Mawlana Bhashani Science and Technology University

Santosh, Tangail-1902.

Lab Report

Department of Information and Communication Technology

Report No: 02

Report Name: Programming with Python

Course Title: Network Planning and designing Lab.

Course Code: ICT-3208

Submitted By	Submitted To
Name: **Zafrul Hasan Khan & Hasibul Islam Imon** ID: **IT-18003 & IT-18047** Session: 2017-18 3rd Year 2nd Semester Dept. of Information & Communication Technology, MBSTU.	**Nazrul Islam** **Assistant Professor,** Dept. of Information & Communication Technology, MBSTU.

Objectives: The main objectives of this lab how to python function works, python modules works. To Understand the use of global and local variables and Learning the basis of networking programing 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).

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).

Modules: Modules allow reusing a number of functions in other programs.

Networking background for sockets: A socket is one endpoint of a two-way communication link between two programs running on the network or PC.

On the server-side: The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening.

Methodology :

Defining functions: Functions are defined using the def keyword.

```
def XX_YY(variable1, varible2)
```

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

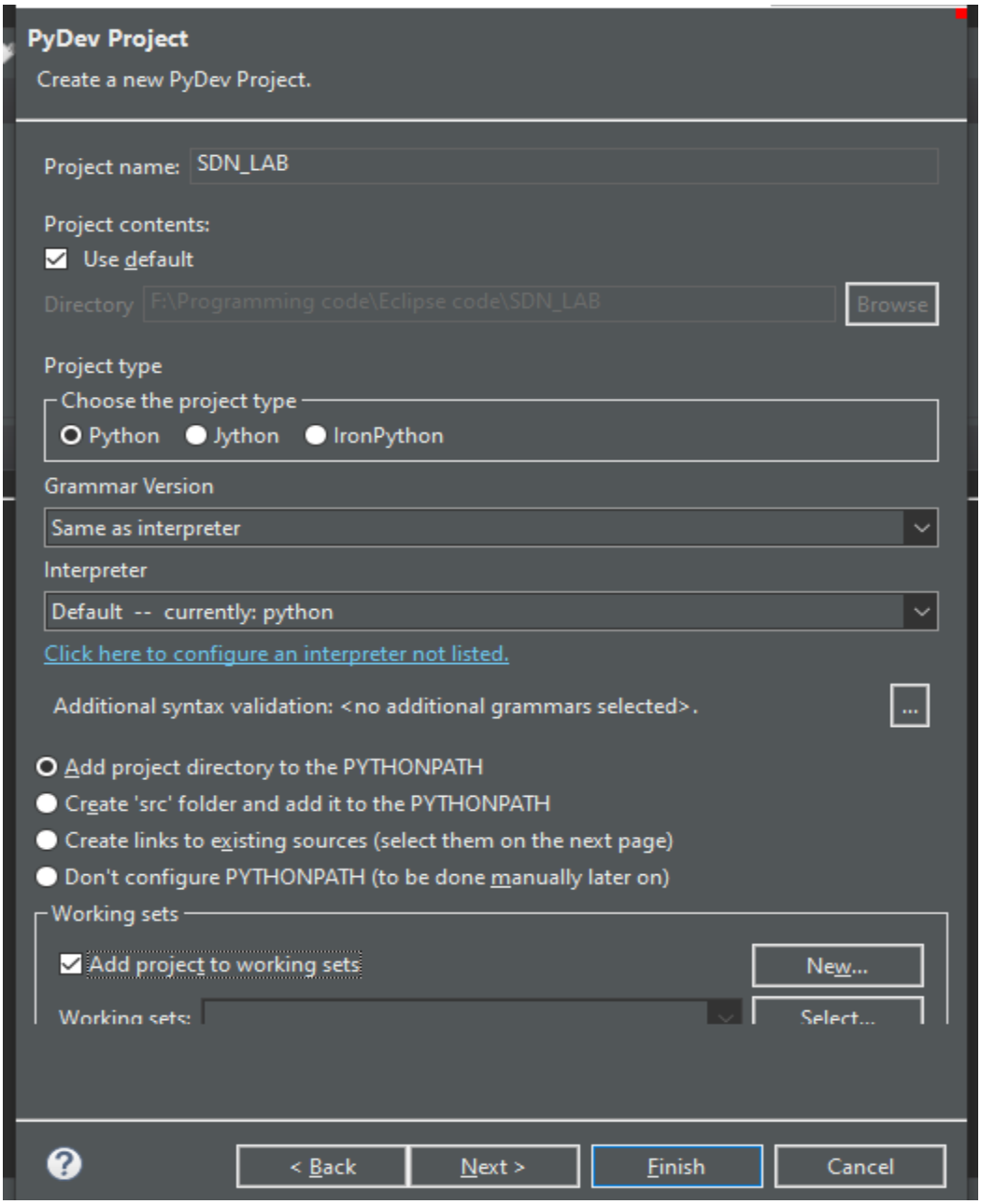
```
def xx_yy():
    aa
```

Using modules: A module can be imported by another program to make use of its functionality. This is how we can use the Python standard library as well.

```
import xx_yy
```

Exercises:

Exercise 4.1.1: Create a python project using with SDN_LAB



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

```python
'''
Created on Jan 15, 2021

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

```
<terminated> function.py [C:\Users\Zafrul Hasan Nasim\AppData\Lo
hello world
```

No parameter needed for this above function.

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

Output of this function:

function_2

```python
'''
Created on Jan 15, 2021

@author: Zafrul Hasan Nasim
'''
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)
```

Console

```
<terminated> function_2.py [C:\Users\Zafrul Hasan Nasim\AppDat
4 is maximum
7 is maximum
```

This function need two parameter.

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

function_local

```python
'''
Created on Jan 15, 2021

@author: Zafrul Hasan Nasim
'''
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\Zafrul Hasan Nasim\
x is 50
Changed local x to 2
x is still 50
```

Final value of the variable x is 50. variable x does not change to 2 because x is local variable that is declared inside a function definition are not related in any way to other variables with the same names used outside the function.

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

```
function_global

@author: Zafrul Hasan Nasim
'''
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\Zafrul Hasan Nasim\
x is 50
Changed global x to 2
Value of x is 2
```

The final value of variable x is 2 . variable x change this time because this time variable x is defined as globally.

Exercise 4.1.6: Python modules

```
mymodule (src1)    module_demo    module_demo2

'''
Created on Jan 15, 2021

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

```
Console
<terminated> module_demo2.py [C:\Users\Zafrul Hasan Nasim\AppData\Local\
Hi, this is mymodule speaking.
Version 0.1
```

mymodule (src1) | module_demo

```python
'''
Created on Jan 15, 2021

@author: Zafrul Hasan Nasim
'''
import mymodule
if __name__ == '__main__':
mymodule.say_hi()
print('Version', mymodule.__version__)
```

Console

```
<terminated> module_demo.py [C:\Users\Zafrul Hasan Nasim\AppData\Loca
Hi, this is mymodule speaking.
Version 0.1
```

mymodule (src1) | module_demo | module_demo2

```python
'''
Created on Jan 15, 2021

@author: Zafrul Hasan Nasim
'''
from mymodule import say_hi, __version__
if __name__ == '__main__':
say_hi()
print('Version', __version__)
```

Console

```
<terminated> module_demo2.py [C:\Users\Zafrul Hasan Nasim\AppData\Loca
Hi, this is mymodule speaking.
Version 0.1
```

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

```
local_machine_info
'''
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> local_machine_info.py [C:\Users\Zafrul Hasan Nasim\AppData\Local\Pro
 Host name: DESKTOP-96MSSI0
 IP address: 192.168.56.1
```

Exercise 4.2.2: Retrieving a remote machine's IP address

```
local_machine_info    *remote_machine_info

@author: Zafrul Hasan Nasim
'''
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
<terminated> remote_machine_info.py [C:\Users\Zafrul Hasan Nasim\AppData\Local\Programs\Python\Python39\py
 Remote host name: www.python.org
Error accesing www.python.org: error number and detail [Errno 11001]
```

Exercise 4.2.3: Converting an IPv4 address to different formats

```python
ip4_address_conversion
@author: Zafrul Hasan Nasim
'''
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()
```

```
Console
<terminated> ip4_address_conversion.py [C:\Users\Zafrul Hasan Nasim\AppData\Local\Programs\Python\Python39\pyth
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
finding_service_name
'''
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()
```

```
Console
<terminated> finding_service_name.py [C:\Users\Zafrul Hasan Nasim\AppData\Local\Pr
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

socket_timeout

```python
@author: Zafrul Hasan Nasim
'''
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()
```

Console

```
<terminated> socket_timeout.py [C:\Users\Zafrul Hasan Nasim\AppData\Local\Programs\Python\Pyt
Default socket timeout: None
Current socket timeout: 100.0
```

Exercise 4.2.6: Writing a simple echo client/server application (Tip: Use port 9900)
Create python scrip using the syntax below (save as echo_server.py):

echo_server echo_client

```python
@author: Zafrul Hasan Nasim
'''
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
connections
    sock.listen(backlog)

    while True:
    print ("Waiting to receive message from client")
    client, address = sock.accept()
    data = client.recv(data_payload)
```

```python
    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)
```

Create python scrip using the syntax below (save as echo_client.py):

echo_server | *echo_client

```python
@author: Zafrul Hasan Nasim
'''
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
    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    sock.connect(server_address)
    try:
    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:
```

```python
    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()
    port = given_args.p
    echo_client(port)
```

Conclusion : From this lab , I have known that how to understand python function works and python modules works. I also learnt that how to use of global and local variables. Beside, Learning the basis of networking programing with python . In this lab , I have understood that networking programing with sockets for server and client. In Server and Client normally a server runs on a specific computer and has a socket that is bound to a specific port number.