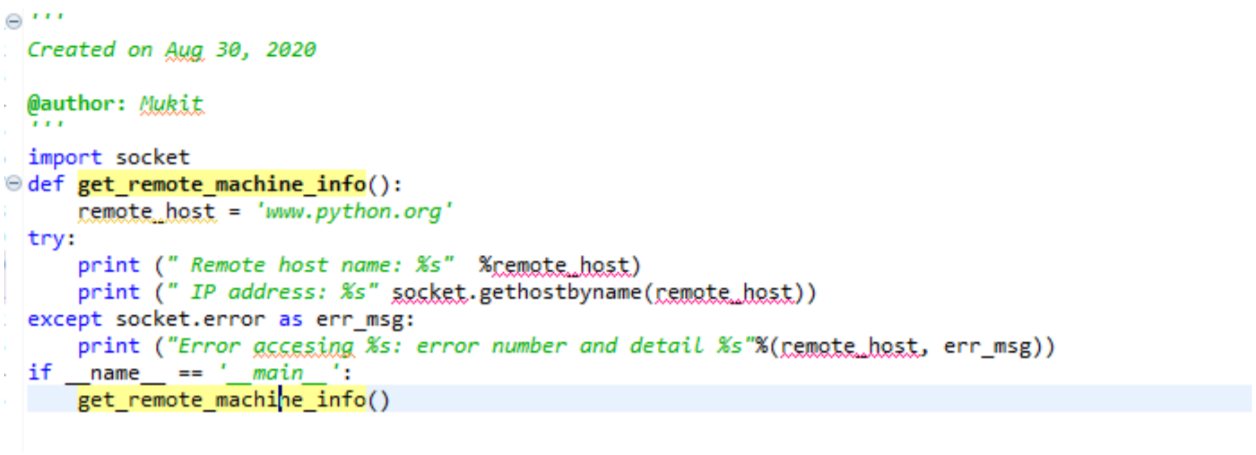

mymodule   mymodule2   Localmachineinfo

```
'''
Created on Aug 30, 2020

@author: Mahade Mukit
'''
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()
```

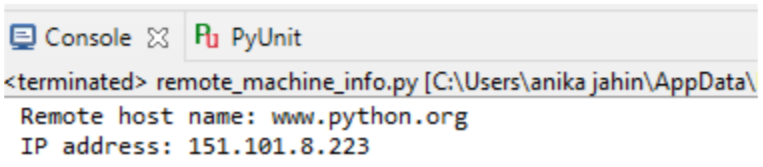
Console

```
<terminated> Localmachineinfo.py [E:\pythoncode\venv\Scripts\python.exe]
 Host name: DESKTOP-QL58EEL
 IP address: 192.168.56.1
```

Exercise 4.2.2: Retrieving a remote machine's IP address



```
'''
Created on Aug 30, 2020

@author: Mukit
'''
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()
```

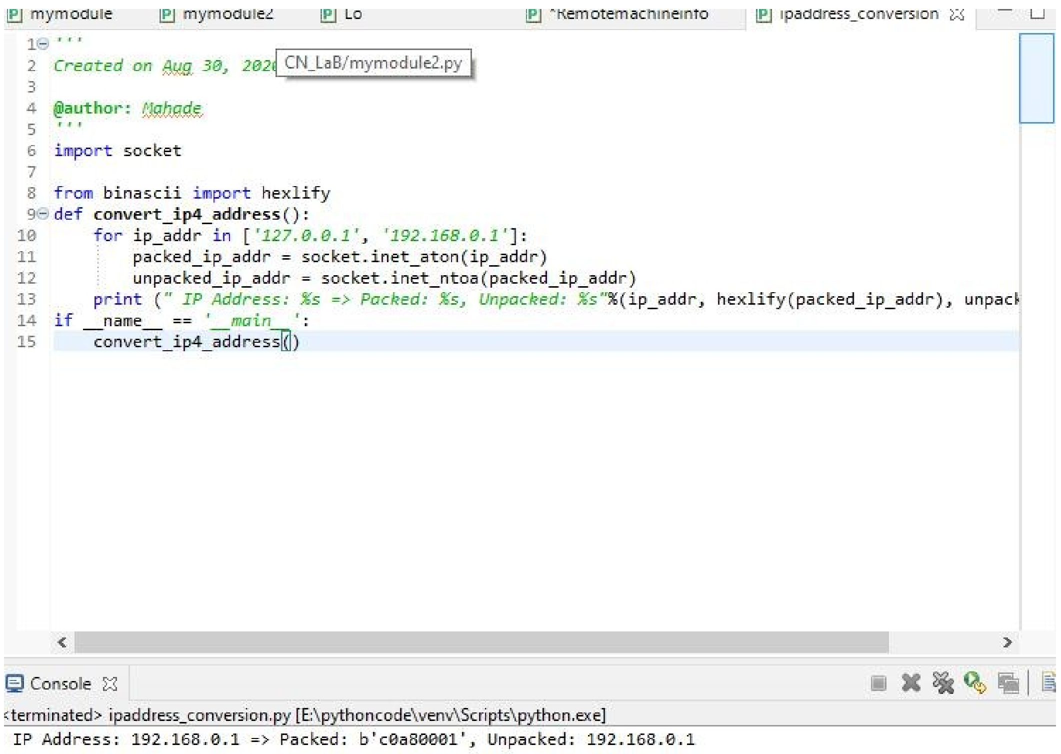


Console ☒ PyUnit

<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



mymodule | mymodule2 | Lo | *Remotemachineinfo | ipaddress_conversion ☒

CN_LaB/mymodule2.py

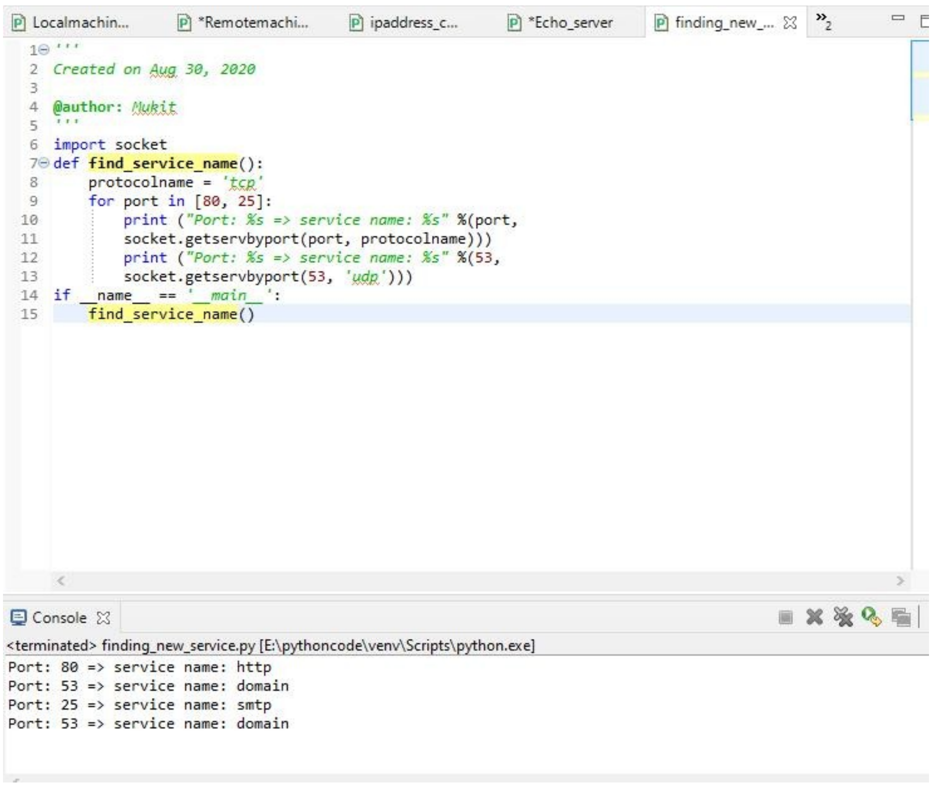
```
'''
Created on Aug 30, 2020

@author: Mahade
'''
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), unpack
if __name__ == '__main__':
    convert_ip4_address()
```

Console ☒

<terminated> ipaddress_conversion.py [E:\pythoncode\venv\Scripts\python.exe]

```
IP Address: 192.168.0.1 => Packed: b'c0a80001', Unpacked: 192.168.0.1
```

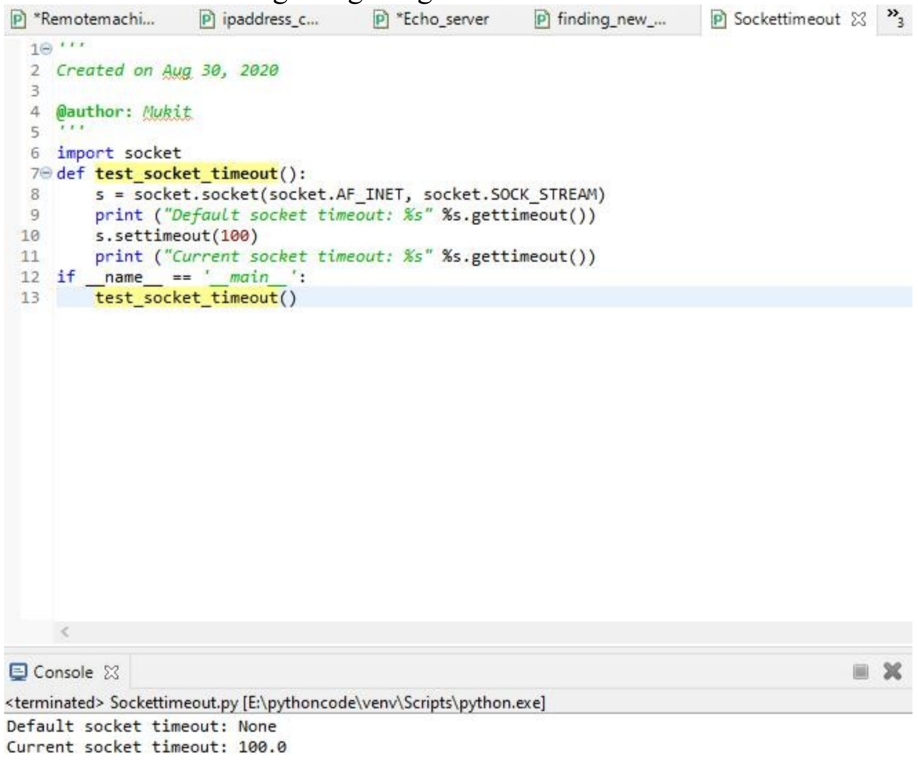
Exercise 4.2.4: Finding a service name, given the port and protocol



```
'''
Created on Aug 30, 2020

@author: Mukit
'''
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_new_service.py [E:\pythoncode\venv\Scripts\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

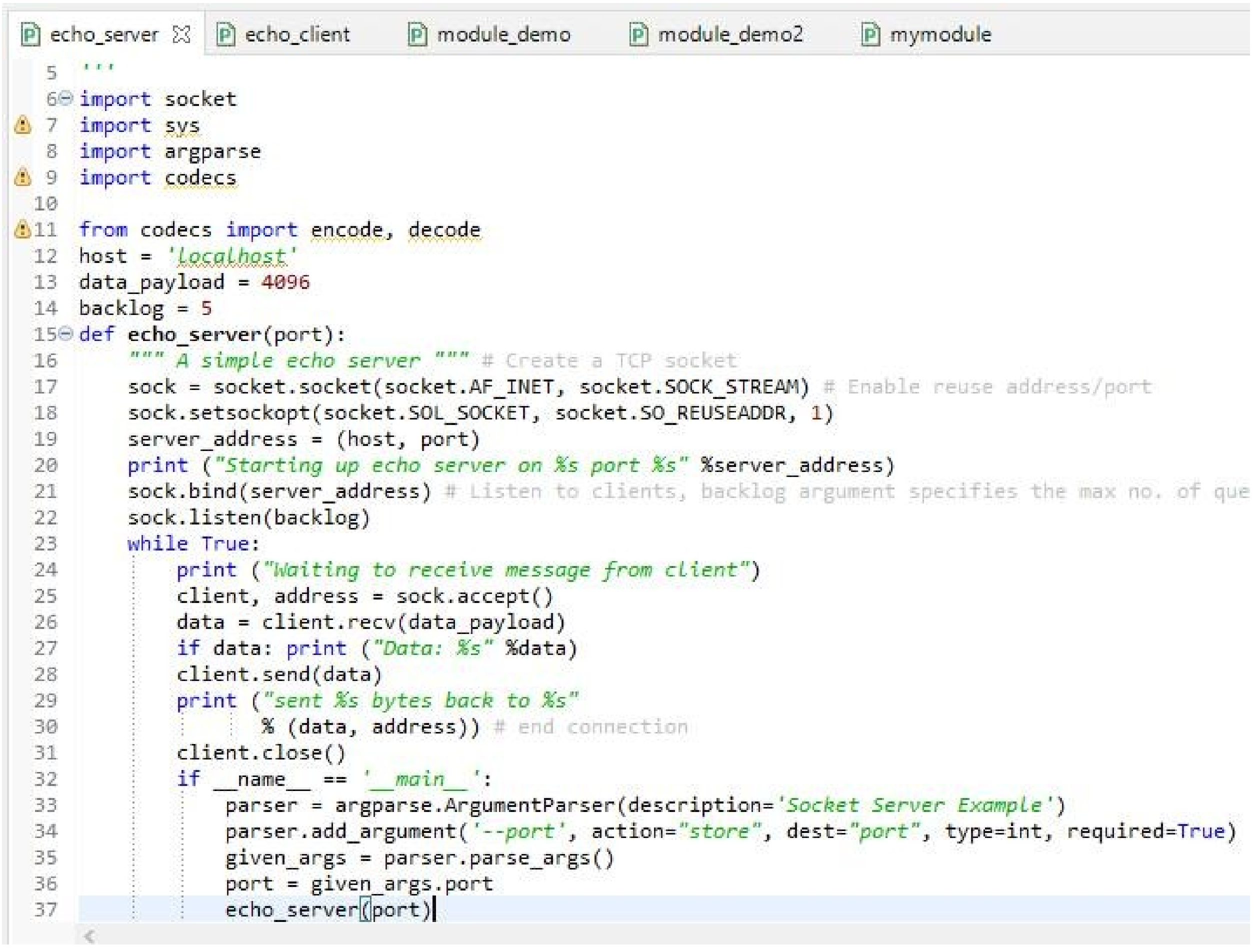


```
'''
Created on Aug 30, 2020

@author: Mukit
'''
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> Sockettimeout.py [E:\pythoncode\venv\Scripts\python.exe]
Default socket timeout: None
Current socket timeout: 100.0
```

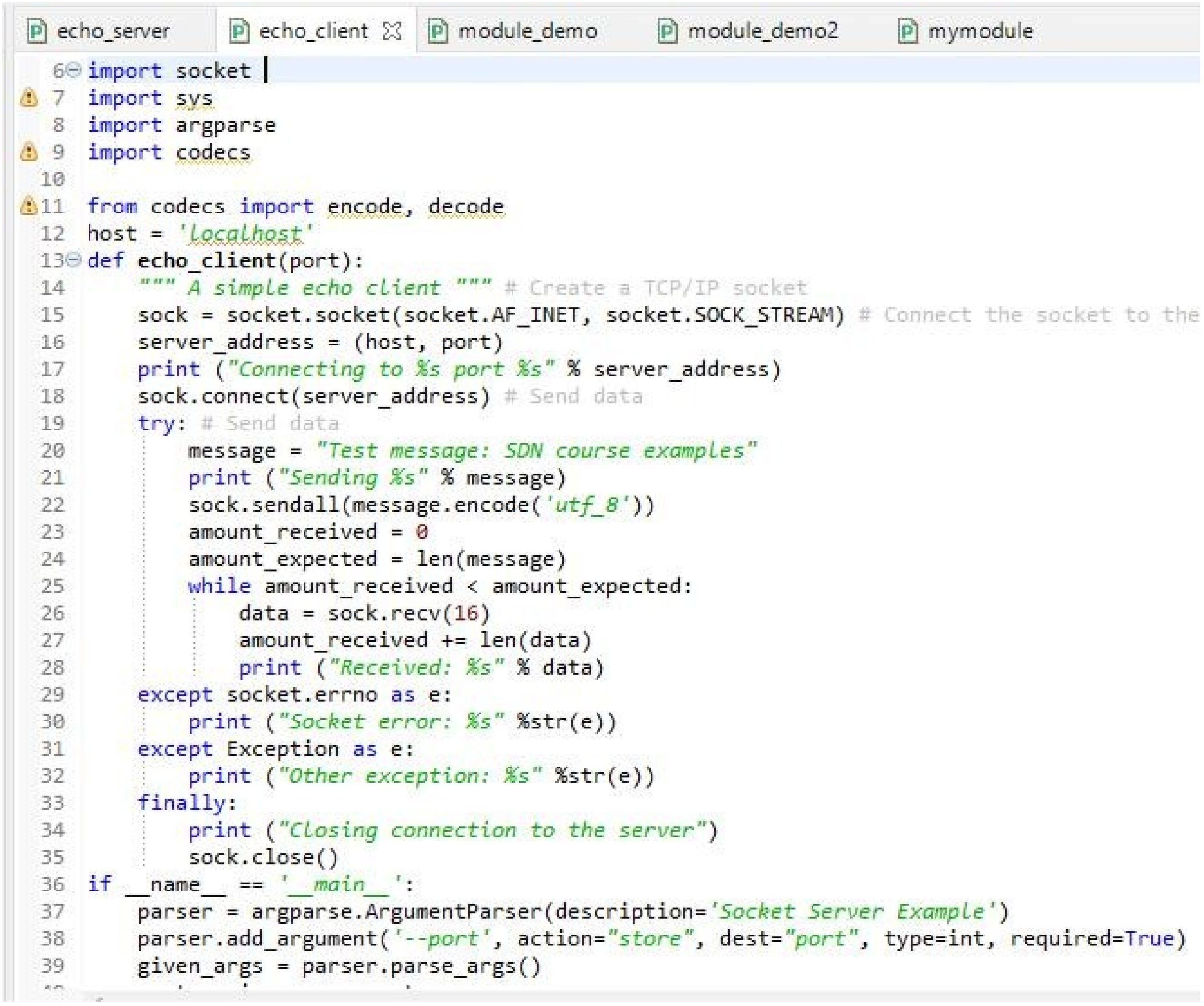
Exercise 4.2.6: Writing a simple echo client/server application (**Tip:** Use port 9900)

Server code:



```
'''
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...