Mawlana Bhashani Science and Technology University

Lab-Report

Lab Report No : 02

Lab Report Name : Programming with Python

Course Title : Computer Networks Lab

Submitted by

Name: Moniruzzaman & Mst.Zakia Sultana

ID: IT-18007 & IT-18027

3rd year 2nd semester

Session: 2017-2018

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.

Networking background for sockets

What is a socket and how use it? A socket is one endpoint of a two-way communication link between two programs running on the network or PC. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. Endpoint: An endpoint is a combination of an IP address and a port number. Server and Client: Normally, a server runs on a specific computer and has a socket that is bound to a specific port number.

- **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. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

- **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.

  TCP is the protocol of choice for many of the most popular uses for the internet, including WWW, FTP, SSH, and email. It is safe to say that the internet we know today would not be here without TCP.

- **UDP:** UDP stands for user datagram protocol. It is a popular companion protocol to TCP and is also implemented in the transport layer.

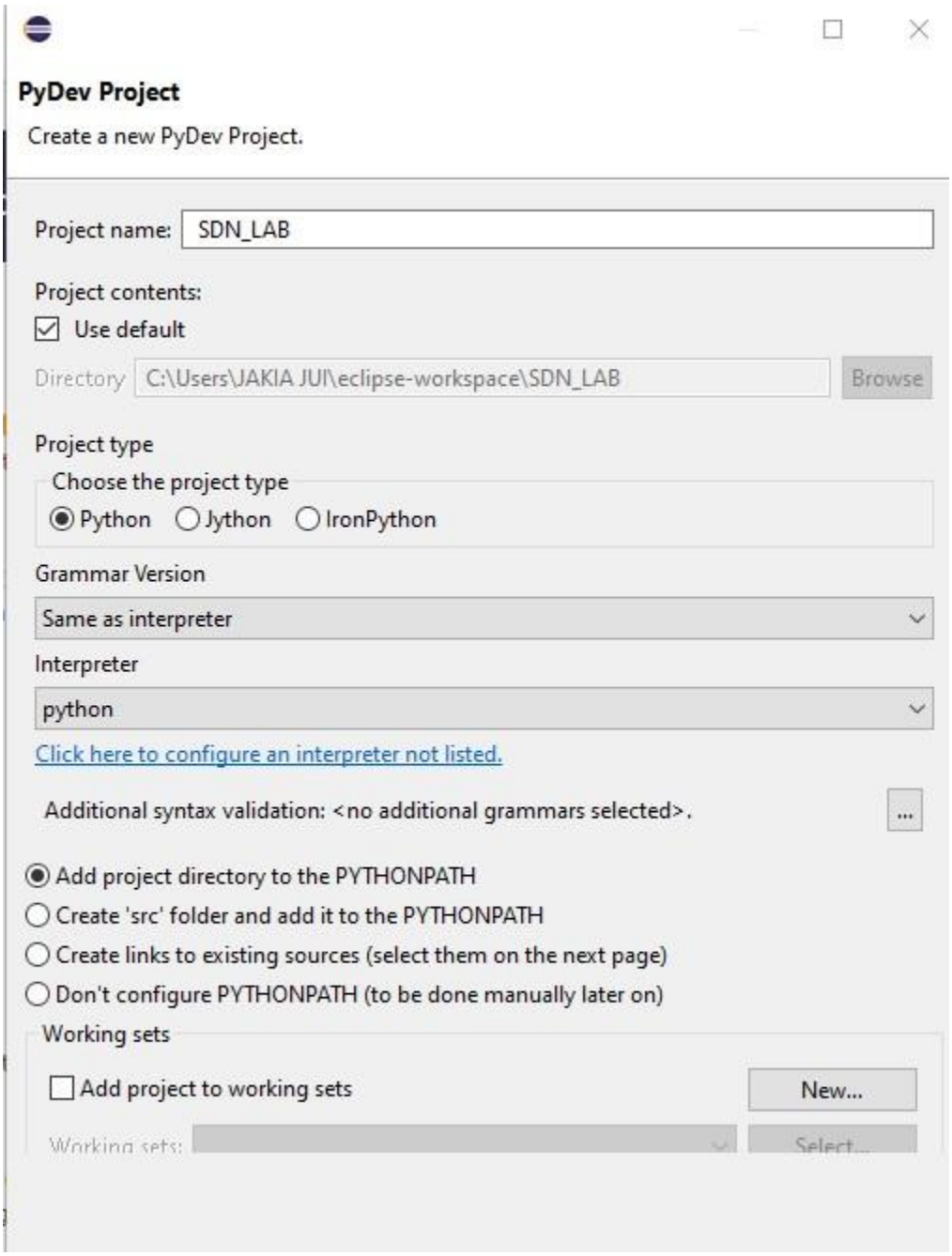
The fundamental difference between UDP and TCP is that UDP offers unreliable data transfer. It does not verify that data has been received on the other end of the connection. This might sound like a bad thing, and for many purposes, it is. However, it is also extremely important for some functions.

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.

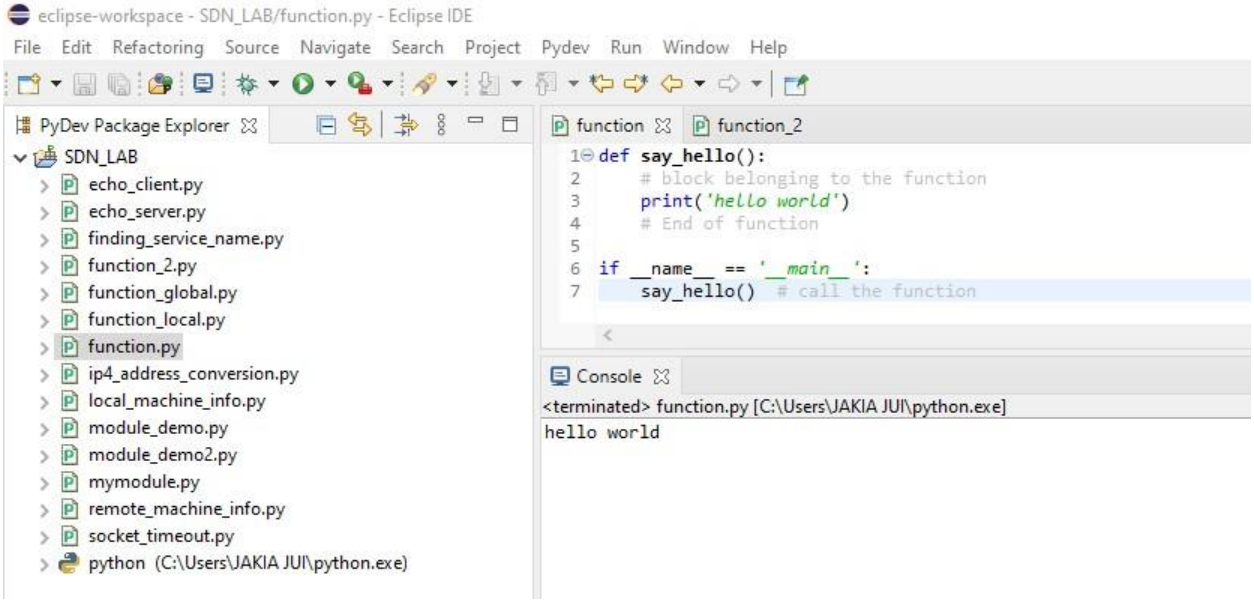
Exercises:

Section 4.1: Python function variables and modules.

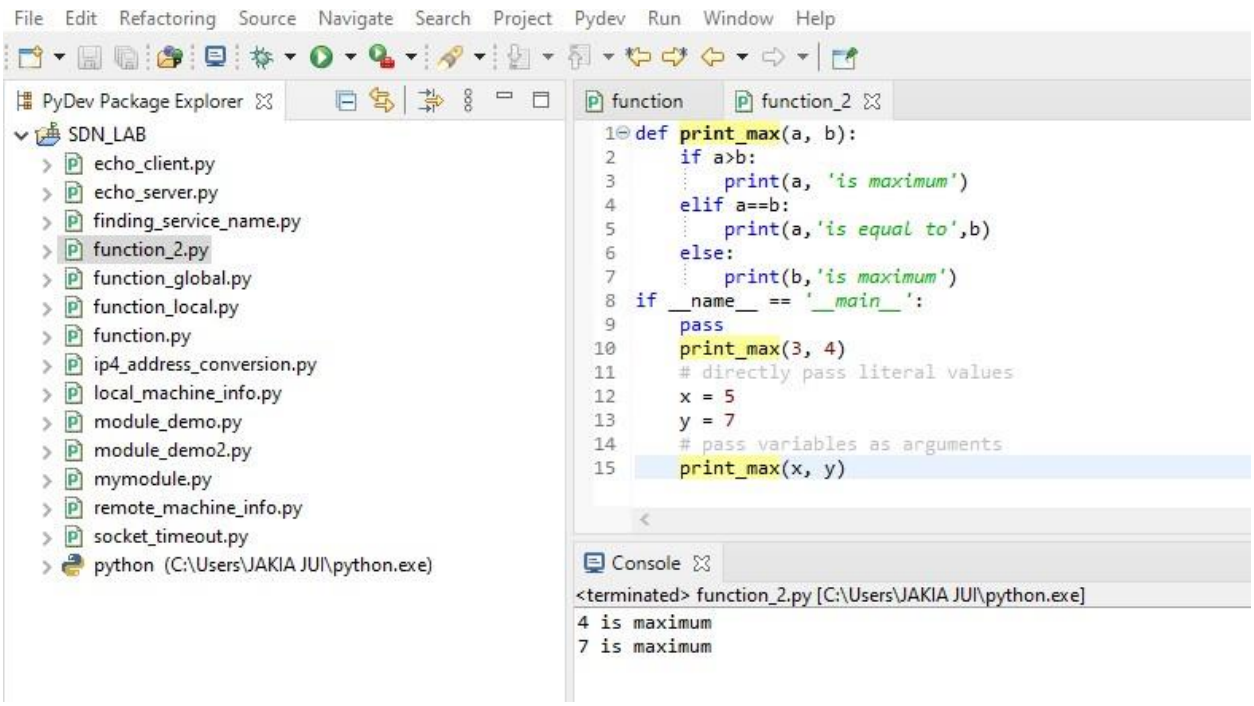
Exercise 4.1.1: Create a python project using with SDN_LAB



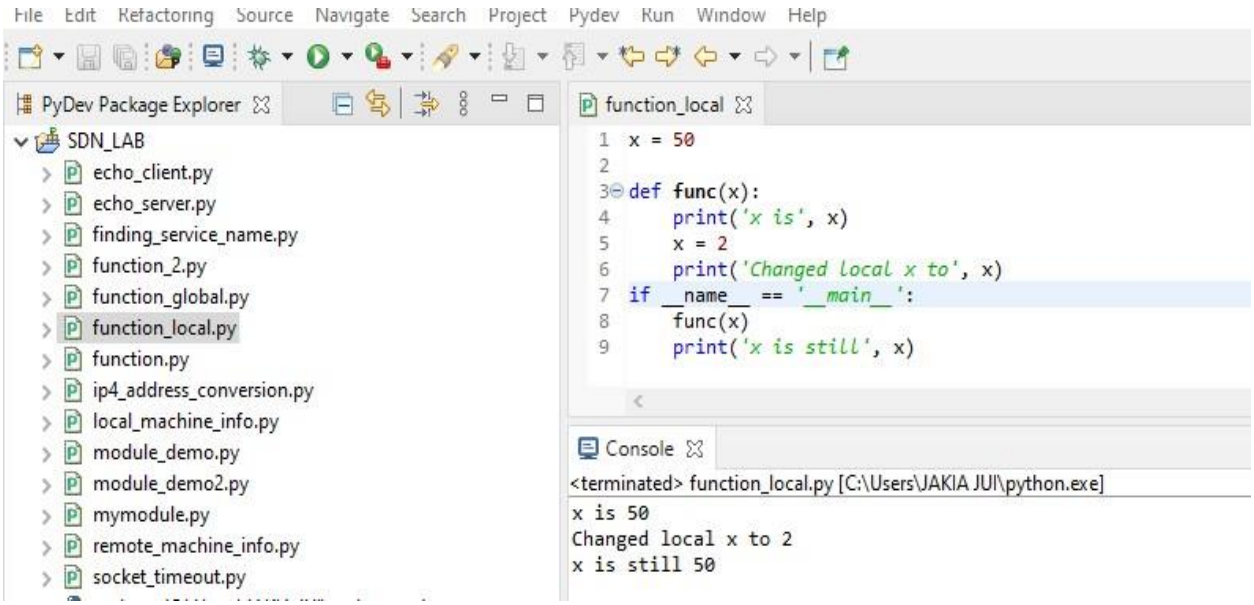
Exercise 4.1.2: Python function (save as function.py)



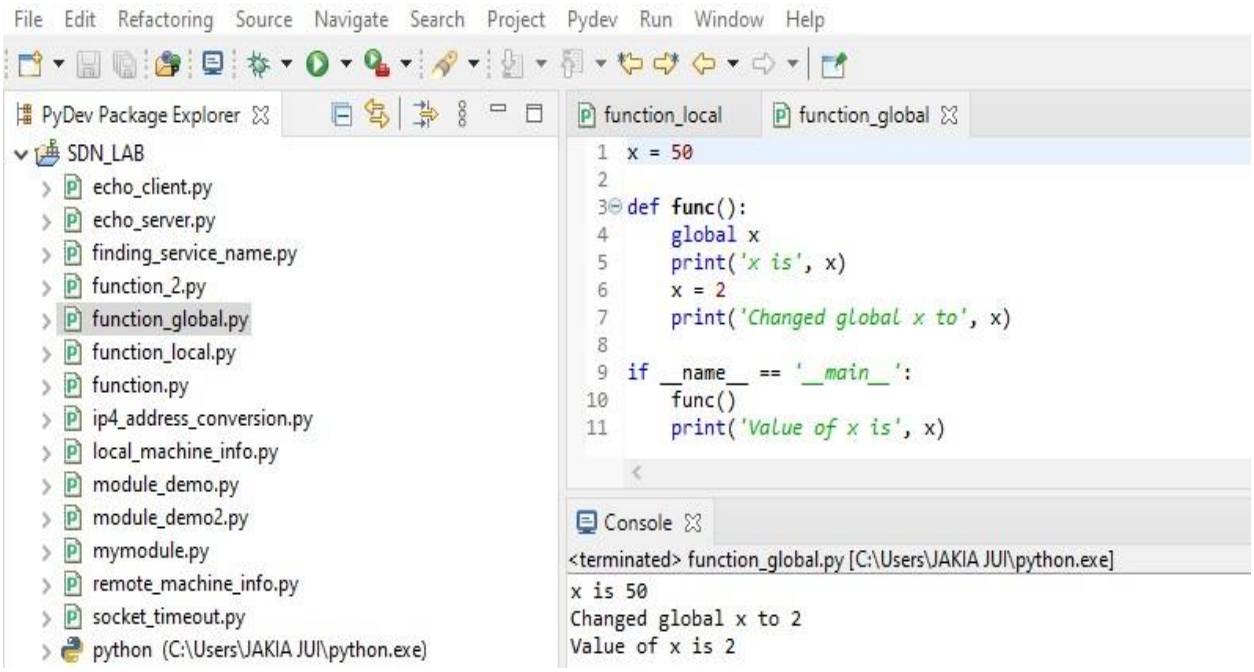
Exercise 4.1.3: Python function (save as function_2.py)



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



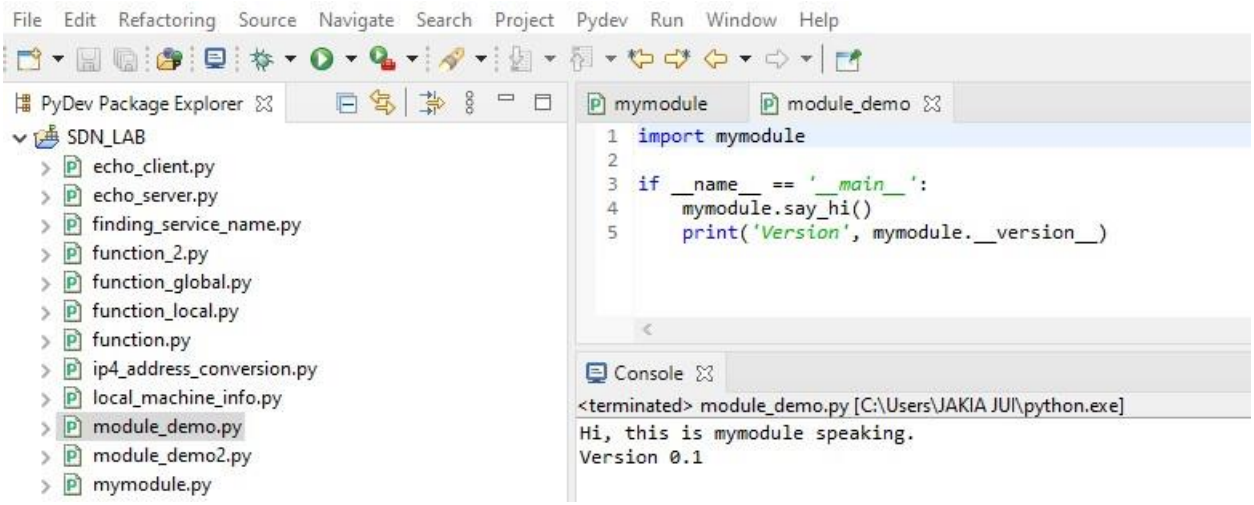
Exercise 4.1.5: Global variable (save as function_global.py)

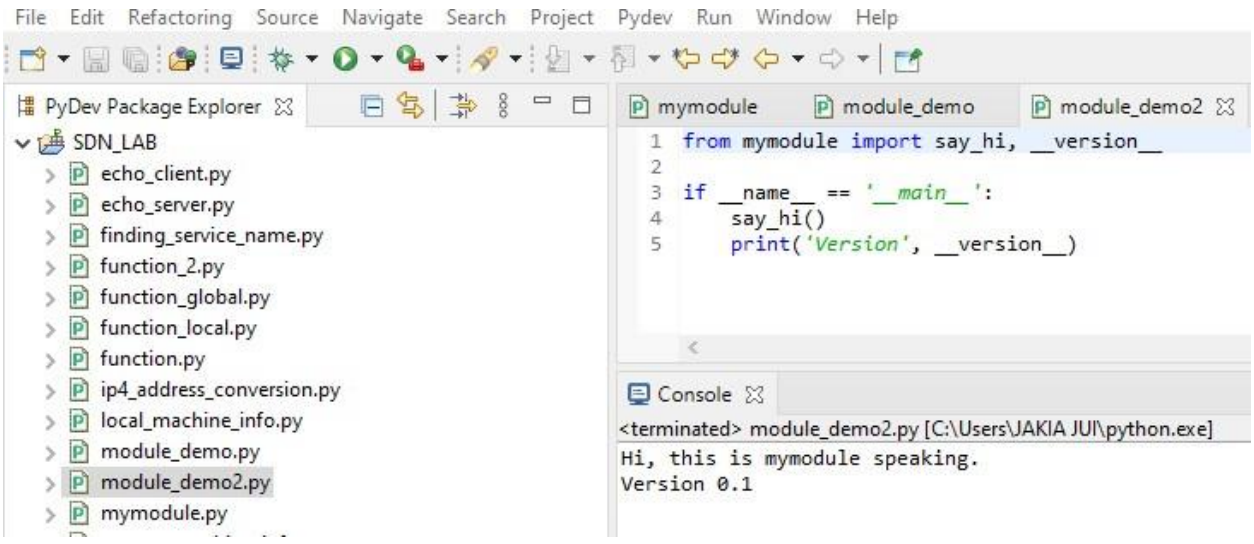


Exercise 4.1.6: Python modules

mymodule

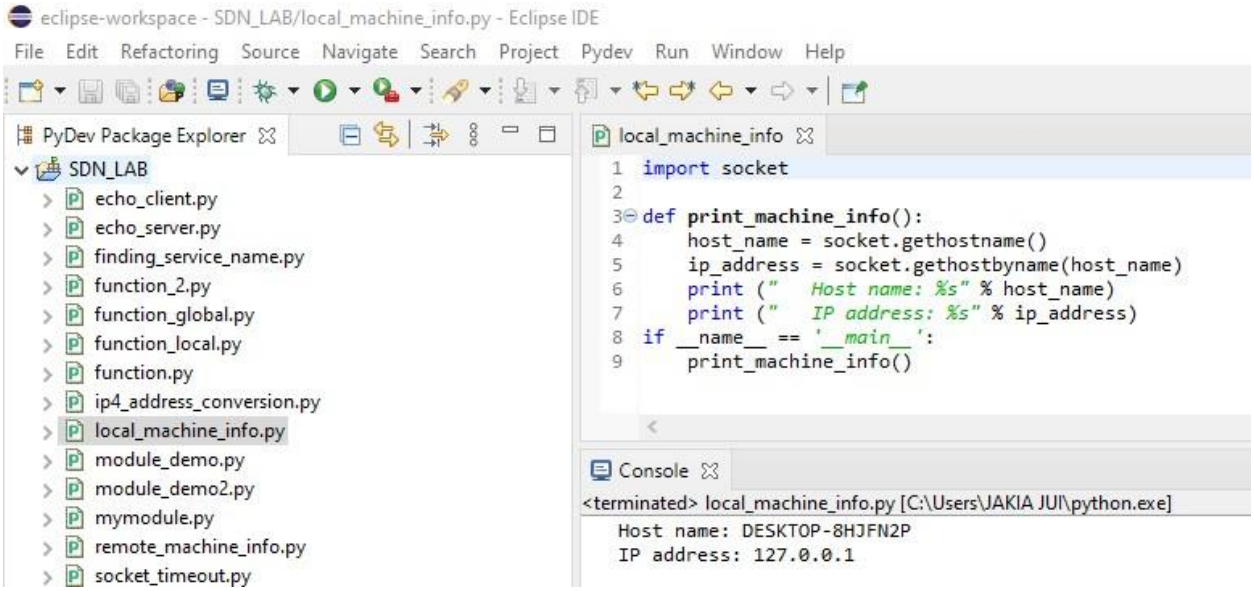
```
1  def say_hi():
2      print('Hi, this is mymodule speaking.')
3
4  __version__ = '0.1'
```



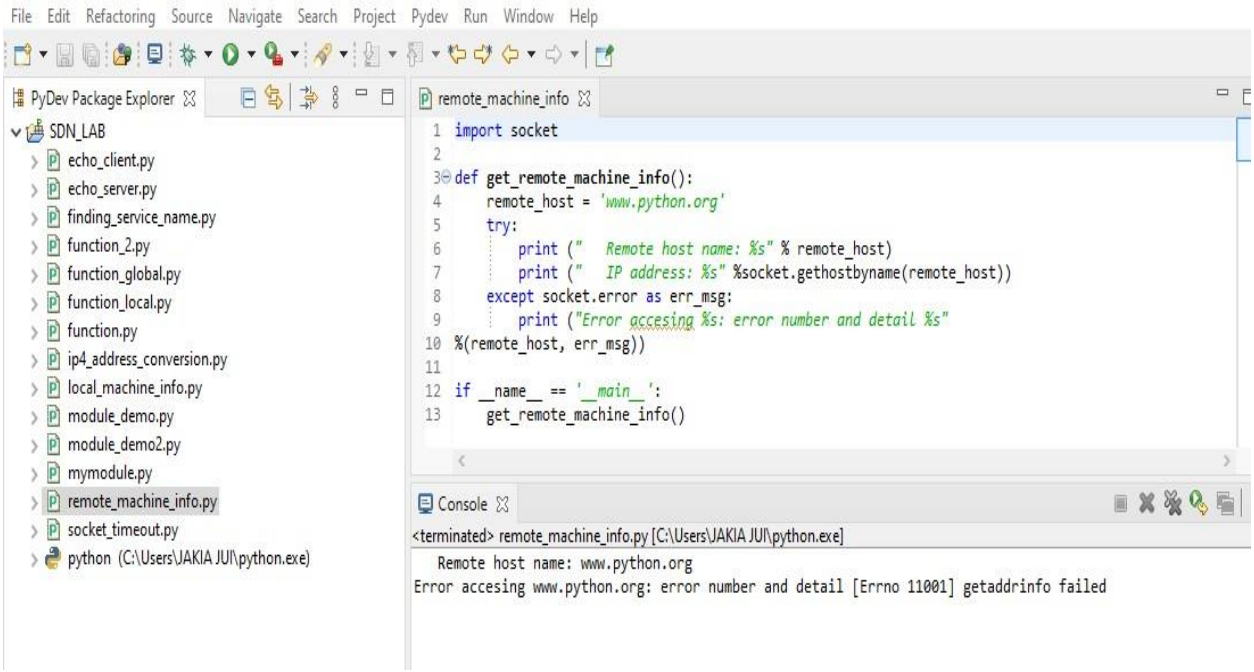


Section 4.2: Sockets, IPv4, and Simple Client/Server Programming.

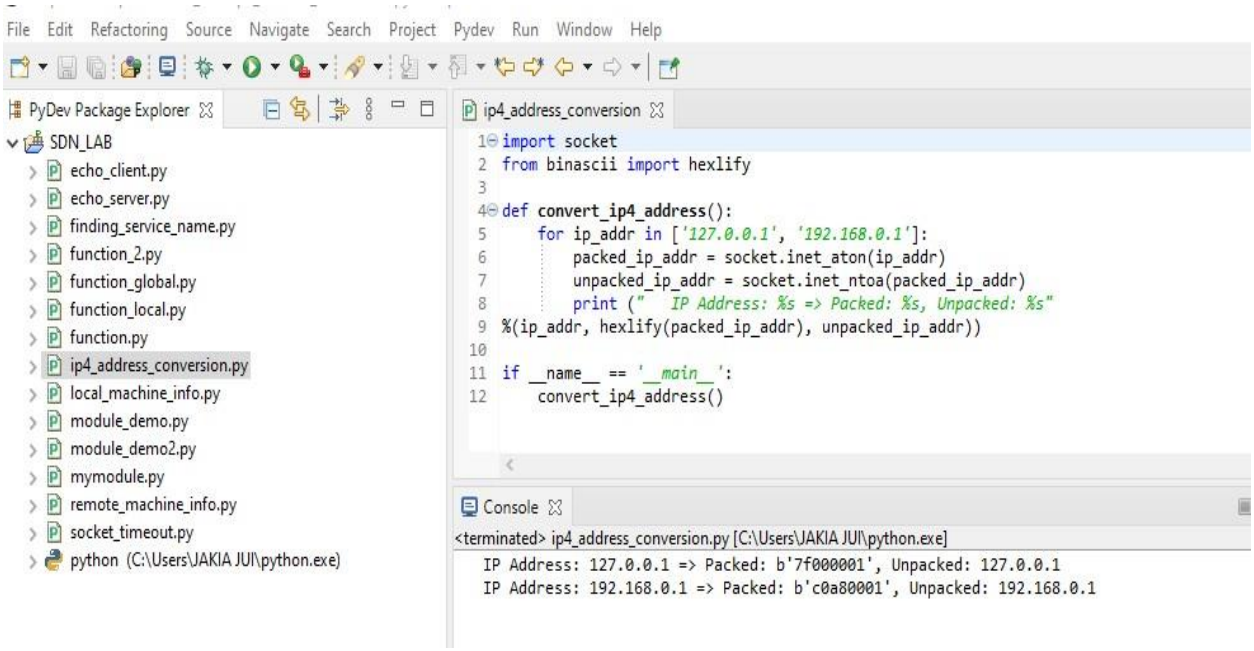
Exercise 4.2.1: Printing your machine's name and IPv4 address



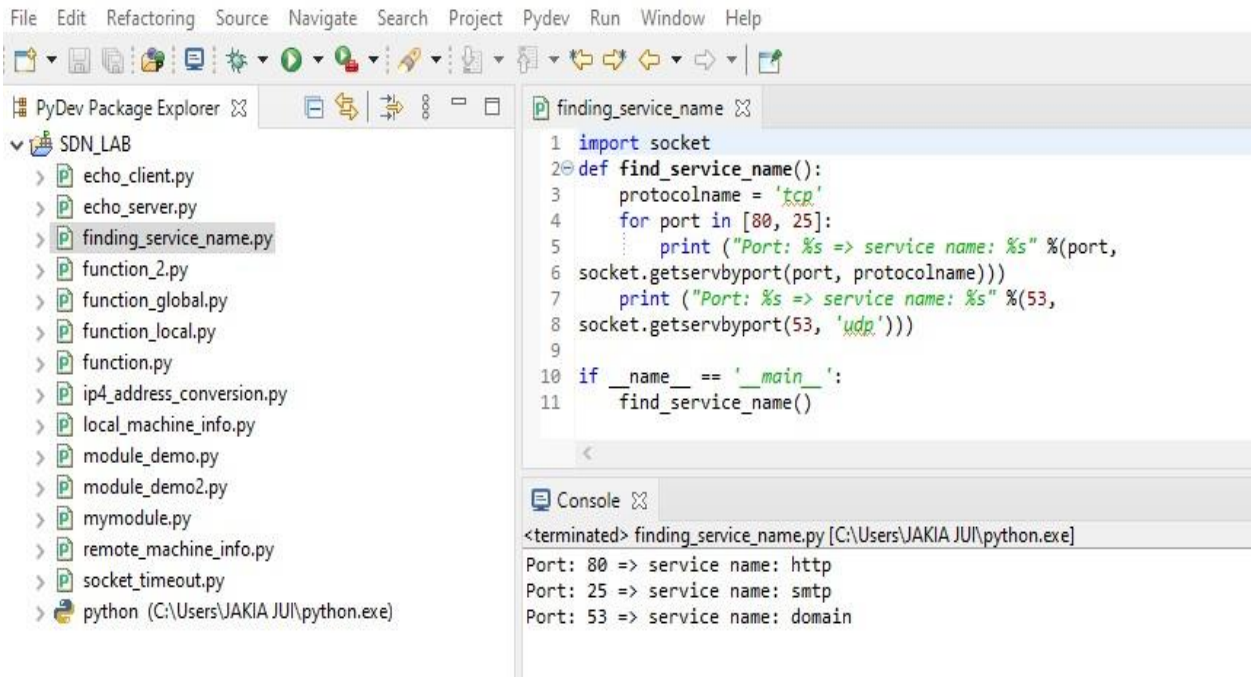
Exercise 4.2.2: Retrieving a remote machine's IP address



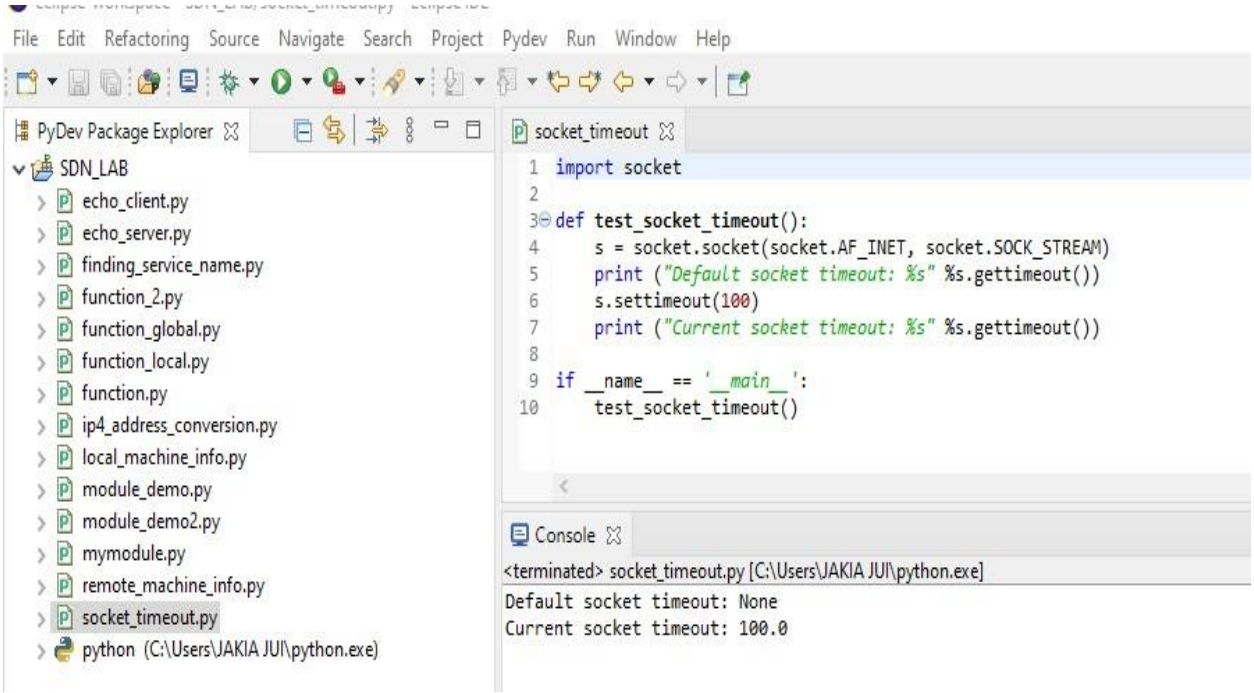
Exercise 4.2.3: Converting an IPv4 address to different format



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 = 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)
    # 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()
```

```python
        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)
```

Client Code:

echo_server | *echo_client

```
#!/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
        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:
        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.port
    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.

Questions:

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

Answer:

Function is the term derived from maths and means routine (a piece of code) that accepts some arguments and after execution provides a result. In C and C++ this is the only official name for routines.

Module is the term describing the aggregation of multiple function into some logical block that can be referred to and used in other areas of the application. While the term is a bit vague, different languages apply it a little bit different. In Python they form a namespace to pull functions from (using import), in Java they have form of packages. C doesn't have modules. C++ doesn't have modules yet, there is ongoing work to add them - and it's going to be significant change in how we arrange code in larger projects. Modules are not libraries.

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

Answer:

Much programming advice and "best practices" comes down to the question of managing complexity. Or to put it plainly: How do we write and manage a large complex program without being overwhelmed. The solution is (like with most large problems) to split it into smaller, more manageable pieces.

Each variable is a bit of complexity, but a global variable adds complexity to the whole program (because it may have effects all over the program, hence the name), while a local variable adds complexity only in a single isolated unit, the function.

The worst fear of a developer is to have a program where a change in a single function causes a totally different part of the program to fail. Each global variable increases this risk.

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** is one endpoint of a **two way** communication link between two programs running on the network. The socket mechanism provides a means of

inter-process communication (IPC) by establishing named contact points between which the communication take place.
Like ‘Pipe’ is used to create pipes and sockets is created using **‘socket’** system call. The socket provides bidirectional **FIFO** Communication facility over the network. A socket connecting to the network is created at each end of the communication. Each socket has a specific address. This address is composed of an IP address and a port number.
Socket are generally employed in client server applications. The server creates a socket, attaches it to a network port addresses then waits for the client to contact it. The client creates a socket and then attempts to connect to the server socket. When the connection is established, transfer of data takes place.

Question 5.4: Why is relevant to have the IPv4 address of remote server? Explain what is Domain Name System (DNS)?

Answer:

Domain Name System (DNS):

The Domain Name System (DNS) is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities. Most prominently, it translates more readily memorized domain names to the numerical IP addresses needed for locating and identifying computer services and devices with the underlying network protocols.

The Domain Name System also specifies the technical functionality of the database service that is at its core. It defines the DNS protocol, a detailed specification of the data structures and data communication exchanges used in the DNS, as part of the Internet Protocol Suite.