MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

Santosh,Tangail – 1902

Course Title : Computer Networks Lab

Lab Report Name : Programming with Python

Lab Report No : 02

Submitted by,   **Submitted to,**

 Name: Tania sultana asha &   **NAZRUL ISLAM**

     Wahidul hasan piyal   **Assistant Professor**

ID: IT-18010 & 18048   **Dept. of ICT, MBSTU**

Session: 2017-18

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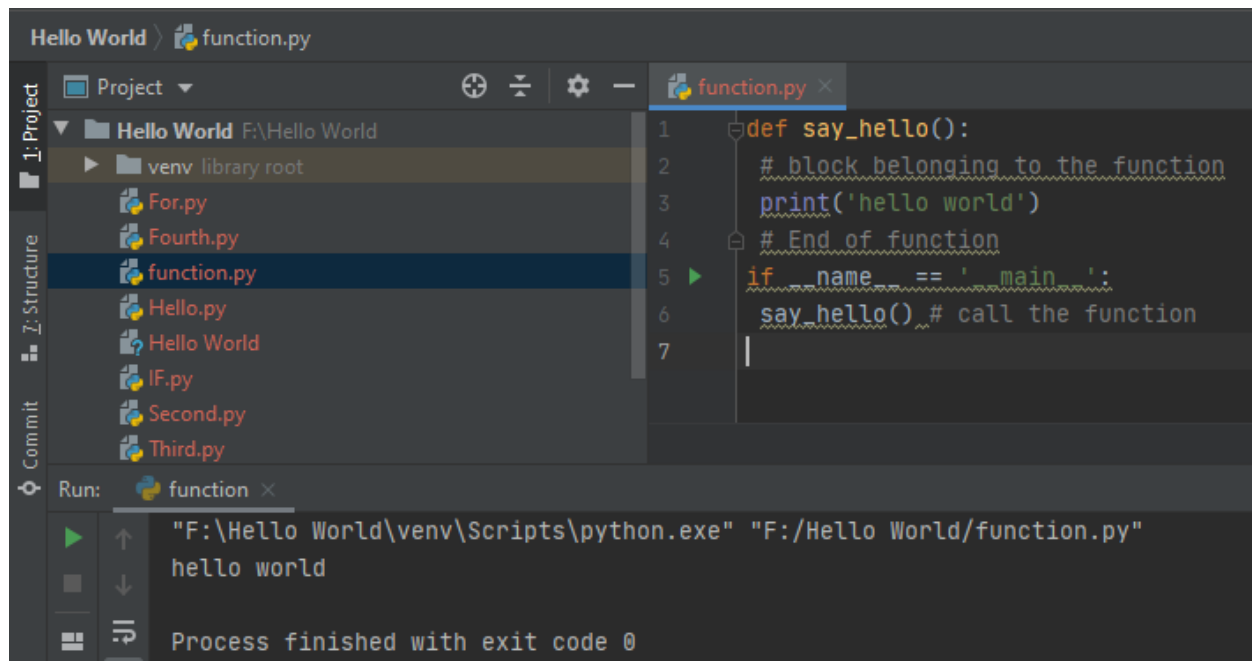
Modules: Modules allow reusing a number of functions in other programs.

Exercises:

Exercise 4.1.1: Create a python project using with SDN_LAB

Project name: SDN_LAB

Project contents:
☑ Use default

Directory C:\Users\JTech\eclipse-workspace\SDN_LAB   Browse

Project type
Choose the project type
◉ Python   ○ Jython   ○ IronPython

Grammar Version
Same as interpreter   ⌄
Interpreter

Please configure an interpreter before proceeding.

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   New...

Working sets: Select...

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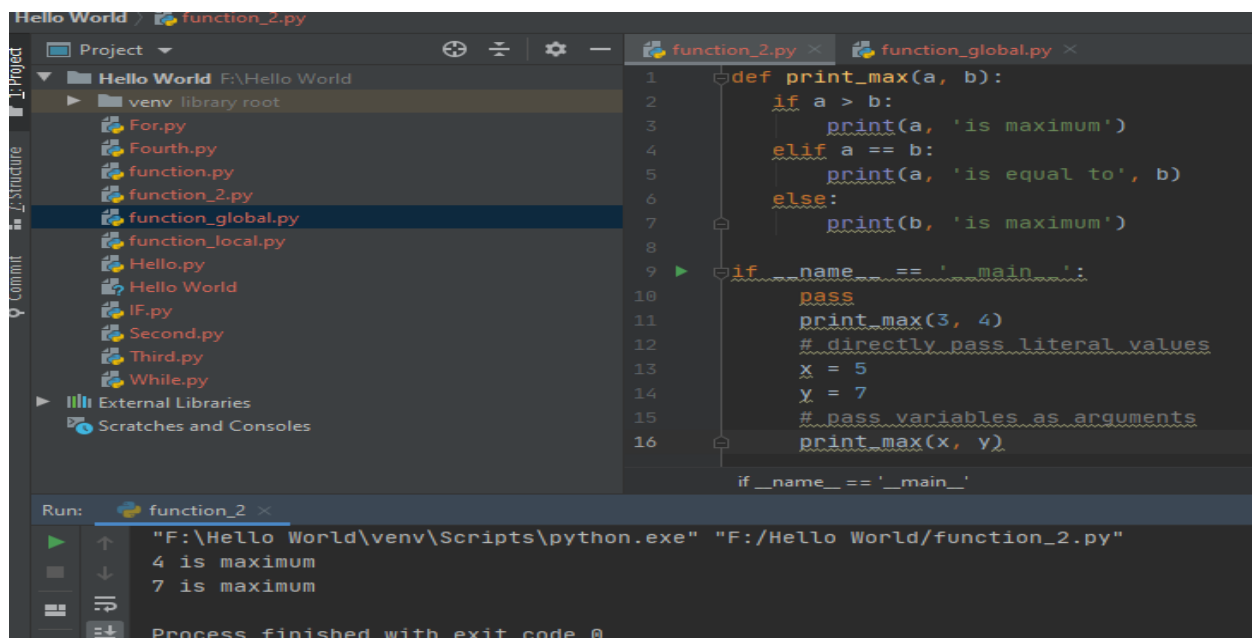

```python
def say_hello():
    # block belonging to the function
    print('hello world')
    # End of function
if __name__ == '__main__':
    say_hello() # call the function
```

```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/function.py"
hello world

Process finished with exit code 0
```

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


```python
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)
    # directly pass literal values
    x = 5
    y = 7
    # pass variables as arguments
    print_max(x, y)
```

```
if __name__ == '__main__'
```

```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/function_2.py"
4 is maximum
7 is maximum

Process finished with exit code 0
```

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

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

Exercise 4.1.6: Python modules

```python
def say_hi():
    print('Hi, this is mymodule speaking.')
    __version__ = '0.1'
```



```python
import mymodule
if __name__ == '__main__':
    mymodule.say_hi()
    print('Version', mymodule.__version__)

    from mymodule import say_hi, __version__


if __name__ == '__main__':
    say_hi()
    print('Version', __version__)
```

```
if _name_ == '_main_' > if _name_ == '_main_'
```

Run: module_demo

```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/module_demo.py"
Hi, this is mymodule speaking.
Version 0.1
Hi, this is mymodule speaking.
Version 0.1

Process finished with exit code 0
```

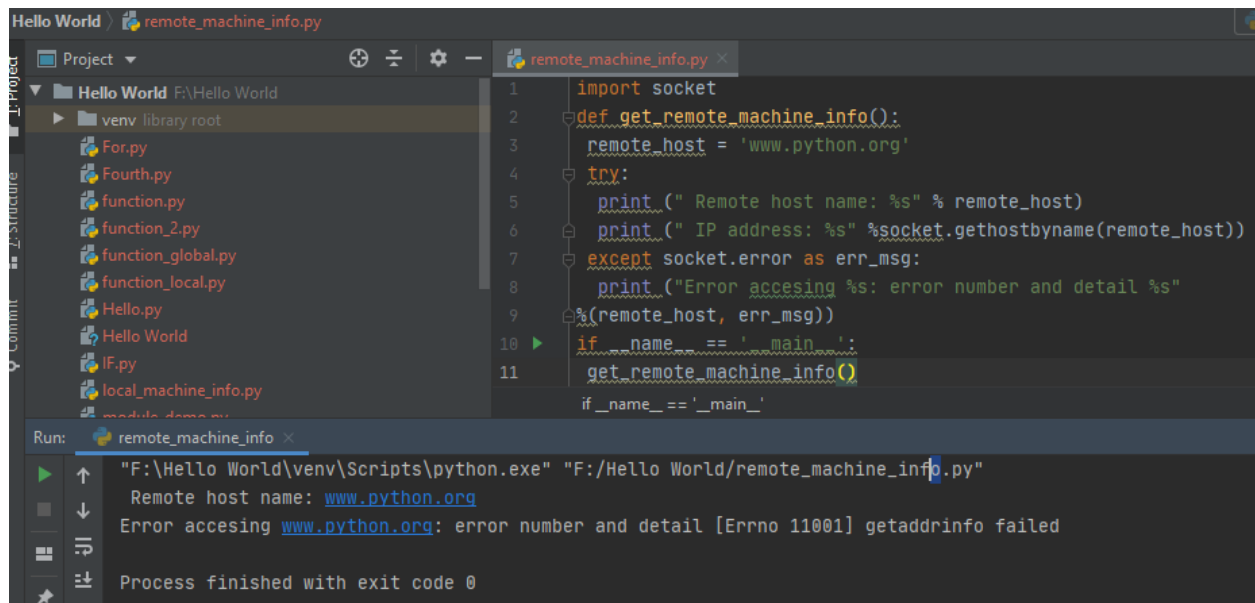## Exercise 4.2.1: Printing your machine's name and IPv4 address


```python
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()
```

```
if __name__ == '_main_'
```

Run: local_machine_info

```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/local_machine_info.py"
Host name: DESKTOP-FB5S452
IP address: 127.0.0.1

Process finished with exit code 0
```

Exercise 4.2.2: Retrieving a remote machine's IP address


```python
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()
```
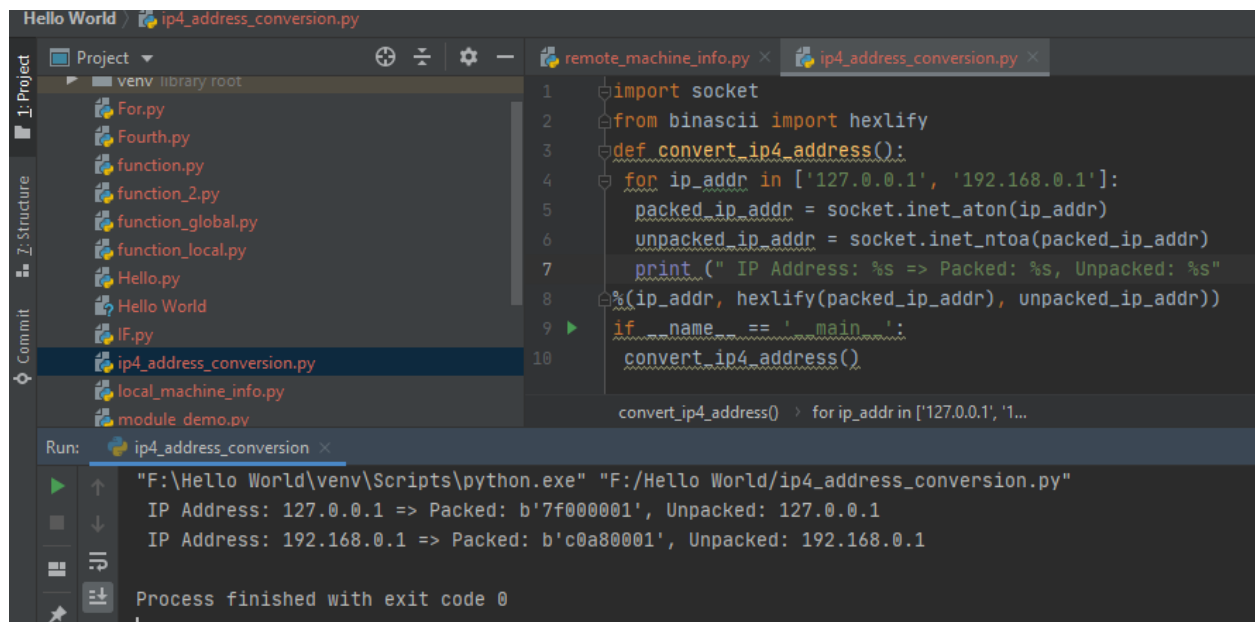
```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/remote_machine_info.py"
 Remote host name: www.python.org
Error accesing www.python.org: error number and detail [Errno 11001] getaddrinfo failed

Process finished with exit code 0
```

Exercise 4.2.3: Converting an IPv4 address to different formats


```python
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()
```

```
"F:\Hello World\venv\Scripts\python.exe" "F:/Hello World/ip4_address_conversion.py"
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

Process finished with exit code 0
```

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

Exercise 4.2.5: Setting and getting the default socket timeout

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

Server Code:

```python
import socket
import sys
import argparse
import codecs
from codecs import encode, decode
host = 'localhost'
data_payload = 4896
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)
    # Bind the socket to the port
    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 queued connections
    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)

        echo_server()
```

Client Code:

```python
#!/usr/bin/env 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
    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'))
        # Look for the response
        amount_received = 0
```

```
  echo_server.py ×      echo_client.py ×
22              # Look for the response
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,
39      required=True)
40      given_args = parser.parse_args()
41      port = given_args.port
42      echo_client(port)
```

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.