

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.

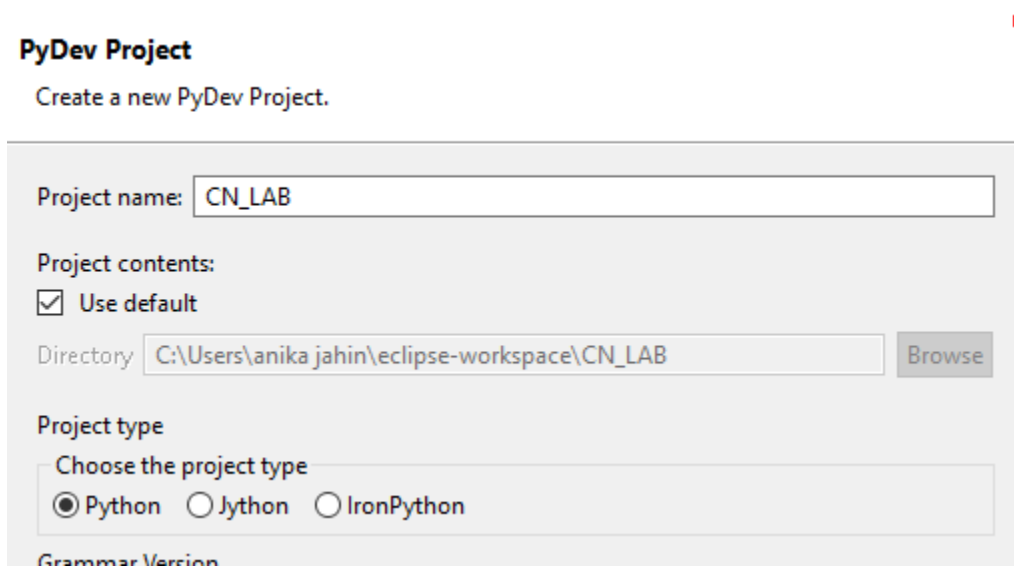
Local Variables: Variables declared inside a function definition are not related in any way to other variables with the same names used outside the function (variable names are local to the function). This is called the scope of the variable. All variables have the scope of the block they are declared 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



The screenshot shows the 'PyDev Project' dialog box. At the top, it says 'PyDev Project' and 'Create a new PyDev Project.' Below this, there are several sections: 'Project name:' with a text box containing 'CN_LAB'; 'Project contents:' with a checked checkbox 'Use default'; 'Directory:' with a text box containing 'C:\Users\anika jahin\eclipse-workspace\CN_LAB' and a 'Browse' button; 'Project type:' with a label 'Choose the project type' and three radio buttons: 'Python' (selected), 'Jython', and 'IronPython'. At the bottom, there is a partially visible 'Grammar Version' section.

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

```
function function_2
1 '''
2 Created on Aug 29, 2020
3
4 @author: anika jahin
5 '''
6 def say_hello(): # block belonging to the function
7     print('hello world') # End of function
8 if __name__ == '__main__':
9     say_hello()
```

Console

<terminated> function.py [C:\Users\anika jahin\AppData\Local\Programs\Python\Python39-64\python.exe]
hello world

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

```
function function_2
1 '''
2 Created on Aug 29, 2020
3
4 @author: anika jahin
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
14 if __name__ == '__main__':
15     pass
16     print_max(3, 4) # directly pass lit
17     x = 5
18     y = 7 # pass variables as arguments
19     print_max(x, y)
```

Console

<terminated> function_2.py [C:\Users\anika jahin\AppData\Local\Programs\Python\Python39-64\python.exe]
4 is maximum
7 is maximum

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

```
function function_2 function_local x
1 '''
2 Created on Aug 29, 2020
3
4 @author: anika jahin
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\anika jahin\AppData\
x is 50
Changed local x to 2
x is still 50
```

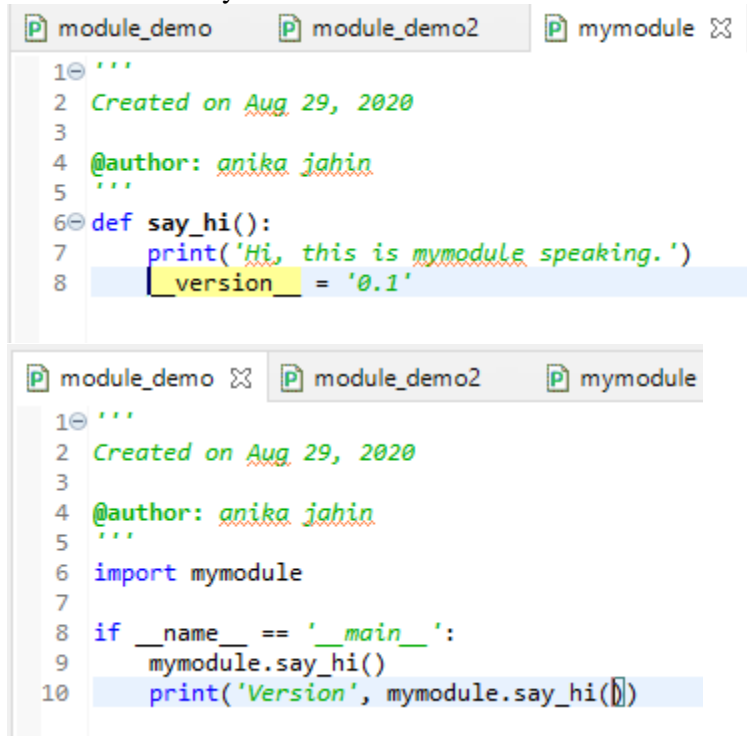
Exercise 4.1.5: Global variable (save as function_global.py)

```
function function_2 function_local function_global x
1 '''
2 Created on Aug 29, 2020
3
4 @author: anika jahin
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\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

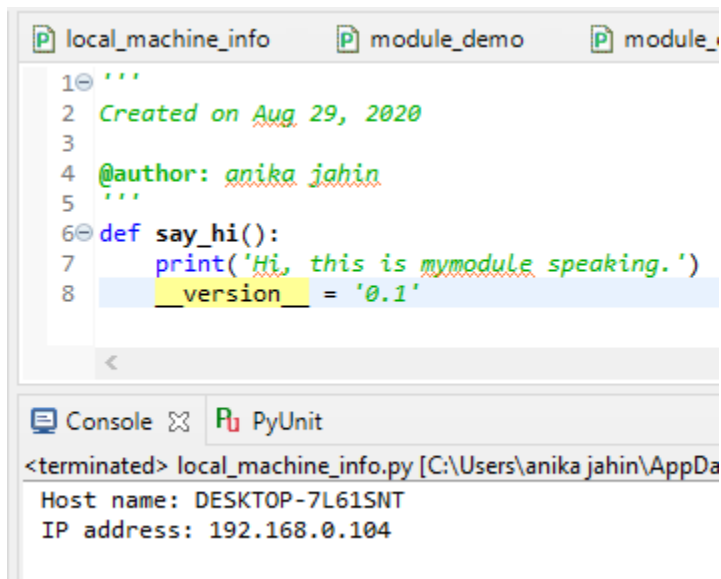


```
1-'''
2  Created on Aug 29, 2020
3
4  @author: anika jahin
5  '''
6-def say_hi():
7    print('Hi, this is mymodule speaking.')
8    __version__ = '0.1'

module_demo  module_demo2  mymodule

1-'''
2  Created on Aug 29, 2020
3
4  @author: anika jahin
5  '''
6  import mymodule
7
8  if __name__ == '__main__':
9      mymodule.say_hi()
10     print('Version', mymodule.__version__)
```

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



```
local_machine_info  module_demo  module_

1-'''
2  Created on Aug 29, 2020
3
4  @author: anika jahin
5  '''
6-def say_hi():
7    print('Hi, this is mymodule speaking.')
8    __version__ = '0.1'

Console  PyUnit

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

Exercise 4.2.2: Retrieving a remote machine's IP address

```
remote_machine_info  local_machine_info  module
1 '''
2 Created on Aug 29, 2020
3
4 @author: anika jahin
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
13 if __name__ == '__main__':
14    print_machine_info()
```

Console PyUnit

<terminated> remote_machine_info.py [C:\Users\anika jahin\AppData\Local\Programs\Python\Python38-32\python.exe]
Remote host name: www.python.org
IP address: 151.101.8.223

Exercise 4.2.3: Converting an IPv4 address to different formats

```
module_demo  module_demo2  mymodule  ip4_address_conversion
1 '''
2 Created on Aug 29, 2020
3
4 @author: anika jahin
5 '''
6 import socket
7 from binascii import hexlify
8 def convert_ip4_address():
9     for ip_addr in ['127.0.0.1', '192.168.0.1']:
10         packed_ip_addr = socket.inet_aton(ip_addr)
11         unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
12         print (" IP Address: %s => Packed: %s, Unpacked: %s" %(ip_addr, hexlify(packed_ip_addr), unpacked_ip_addr))
13 if __name__ == '__main__':
14    convert_ip4_address()
```

Console PyUnit

<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

```
module_demo module_demo2 mymodule ip4_address_conversion finding_service_name X
1 '''
2 Created on Aug 29, 2020
3
4 @author: anika jahin
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, socket.getservbyport(port, protocolname)))
11         print ("Port: %s => service name: %s" %(53, socket.getservbyport(53, 'udp')))
12 if __name__ == '__main__':
13     find_service_name()
```

Console PyUnit

<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

```
module_demo module_demo2 mymodule socket_timeout X
1 '''
2 Created on Aug 29, 2020
3
4 @author: anika jahin
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 PyUnit

<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
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:


```
echo_server echo_client module_demo module_demo2 mymodule
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...