

Lab Report: 02

Report Name: Programming with Python

Course code: ICT-3208

Course title: Computer Networks Lab

Date of

Performance: 15 /01/2021

Date of

Submission: / /2021

SUBMITTED BY

Name: Shakhera khanom

ID: IT-18033

3rd year 2nd semester

Session: 2017-18

Department of ICT,

MBSTU.

SUBMITTED TO

Nazrul Islam

Assistant Professor

Department of ICT,

MBSTU.

Experiment No: 02

Experiment Name: Programming with Python

Objectives:

- Understand how python function works
- Understand the use of global and local variables
- Understand how python modules works
- Learning the basis of networking 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.

Methodology:

Defining functions: Functions are defined using the def keyword.

```
def xx_yy(variable1, variable2):
```

Defining local and global variables: Local and global variables can be defined using:

```
x = 50 #local  
global x
```

Defining modules: There are various methods of writing modules, but the simplest way is to create a file with a .py extension that contains functions and variables.

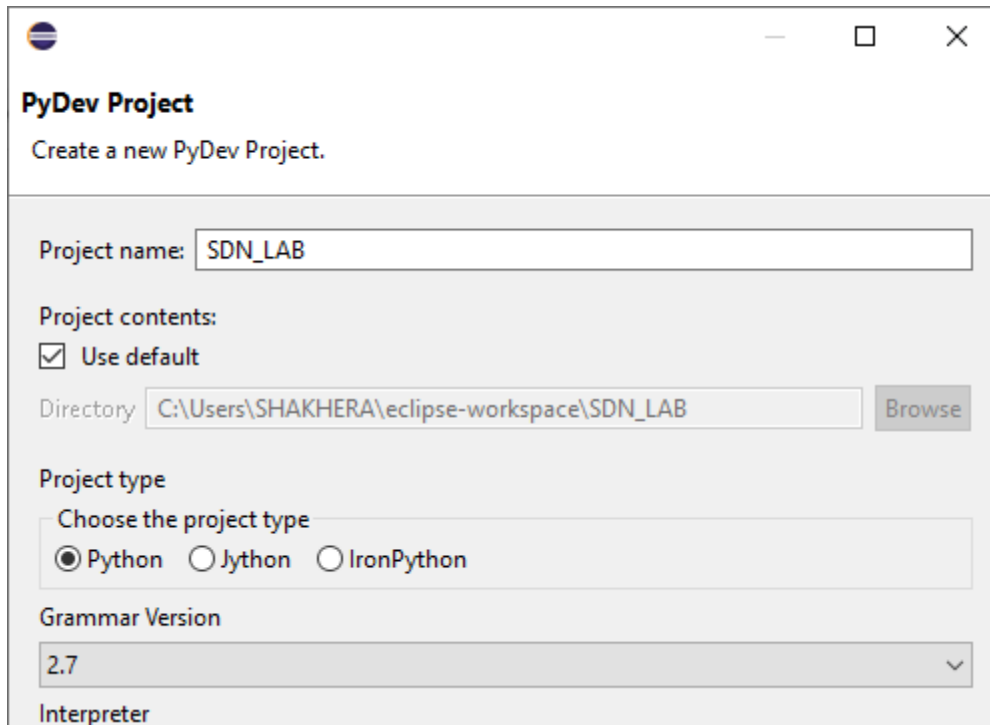
```
def xx_yy():  
    aa
```

Using modules: A module can be imported by another program to make use of its functionality.

```
import xx_yy
```

Exercises:

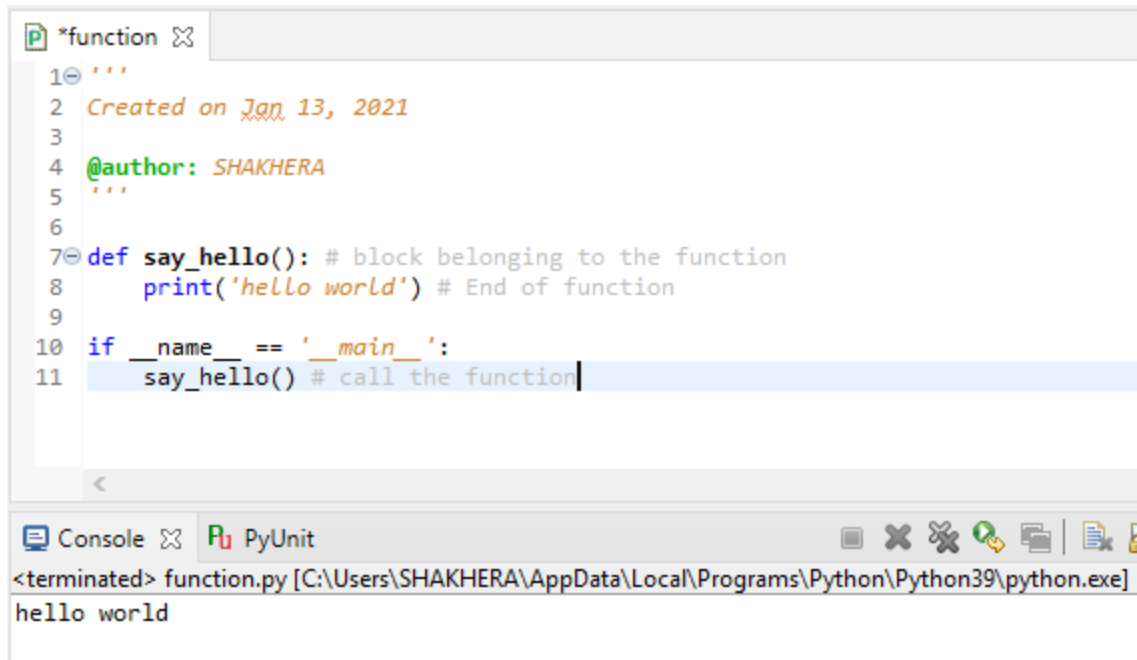
Exercise 4.1.1: Create a python project using with SDN_LAB



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

- Project name:** SDN_LAB
- Project contents:**
 - ☒ Use default
 - Directory:** C:\Users\SHAKHERA\eclipse-workspace\SDN_LAB (with a 'Browse' button)
- Project type:**
 - Choose the project type
 - ☒ Python ☐ Jython ☐ IronPython
- Grammar Version:** 2.7 (dropdown menu)
- Interpreter:** (label, no input field visible)

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



```
*function
1 '''
2 Created on Jan 13, 2021
3
4 @author: SHAKHERA
5 '''
6
7 def say_hello(): # block belonging to the function
8     print('hello world') # End of function
9
10 if __name__ == '__main__':
11     say_hello() # call the function
```

Console PyUnit

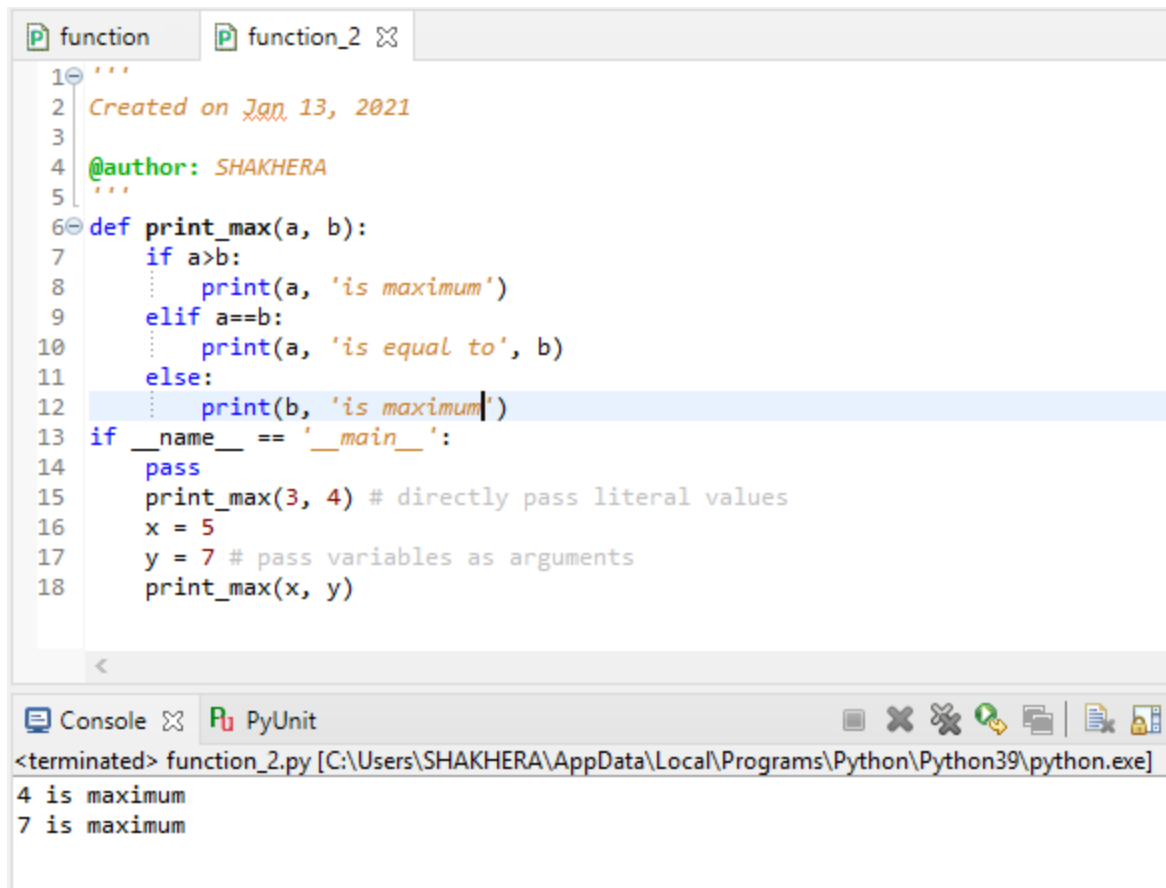
<terminated> function.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
hello world

Does the function need any parameter?

Answer:

This function need not any parameter.

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



The screenshot shows a Python IDE with two tabs: 'function' and 'function_2'. The 'function' tab is active, displaying a Python script. The script defines a function `print_max(a, b)` that compares two numbers and prints the maximum. It also includes a main block that calls the function with literal values (3, 4) and variables (x=5, y=7). The console output shows the results of these calls.

```
1 """
2 Created on Jan 13, 2021
3
4 @author: SHAKHERA
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) # directly pass literal values
16     x = 5
17     y = 7 # pass variables as arguments
18     print_max(x, y)
```

Console output:


```
<terminated> function_2.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
4 is maximum
7 is maximum
```

Does the function need any parameter?

Answer:

Yes, this function need parameter.

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



```
1 '''  
2 Created on Jan 13, 2021  
3  
4 @author: SHAKHERA  
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)
```

Console

```
<terminated> function_local.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]  
x is 50  
changed Local x to 2  
x is still 50
```

Which is the final value of variable x? Why variable x does not change to 2?

Answer:

The final value of variable x is 50.

Variables that are defined inside a function body is called local variable, and can only be used inside the function.

In the above program, we define x as a local variable. That's why inside the func() function x is print by 2 but outside the function body x print by 50.

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

```
function_local  function_global ✕
1  """
2  Created on Jan 13, 2021
3
4  @author: SHAKHERA
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)

<
Console ✕
<terminated> function_global.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
x is 50
changed Global x to 2
Value of x is 2
```

Which is the final value of variable x? Why variable x change this time?

Answer:

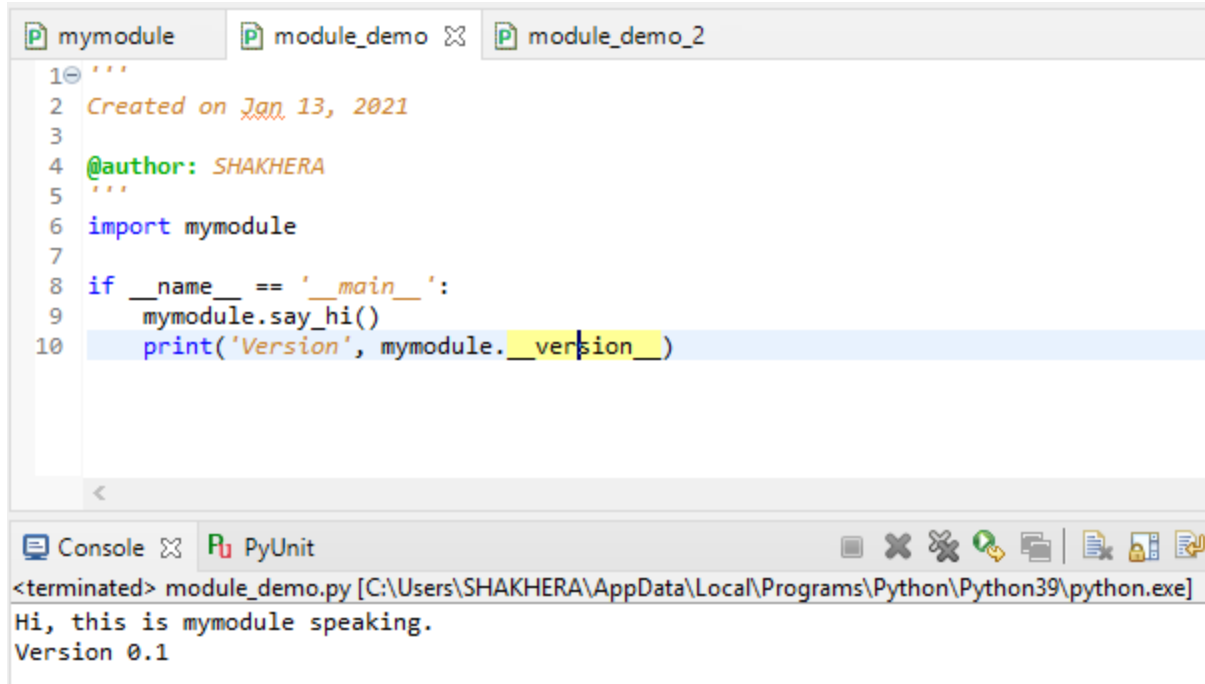
The final value of variable x is 2.

In the above program, we define x as a global keyword inside the func() function. When, the value of x is changed by 2, i.e. x = 2. After that, we call the func() function. Finally, we print the global variable x.

As we can see, change also occurred on the global outside the function, x=2.

Exercise 4.1.6: Python modules

```
mymodule ✕  module_demo  *module_demo_2
1  """
2  Created on Jan 13, 2021
3
4  @author: SHAKHERA
5  """
6  def say_hi():
7      print('Hi, this is mymodule speaking. ')
8      __version__ = '0.1'
9      import mymodule
10 if __name__ == '__main__':
11     mymodule.say_hi()
12     print('Version', mymodule.__version__)
```



The screenshot shows a Python IDE with three tabs: 'mymodule', 'module_demo', and 'module_demo_2'. The 'module_demo' tab is active, displaying the following code:

```
1 '''
2 Created on Jan 13, 2021
3
4 @author: SHAKHERA
5 '''
6 import mymodule
7
8 if __name__ == '__main__':
9     mymodule.say_hi()
10    print('Version', mymodule.__version__)
```

Below the code editor, the 'Console' tab is active, showing the output of the execution:

```
<terminated> module_demo.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
Hi, this is mymodule speaking.
Version 0.1
```

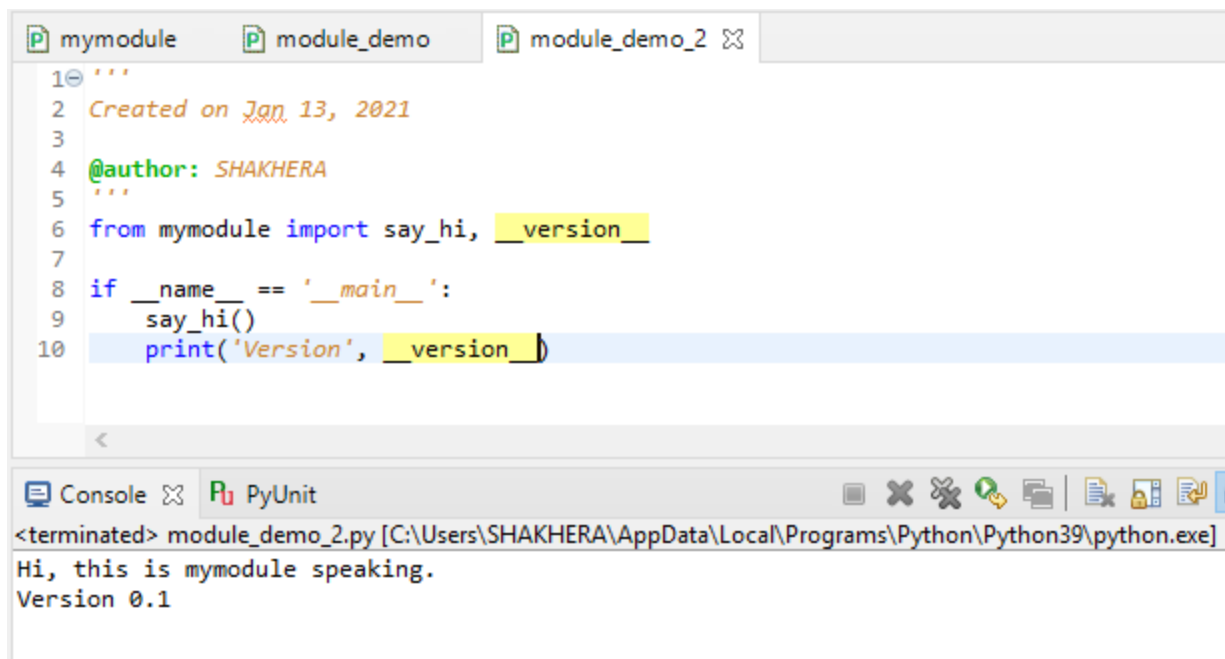
Which is the role of import?

Answer:

Python code in one module gains access to the code in another module by the process of importing it. The import statement is the most common way of invoking the import machinery, but it is not the only way.

Import is a keyword. Import keyword is used to import built-in and user-defined packages into your.

- The import keyword finds and loads a package, a sub-package or a module if they are found in the system using the import mechanism.
- Once a module is loaded, the name of the imported module is introduced in the current scope where the import statement is executed.
- Which name gets introduced - name of the actual module or the top-level module...depends on how the import statement is used.



The screenshot shows a Python IDE with three tabs: 'mymodule', 'module_demo', and 'module_demo_2'. The 'module_demo_2' tab is active, displaying a Python script. The script includes a docstring with the author 'SHAKHERA' and a version '0.1'. It imports 'say_hi' and 'version' from 'mymodule'. A main block is defined with an if statement that checks if the script is being run directly, and if so, it calls 'say_hi()' and prints the version. The console at the bottom shows the output of the script: 'Hi, this is mymodule speaking.' and 'Version 0.1'.

```
1 """  
2 Created on Jan 13, 2021  
3  
4 @author: SHAKHERA  
5 """  
6 from mymodule import say_hi, version  
7  
8 if __name__ == '__main__':  
9     say_hi()  
10    print('Version', version)
```

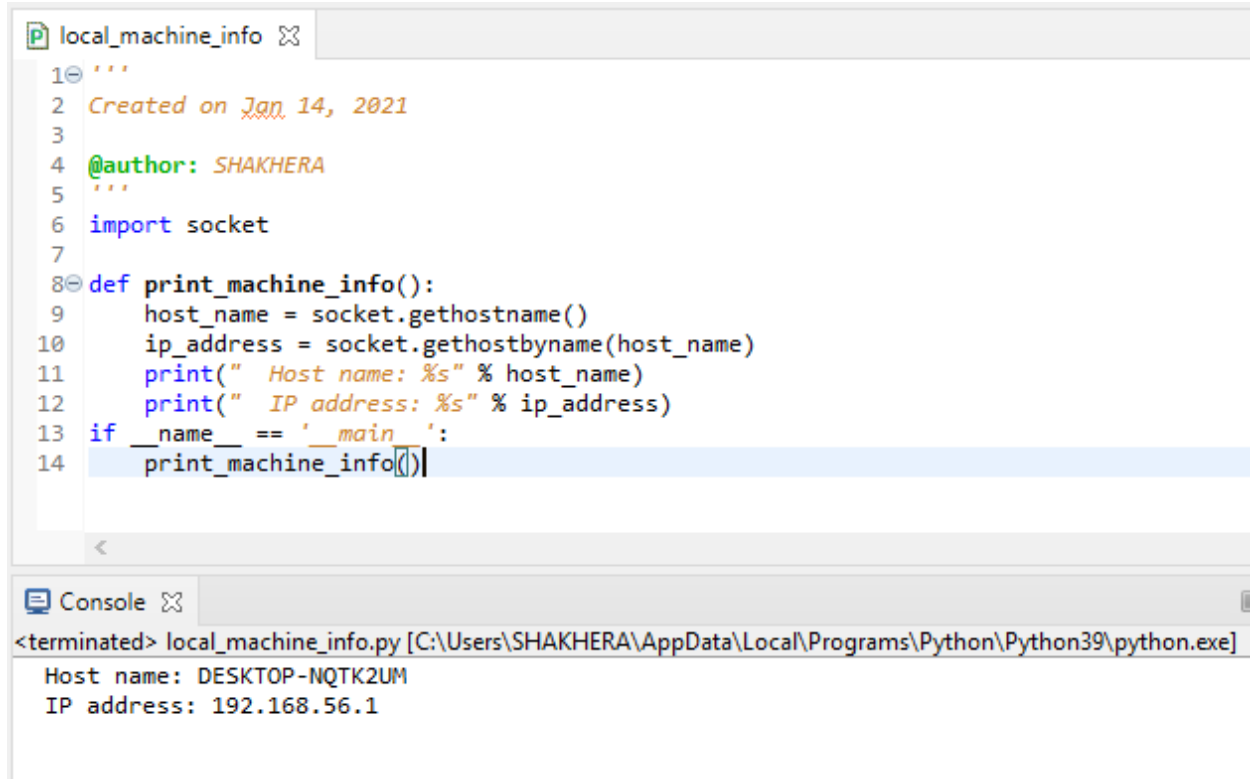
Console
PyUnit
<terminated> module_demo_2.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
Hi, this is mymodule speaking.
Version 0.1

Which is the role of from, import?

Answer:

- While the import command is very straightforward it lacks efficiency. In many scenarios you may need only a small portion of the module. To solve this the **from** keyword is born.
- When a Python module is imported using the keywords import and from, the given module is imported and the given name of the module is bound to the current scope regardless of whether it is a top level module or not.

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



The screenshot shows a Python IDE with a file named `local_machine_info.py`. The code is as follows:

```
1 '''  
2 Created on Jan 14, 2021  
3  
4 @author: SHAKHERA  
5 '''  
6 import socket  
7  
8 def print_machine_info():  
9     host_name = socket.gethostname()  
10    ip_address = socket.gethostbyname(host_name)  
11    print(" Host name: %s" % host_name)  
12    print(" IP address: %s" % ip_address)  
13 if __name__ == '__main__':  
14    print_machine_info()
```

Below the code editor is a console window showing the output of the script:

```
<terminated> local_machine_info.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]  
Host name: DESKTOP-NQTK2UM  
IP address: 192.168.56.1
```

Which module the program uses? Provide two additional functions of socket?

Answer:

There are many ways to find hostname and IP address of a local machine. Here is a simple method to find hostname and IP address using python code.

Socket module is used in the above program.

Functions used:

- **gethostname():** The gethostname function retrieves the standard host name for the local computer
- **gethostbyname():** The gethostbyname function retrieves host information corresponding to a host name from a host database.

Exercise 4.2.2: Retrieving a remote machine's IP address

```
remote_machine_info
1 '''
2 Created on Jan 14, 2021
3
4 @author: SHAKHERA
5 '''
6 import socket
7
8 def get_remote_machine_info():
9     remote_host = 'www.python.org'
10     try:
11         print("    Remote host name: %s" % remote_host)
12         print("    IP address: %s" % socket.gethostbyname(remote_host))
13     except socket.error as err_msg:
14         print("Error accessing %s: error number and detail %s" %(remote_host, err_msg))
15 if __name__ == '__main__':
16     get_remote_machine_info()
```

```
Console
<terminated> remote_machine_info.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
Remote host name: www.python.org
IP address: 151.101.8.223
```

Modify the code for getting the RMIT website info.

Answer:

```
modify_remote_machine_info ❏
1  """
2  Created on Jan 15, 2021
3
4  @author: SHAKHERA
5  """
6  import socket
7
8  def get_rmit_machine_info():
9      remote_host = 'www.rmit.edu.au'
10     try:
11         print("    RMIT host name: %s" % remote_host)
12         print("    IP address: %s" % socket.gethostbyname(remote_host))
13     except socket.error as err_msg:
14         print("Error accessing %s: error number and detail %s" %(remote_host, err_msg))
15 if __name__ == '__main__':
16     get_rmit_machine_info()

<
Console ❏
<terminated> modify_remote_machine_info.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
    RMIT host name: www.rmit.edu.au
    IP address: 54.66.185.131
```

Exercise 4.2.3: Converting an IPv4 address to different formats

```
*ip4_address_conversion

1 '''
2 Created on Jan 14, 2021
3
4 @author: SHAKHERA
5 '''
6 import socket
7 from binascii import hexlify
8
9
10 def convert_ip4_address():
11     for ip_addr in ['127.0.0.1', '192.168.0.1']:
12         packed_ip_addr = socket.inet_aton(ip_addr)
13         unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
14         print("    IP Address: %s => Packed: %s, Unpacked: %s"
15 % (ip_addr, hexlify(packed_ip_addr), unpacked_ip_addr))
16
17 if __name__ == '__main__':
18     convert_ip4_address()
```

```
<terminated> ip4_address_conversion.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\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
```

How binascii works?

Answer:

binascii means convert between binary and ASCII.

The binascii module contains a number of methods to convert between binary and various ASCII-encoded binary representations.

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

```
ip4_address_conversion  finding_server_name  X
1  """
2  Created on Jan 14, 2021
3
4  @author: SHAKHERA
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
15 if __name__ == '__main__':
16     find_service_name()
<
Console  X
<terminated> finding_server_name.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
Port: 80 => service name: http
Port: 53 => service name: domain
Port: 25 => service name: smtp
Port: 53 => service name: domain
```

Modify the code for getting complete the table:

Port	Protocol
21	
22	
110	

Answer:

```
finding_server_name  modified_finding_server_name
1 '''
2 Created on Jan 14, 2021
3
4 @author: SHAKHERA
5 '''
6 import socket
7 def find_protocol_name():
8     protocolname = 'tcp'
9     for port in [21, 22, 110]:
10         print ("Port: %s => Protocol name: %s" %(port,
11             socket.getservbyport(port, protocolname)))
12
13 if __name__ == '__main__':
14     find_protocol_name()
```

```
Console
<terminated> modified_finding_server_name.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
Port: 21 => Protocol name: ftp
Port: 22 => Protocol name: ssh
Port: 110 => Protocol name: pop3
```

Exercise 4.2.5: Setting and getting the default socket timeout

```
socket_timeout
1 '''
2 Created on Jan 14, 2021
3
4 @author: SHAKHERA
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
13 if __name__ == '__main__':
14     test_socket_timeout()
```

```
Console
<terminated> socket_timeout.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
Default socket timeout: None
Current socket timeout: 100.0
```

Which is the role of socket timeout in real applications?

Answer:

Use `settimeout()` to change the **timeout** of a socket to a floating point value representing the number of seconds to block before deciding the socket is not ready for the operation, When the timeout expires, a timeout exception is raised.

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

Server code:

```
*echo_server  echo_client
1  """
2  Created on Jan 14, 2021
3
4  @author: SHAKHERA
5  """
6  import socket
7  import sys
8  import argparse
9  import codecs
10 from codecs import encode, decode
11 host = 'localhost'
12 data_payload = 4096
13 backlog = 5
14 def echo_server(port):
15     """A simple echo server """ # Create a TCP socket
16     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17     # Enable reuse address/port
18     sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # Bind the socket to the po
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
22     # the max no. of queued connections
23     sock.listen(backlog)
24     while True:
25         print("Waiting to receive message from client")
26         client, address = sock.accept()
27         data = client.recv(data_payload)
28         if data:
29             print("Data: %s" %data)
30             client.send(data)
31             print("sent %s bytes back to %s" %(data, address)) # end connection
32             client.close()
33 if __name__ == '__main__':
34     parser = argparse.ArgumentParser(description='Socket Server Example')
35     parser.add_argument('--port', action='store', dest="port", type=int, required=True)
36     given_args = parser.parse_args()
37     port = given_args.port
38     echo_server(port)
39
```

Client code:


```
*echo_server  *echo_client  x
2 Created on Jan 14, 2021
3
4 @author: SHAKHERA
5
6 import socket
7 import sys
8 import argparse
9 import codecs
10 from codecs import encode, decode
11 host = 'localhost'
12 def echo_client(port):
13     """A simple echo client """ # Create a TCP/IP socket
14     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # connect the socket to the server
15     server_address = (host, port)
16     print("Connecting to %s port %s" %server_address)
17     sock.connect(server_address) # Send data
18     try: # Send data
19         message = "Test message: SDN course examples"
20         print("Sending %s" % message)
21         sock.sendall(message.encode('utf_8')) # Look for the response
22         amount_received = 0
23         amount_expected = len(message)
24         while amount_received < amount_expected:
25             data = sock.recv(16)
26             amount_received +=len(data)
27             print("Received: %s" % data)
28     except socket.errno as e:
29         print("Socket error: %s" %str(e))
30     except Exception as e:
31         print("Other exception: %s" %str(e))
32     finally:
33         print("Closing connection to the server")
34         sock.close()
35 if __name__ == '__main__':
36     parser = argparse.ArgumentParser(description='Socket Server Example')
37     parser.add_argument('--port', action='store', dest="port", type=int, required=True)
38     given_args = parser.parse_args()
39     port = given_args.port
40     echo_client(port)(port)
```

What you need to do for running the program?

Answer:

To understand python socket programming, we need to know about three interesting topics – **Socket Server**, **Socket Client** and **Socket**.

So, what is a server? Well, a server is a software that waits for client requests and serves or processes them accordingly.

On the other hand, a client is requester of this service. A client program request for some resources to the server and server responds to that request.

Which program you need to run first client of server?

Answer:

To use python socket connection, we need to import socket module. Then, sequentially we need to perform some task to establish connection between server and client. We can obtain host address by using `socket.gethostname()` function.

Explain how the program works?**Answer:**

We have said earlier that a socket client requests for some resources to the socket server and the server responds to that request.

So we will design both server and client model so that each can communicate with them. The steps can be considered like this.

- i. Python socket server program executes at first and wait for any request
- ii. Python socket client program will initiate the conversation at first.
- iii. Then server program will response accordingly to client requests.
- iv. Client program will terminate if user enters "bye" message. Server program will also terminate when client program terminates, this is optional and we can keep server program running indefinitely or terminate with some specific command in client request.

What you need to do for communicating with another server in the classroom?**Answer:**

A network socket is an endpoint of a two-way communication link between two programs or processes - client and server in our case - which are running on the network. This can be on the same computer as well as on different systems which are connected via the network.

Both parties communicate with each other by writing to or reading from the network socket. The technical equivalent in reality is a telephone communication between two participants. The network socket represents the corresponding number of the telephone line, or a contract in case of cell phones

Questions

- **Question 5.1: Explain in your own words which are the difference between functions and modules?**

Answer:**Python Function:**

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Python gives you many built-in functions like `print()`, etc. but you can also create your own functions. These functions are called user-defined functions

Python Module:

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code

or

In programming, function refers to a segment that groups code to perform a specific task.

A module is a software component or part of a program that contains one or more routines.

That means, functions are groups of code, and modules are groups of classes and functions

• Question 5.2: Explain in your own words when to use local and global variables?

Answer:

Local Variable: Local variable is defined as a type of variable declared within programming block or subroutines. It can only be used inside the subroutine or code block in which it is declared. The local variable exists until the block of the function is under execution. After that, it will be destroyed automatically.

Example:

```
public int add(){  
    int a =4;  
    int b=5;  
    return a+b;  
}
```

Here, 'a' and 'b' are local variables

Global Variable: A global variable in the program is a variable defined outside the subroutine or function. It has a global scope means it holds its value throughout

the lifetime of the program. Hence, it can be accessed throughout the program by any function defined within the program, unless it is shadowed.

Example:

```
int a =4;
int b=5;
public int add(){
    return a+b;
}
```

Here, 'a' and 'b' are global variables.

• Question 5.3: Which is the role of sockets in computing networking? Are the sockets defined random or there is a rule?

Answer:

A socket uniquely identifies the endpoint of a communication link between two application ports.

A port represents an application process on a TCP/IP host, but the port number itself does not indicate the protocol being used: TCP, UDP, or IP. The application process might use the same port number for TCP or UDP protocols. To uniquely identify the destination of an IP packet arriving over the network, you have to extend the port principle with information about the protocol used and the IP address of the network interface; this information is called a socket. A socket has three parts: protocol, local-address, local-port

• Question 5.4: Why is relevant to have the IPv4 address of remote server?

Explain what is Domain Name System (DNS)?

Answer:

The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources. Each device connected to the Internet has a unique IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorize IP addresses such as 192.168.1.1 (in IPv4), or more complex newer alphanumeric IP addresses such as 2400:cb00:2048:1::c629:d7a2 (in IPv6)

- Question 5.5: Create a program that allows exchange messages increased by the user between client and server.

Answer:

Server code:

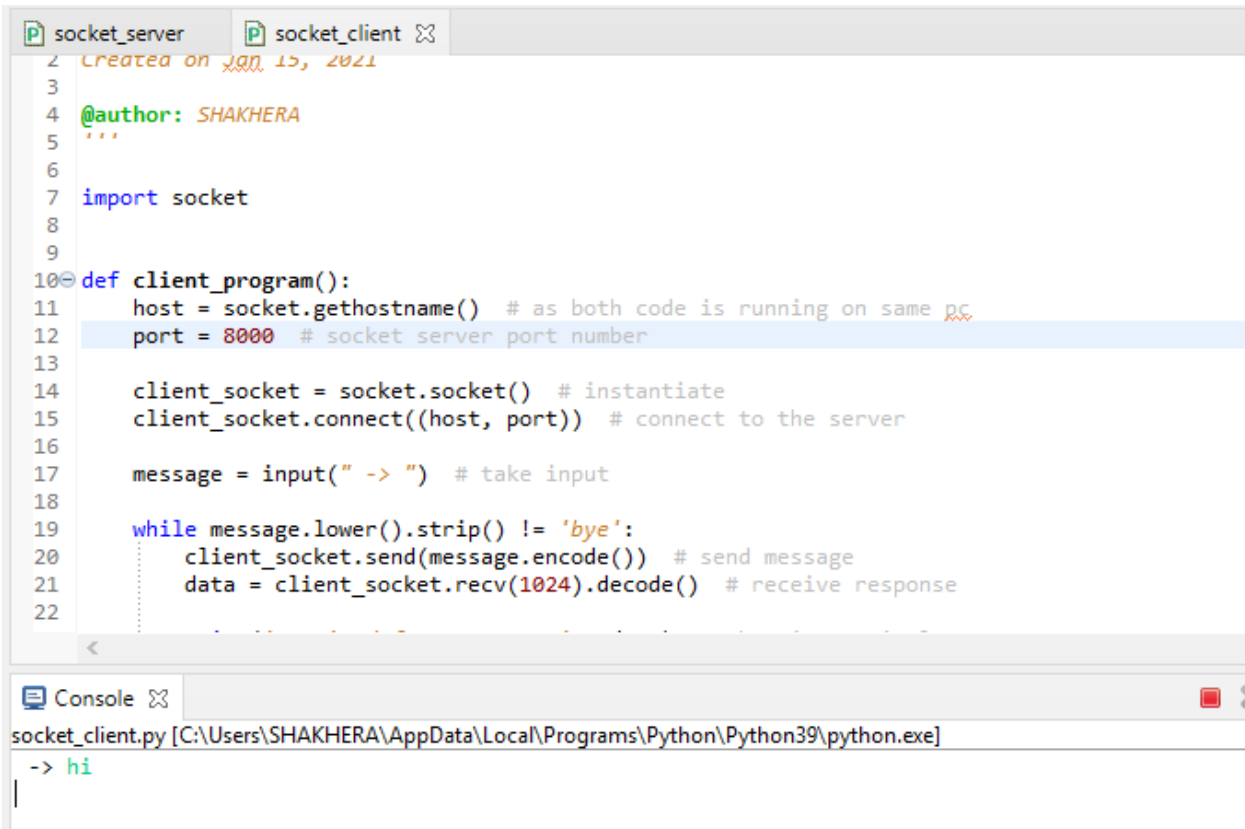
```
*socket_server  ✕  *socket_client

1  """
2  Created on Jan, 15, 2021
3
4  @author: SHAKHERA
5  """
6  import socket
7
8  def server_program():
9      # get the hostname
10     host = socket.gethostname()
11     port = 8000 # initiate port no above 1024
12
13     server_socket = socket.socket() # get instance
14     # look closely. The bind() function takes tuple as argument
15     server_socket.bind((host, port)) # bind host address and port together
16
17     # configure how many client the server can listen simultaneously
18     server_socket.listen(2)
19     conn, address = server_socket.accept() # accept new connection
20     print("Connection from: " + str(address))
21     while True:
22         # receive data stream. it won't accept data packet greater than 1024 bytes
23         data = conn.recv(1024).decode()
24         if not data:
25             # if data is not received break
26             break
27         print("from connected user: " + str(data))
28         data = input(' -> ')
29         conn.send(data.encode()) # send data to the client
30
31     conn.close() # close the connection
32
33 if __name__ == '__main__':
34     server_program()
35
```

Console ✕

socket_client.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]

Client code:



```
socket_server socket_client
2 created on 10/15, 2021
3
4 @author: SHAKHERA
5
6
7 import socket
8
9
10 def client_program():
11     host = socket.gethostname() # as both code is running on same pc
12     port = 8000 # socket server port number
13
14     client_socket = socket.socket() # instantiate
15     client_socket.connect((host, port)) # connect to the server
16
17     message = input(" -> ") # take input
18
19     while message.lower().strip() != 'bye':
20         client_socket.send(message.encode()) # send message
21         data = client_socket.recv(1024).decode() # receive response
22
```

```
Console
socket_client.py [C:\Users\SHAKHERA\AppData\Local\Programs\Python\Python39\python.exe]
-> hi
|
```

Discussion: 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.