

Lab report No: 05

Lab report Name: Programming with python

Name: Abdul Mukit

ID:IT-17055

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.

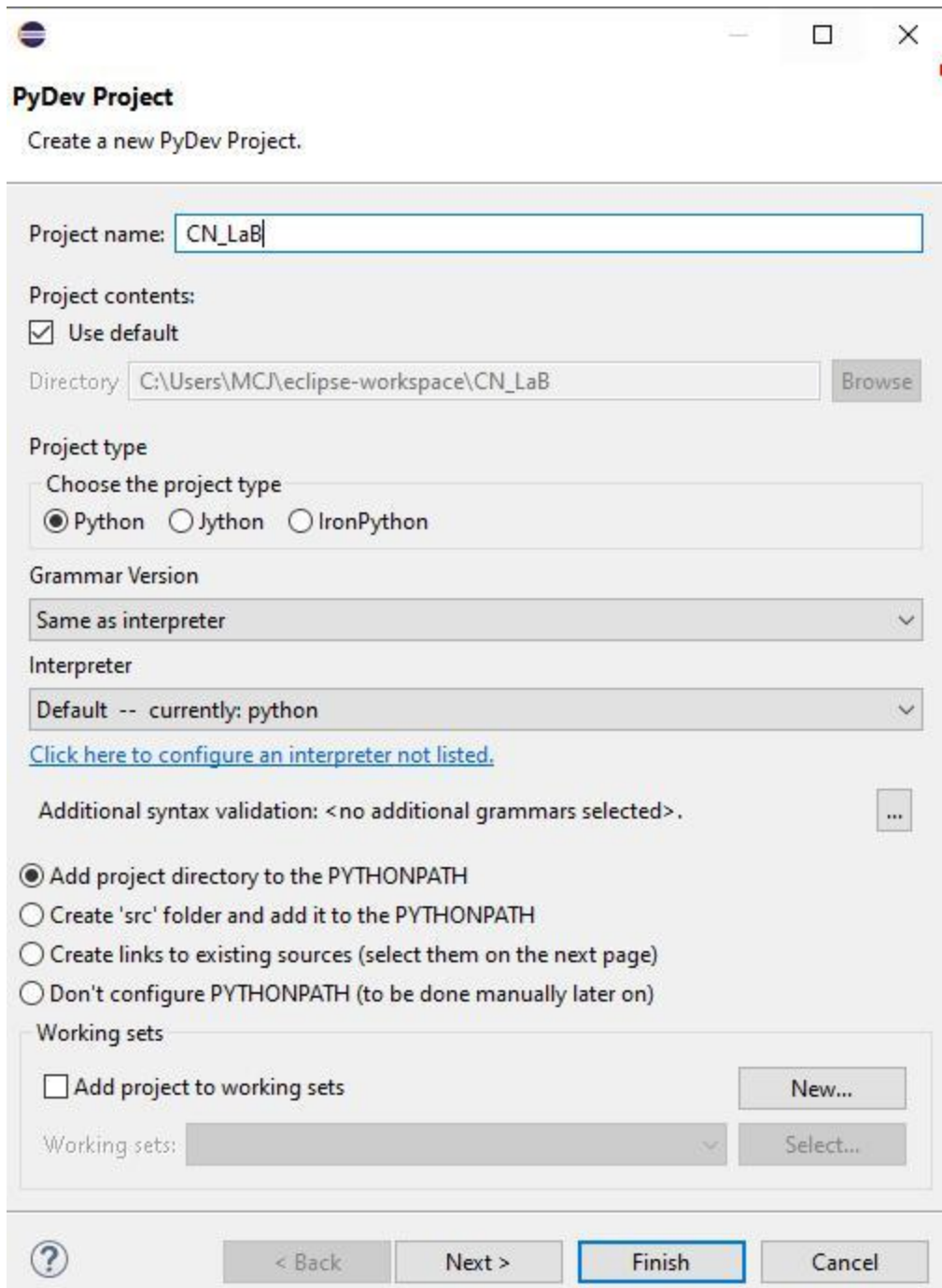
- **TCP:** TCP stands for transmission control protocol. It is implemented in the transport layer of the IP/TCP model and is used to establish reliable connections. TCP is one of the protocols that encapsulate data into packets. It then transfers these to the remote end of the connection using the methods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

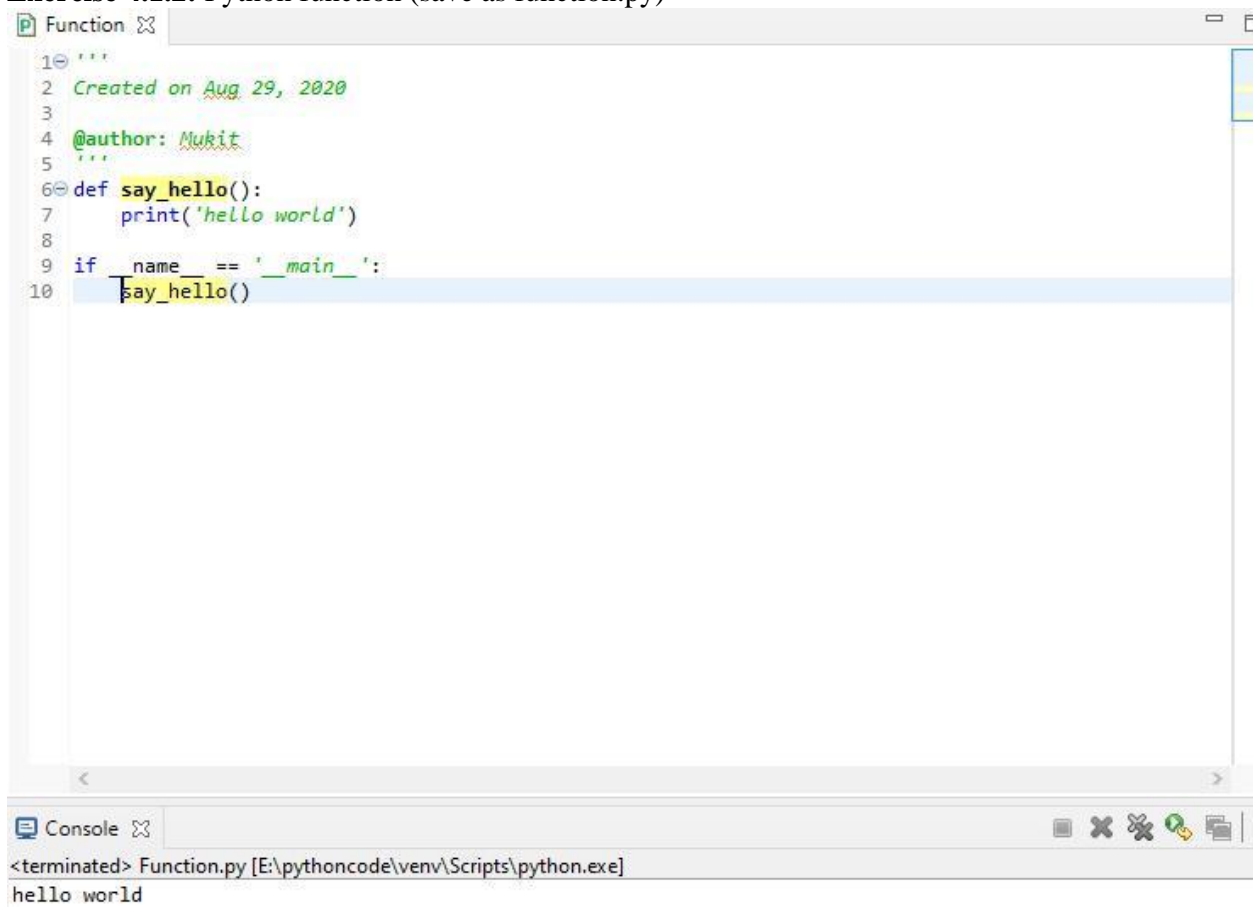


The image shows a 'PyDev Project' dialog box with the following fields and options:

- Project name:** CN_LaB
- Project contents:**
 - ☒ Use default
 - Directory:** C:\Users\MCI\eclipse-workspace\CN_LaB (with a 'Browse' button)
- Project type:** Choose the project type
 - ☒ Python
 - ☐ Jython
 - ☐ IronPython
- Grammar Version:** Same as interpreter (dropdown)
- Interpreter:** Default -- currently: python (dropdown)
 - [Click here to configure an interpreter not listed.](#)
- Additional syntax validation:** <no additional grammars selected>. (with a '...' button)
- PYTHONPATH configuration:**
 - ☒ 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 (with a 'New...' button)
 - Working sets:** (dropdown) (with a 'Select...' button)

At the bottom, there are navigation buttons: '?', '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

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

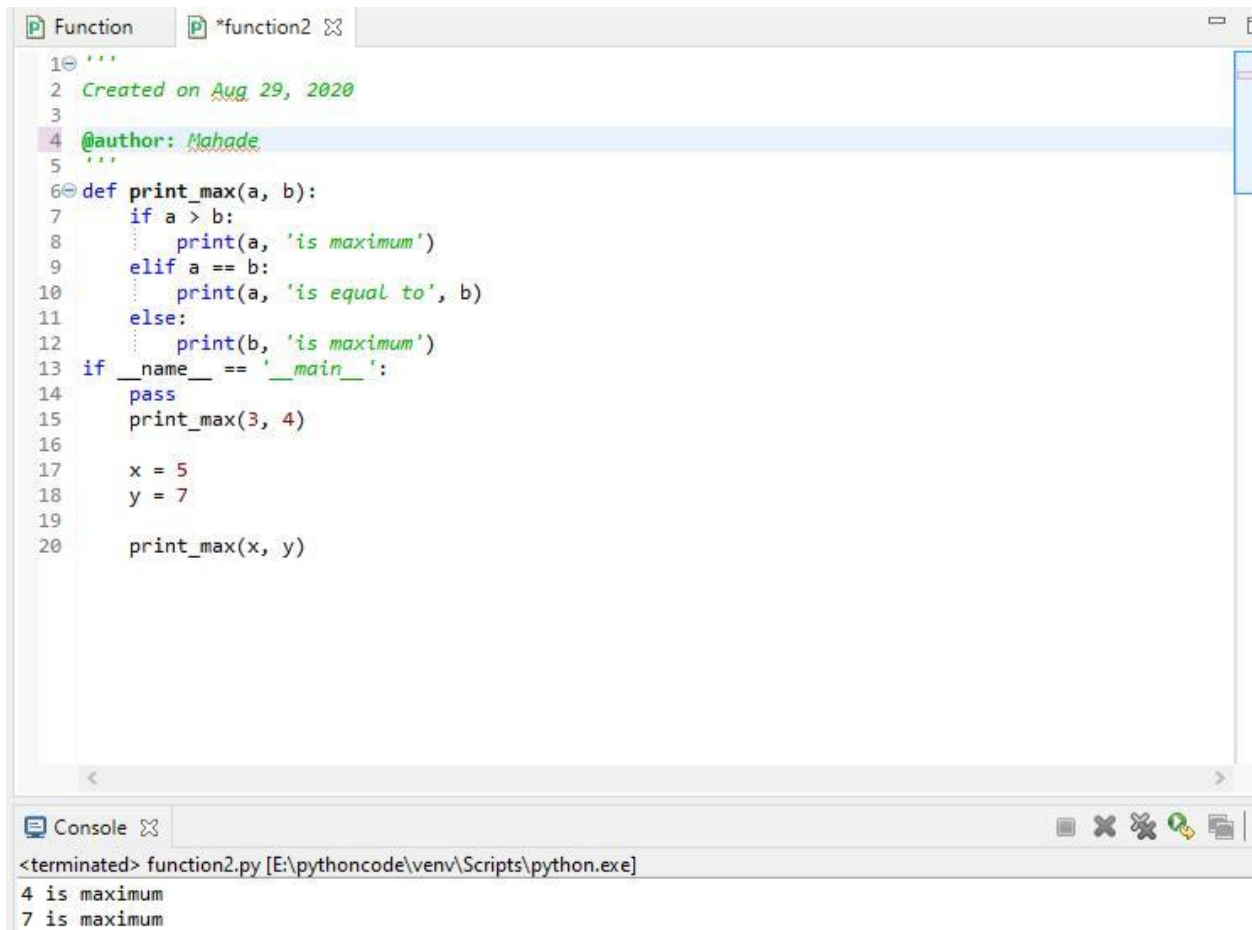


The screenshot shows a Python IDE with two panels. The top panel, titled 'Function', displays a Python script. The script includes a docstring with a creation date and author, a function definition for 'say_hello()', and a main block that calls the function. The bottom panel, titled 'Console', shows the output of the script execution.

```
1 '''  
2 Created on Aug, 29, 2020  
3  
4 @author: Mukit  
5 '''  
6 def say_hello():  
7     print('hello world')  
8  
9 if __name__ == '__main__':  
10     say_hello()
```

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

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



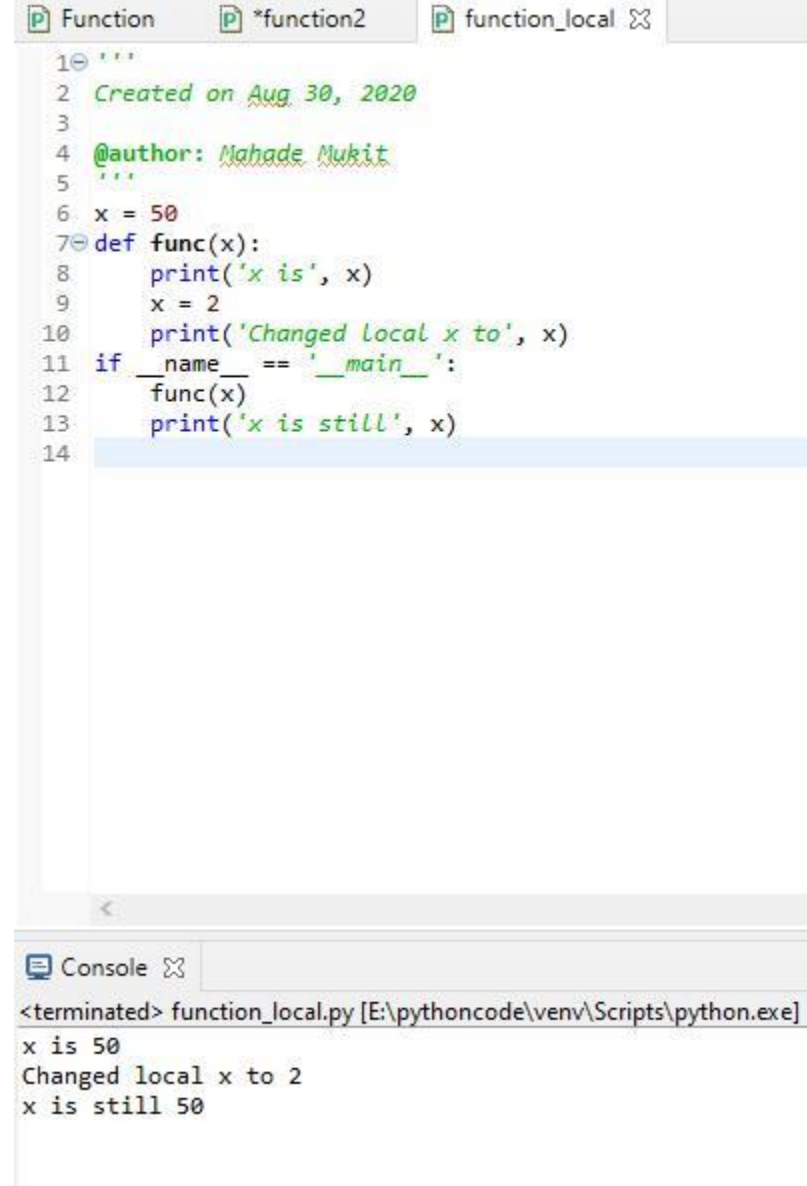
The image shows a Python IDE window with a file named `*function2.py`. The code defines a function `print_max(a, b)` that prints the maximum of two numbers. The function uses an `if-elif-else` structure. Below the function definition, there is a `if __name__ == '__main__':` block that calls `print_max(3, 4)`, sets `x = 5` and `y = 7`, and then calls `print_max(x, y)`. The console output shows the results of these calls: `4 is maximum` and `7 is maximum`.

```
1 """  
2 Created on Aug 29, 2020  
3  
4 @author: Mahade  
5 """  
6 def print_max(a, b):  
7     if a > b:  
8         print(a, 'is maximum')  
9     elif a == b:  
10        print(a, 'is equal to', b)  
11    else:  
12        print(b, 'is maximum')  
13 if __name__ == '__main__':  
14     pass  
15     print_max(3, 4)  
16  
17     x = 5  
18     y = 7  
19  
20     print_max(x, y)
```

Console Output:

```
<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)

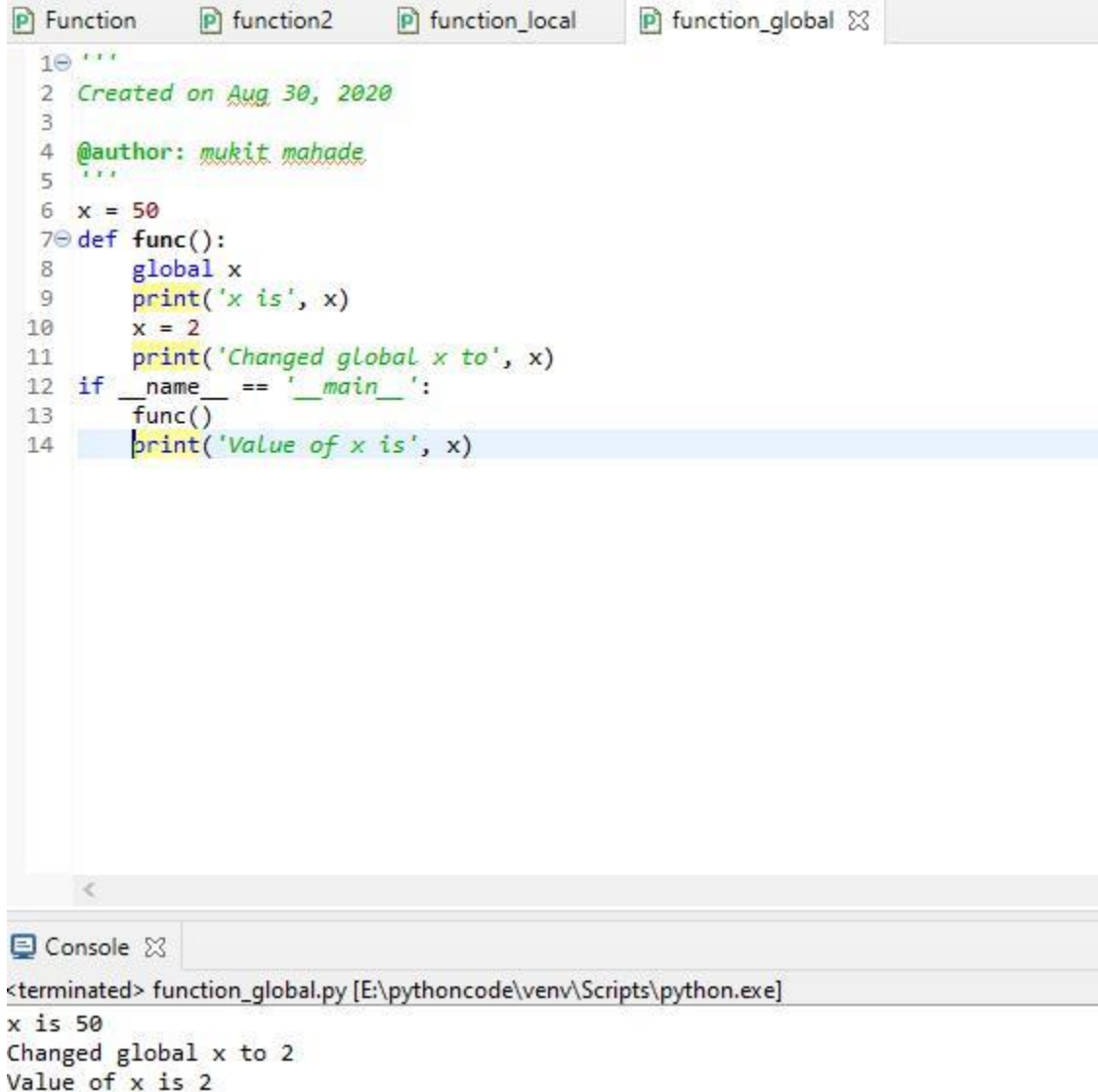


The image shows a Python IDE with a script editor and a console. The script editor has three tabs: 'Function', '*function2', and 'function_local'. The 'function_local' tab is active, showing a Python script. The script starts with a docstring containing the author's name and creation date. It then defines a function 'func(x)' that prints the value of 'x', changes it to 2, and prints it again. Finally, it calls 'func(x)' from the main block and prints the value of 'x' again. The console shows the output of the script, which is 'x is 50', 'Changed local x to 2', and 'x is still 50'.

```
1 '''  
2 Created on Aug 30, 2020  
3  
4 @author: Mahade Mukit  
5 '''  
6 x = 50  
7 def func(x):  
8     print('x is', x)  
9     x = 2  
10    print('Changed local x to', x)  
11 if __name__ == '__main__':  
12     func(x)  
13     print('x is still', x)  
14
```

<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)



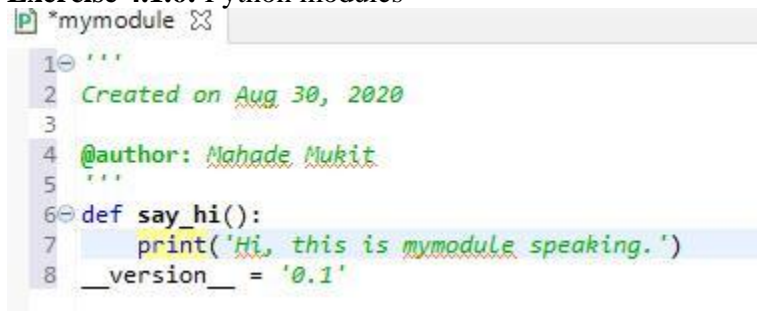
The screenshot shows a Python IDE with a file named `function_global.py` open. The code in the file is as follows:

```
1 '''  
2 Created on Aug 30, 2020  
3  
4 @author: mukit mahade  
5 '''  
6 x = 50  
7 def func():  
8     global x  
9     print('x is', x)  
10    x = 2  
11    print('Changed global x to', x)  
12 if __name__ == '__main__':  
13     func()  
14     print('Value of x is', x)
```

Below the editor, the console window shows the output of the script:

```
<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



The screenshot shows a Python IDE with a file named `*mymodule.py` open. The code in the file is as follows:

```
1 '''  
2 Created on Aug 30, 2020  
3  
4 @author: Mahade Mukit  
5 '''  
6 def say_hi():  
7     print('Hi, this is mymodule speaking.')  
8     __version__ = '0.1'
```

```

P *mymodule  P *mymodule2  ⌵
1  """
2  Created on Aug 30, 2020
3
4  @author: Mahade Mukit
5  """
6  import mymodule
7  if __name__ == '__main__':
8      mymodule.say_hi()
9      print('Version', mymodule.__version__)

```

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

```

P mymodule  P mymodule2  P Localmachineinfo  ⌵
1  """
2  Created on Aug 30, 2020
3
4  @author: Mahade Mukit
5  """
6  import socket
7  def print_machine_info():
8      host_name = socket.gethostname()
9      ip_address = socket.gethostbyname(host_name)
10     print (" Host name: %s" % host_name)
11     print (" IP address: %s" % ip_address)
12 if __name__ == '__main__':
13     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 *Remotemachininfo ipaddress_conversion

```
1'''
2Created on Aug 30, 2020 CN_LaB/mymodule2.py
3
4@author: Mahade
5'''
6import socket
7
8from binascii import hexlify
9def convert_ip4_address():
10    for ip_addr in ['127.0.0.1', '192.168.0.1']:
11        packed_ip_addr = socket.inet_aton(ip_addr)
12        unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
13        print (" IP Address: %s => Packed: %s, Unpacked: %s"%(ip_addr, hexlify(packed_ip_addr), unpacked_ip_addr))
14if __name__ == '__main__':
15    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

```
Localmachin... *Remotemachi... ipaddress_c... *Echo_server finding_new_... »2
1 '''
2 Created on Aug 30, 2020
3
4 @author: Mukit
5 '''
6 import socket
7 def find_service_name():
8     protocolname = 'tcp'
9     for port in [80, 25]:
10         print ("Port: %s => service name: %s" %(port,
11             socket.getservbyport(port, protocolname)))
12         print ("Port: %s => service name: %s" %(53,
13             socket.getservbyport(53, 'udp')))
14 if __name__ == '__main__':
15     find_service_name()
```

Console »

```
<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

```
*Remotemachi... ipaddress_c... *Echo_server finding_new_... Sockettimeout »3
1 '''
2 Created on Aug 30, 2020
3
4 @author: Mukit
5 '''
6 import socket
7 def test_socket_timeout():
8     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9     print ("Default socket timeout: %s" %s.gettimeout())
10    s.settimeout(100)
11    print ("Current socket timeout: %s" %s.gettimeout())
12 if __name__ == '__main__':
13     test_socket_timeout()
```

Console »

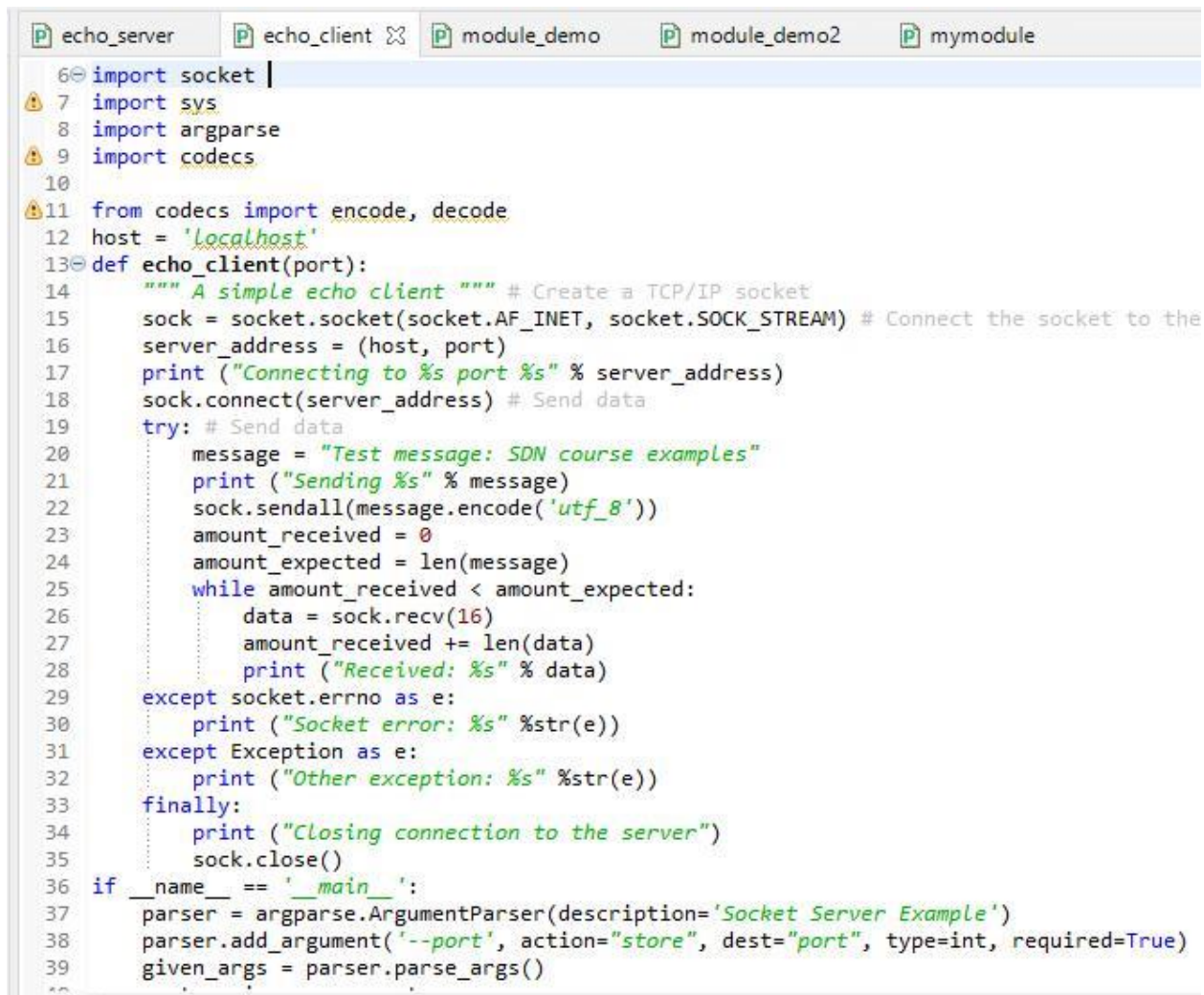
```
<terminated> Sockettimeout.py [E:\pythoncode\venv\Scripts\python.exe]
Default socket timeout: None
Current socket timeout: 100.0
```

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

Server code:

```
echo_server echo_client module_demo module_demo2 mymodule
5 """
6 import socket
7 import sys
8 import argparse
9 import codecs
10
11 from codecs import encode, decode
12 host = 'localhost'
13 data_payload = 4096
14 backlog = 5
15 def echo_server(port):
16     """ A simple echo server """ # Create a TCP socket
17     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Enable reuse address/port
18     sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
19     server_address = (host, port)
20     print ("Starting up echo server on %s port %s" %server_address)
21     sock.bind(server_address) # Listen to clients, backlog argument specifies the max no. of que
22     sock.listen(backlog)
23     while True:
24         print ("Waiting to receive message from client")
25         client, address = sock.accept()
26         data = client.recv(data_payload)
27         if data: print ("Data: %s" %data)
28         client.send(data)
29         print ("sent %s bytes back to %s"
30               % (data, address)) # end connection
31         client.close()
32     if __name__ == '__main__':
33         parser = argparse.ArgumentParser(description='Socket Server Example')
34         parser.add_argument('--port', action="store", dest="port", type=int, required=True)
35         given_args = parser.parse_args()
36         port = given_args.port
37         echo_server(port)
```

Client code:



```
6 import socket |
7 import sys
8 import argparse
9 import codecs
10
11 from codecs import encode, decode
12 host = 'localhost'
13 def echo_client(port):
14     """ A simple echo client """ # Create a TCP/IP socket
15     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Connect the socket to the
16     server_address = (host, port)
17     print ("Connecting to %s port %s" % server_address)
18     sock.connect(server_address) # Send data
19     try: # Send data
20         message = "Test message: SDN course examples"
21         print ("Sending %s" % message)
22         sock.sendall(message.encode('utf_8'))
23         amount_received = 0
24         amount_expected = len(message)
25         while amount_received < amount_expected:
26             data = sock.recv(16)
27             amount_received += len(data)
28             print ("Received: %s" % data)
29     except socket.errno as e:
30         print ("Socket error: %s" %str(e))
31     except Exception as e:
32         print ("Other exception: %s" %str(e))
33     finally:
34         print ("Closing connection to the server")
35         sock.close()
36 if __name__ == '__main__':
37     parser = argparse.ArgumentParser(description='Socket Server Example')
38     parser.add_argument('--port', action="store", dest="port", type=int, required=True)
39     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...