

Lab-Report

Report No:02

Course title: Computer Networks Lab

Lab Report Name: Programing with Python

Date of Performance: 03-01-21

Date of Submission:04-02-21

Submitted by

Name: Nusrat Jahan Jui

ID:IT-18039

3rd year 2nd semester

Session: 2017-2018

Dept. of ICT

MBSTU.

Submitted to

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

1.Objective:

The objective of the lab 2 is:

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

2.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.

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

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. The protocol builds up a connection prior to data transfer using a system called a three-way handshake. This is a way for the two ends of the communication to acknowledge the request and agree upon a method of ensuring data reliability. After the data has been sent, the connection is torn down using a similar four-way handshake. TCP is the protocol of choice for many of the is safe to say that the internet we know today would not be here with 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 Page | 16 SDN-Labs machine to many real-time clients. This makes it ideal for VOIP, games, and other applications that cannot afford delays.

3.Methodology:

Defining functions: Functions are defined using the def keyword. After this keyword comes an identifier name for the function, followed by a pair of parentheses which may enclose some names of variables, and by the final colon that ends the line.

```
def XX_YY(variable1, variable2):  
    # block belonging to the function  
    # End of function
```

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
```

4.Exercises

Section 4.1: Python function variables and modules.

- **Exercise 4.1.1:** Create a python project using with SDN_LAB

```
number_gussing_game.py x
1  import random
2
3  print('You have 2 chances....Best of luck!!')
4  for i in range(2):
5      guess_number=int(input('Enter a number between 1 to 10:'))
6      random_number=random.randint(1,10)
7
8      if random_number==guess_number:
9          print('you won.....*.*')
10         break
11     else:
12         print("didn't match,try again :)")
13         print('Number was',random_number)
```

```
number_gussing_game x
"C:\Program Files (x86)\Python38-32\python.exe" C:/Users/Admin/PycharmProjects/SBN_LAB_01/number_gussing_game.py
You have 2 chances....Best of luck!!
Enter a number between 1 to 10:6
you won.....*.*

Process finished with exit code 0
```

```
number_gussing_game x
"C:\Program Files (x86)\Python38-32\python.exe" C:/Users/Admin/PycharmProjects/SBN_LAB_01/number_gussing_game.py
You have 2 chances....Best of luck!!
Enter a number between 1 to 10:3
didn't match,try again :)
Number was 6
Enter a number between 1 to 10:6
you won.....*.*
```

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

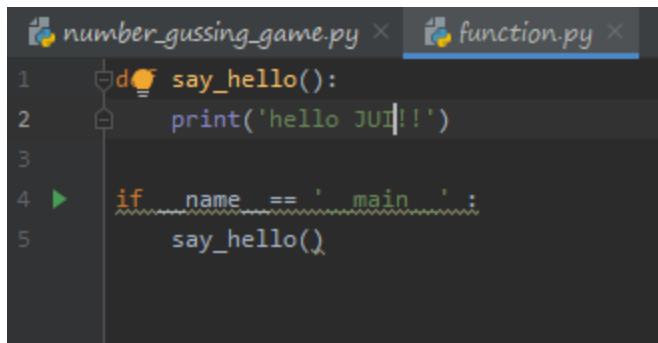
Create python scrip using the syntax provided below.

```
def say_hello():
    # block belonging to the function
    print('hello world')
    # End of function

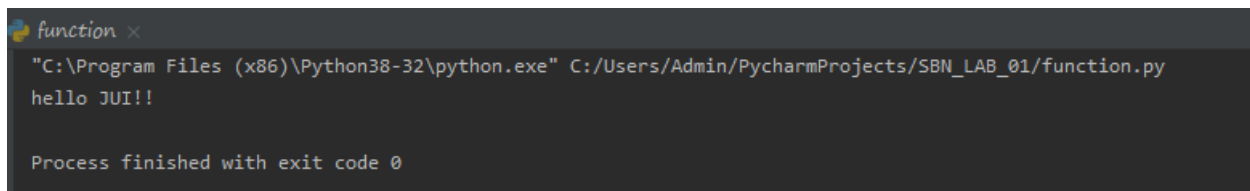
if __name__ == '__main__':
    say_hello() # call the function
```

Which is the output of this function? Does the function need any parameter?

Answer:



```
number_guessing_game.py x function.py x
1 def say_hello():
2     print('hello JUI!!')
3
4 if __name__ == '__main__':
5     say_hello()
```



```
function x
"C:\Program Files (x86)\Python38-32\python.exe" C:/Users/Admin/PycharmProjects/SBN_LAB_01/function.py
hello JUI!!

Process finished with exit code 0
```

No, The function doesn't need any parameter

• **Exercise 4.1.3:** Python function (save as function_2.py) Create python scrip using the syntax provided below.

```
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)

```

Which is the output of this function? Does the function need any parameter?

Answer:

The screenshot shows a PyCharm IDE with three tabs: 'NumberGuess.py', 'function.py', and 'function_2.py'. The 'function_2.py' tab is active, displaying the following code:

```

1 def print_max(a,b):
2     if a>b:
3         print(a,'is maximum')
4     elif a==b:
5         print(a,'is equal to',b)
6     else:
7         print(b,'is maximum')
8 if __name__ == '__main__':
9     pass
10    print_max(3,4)
11    #directly pass literal values
12    x=5
13    y=7
14    print_max(x,y)
15

```

Below the code editor, the 'Run' window shows the execution output:

```

Run: C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/NmberGuessingGame/function_2.py
4 is maximum
7 is maximum
Process finished with exit code 0

```

No, The function doesn't need any parameter.

• **Exercise 4.1.4:** Local variable (save as function_local.py) Create python scrip using the syntax provided below.

```

x = 50

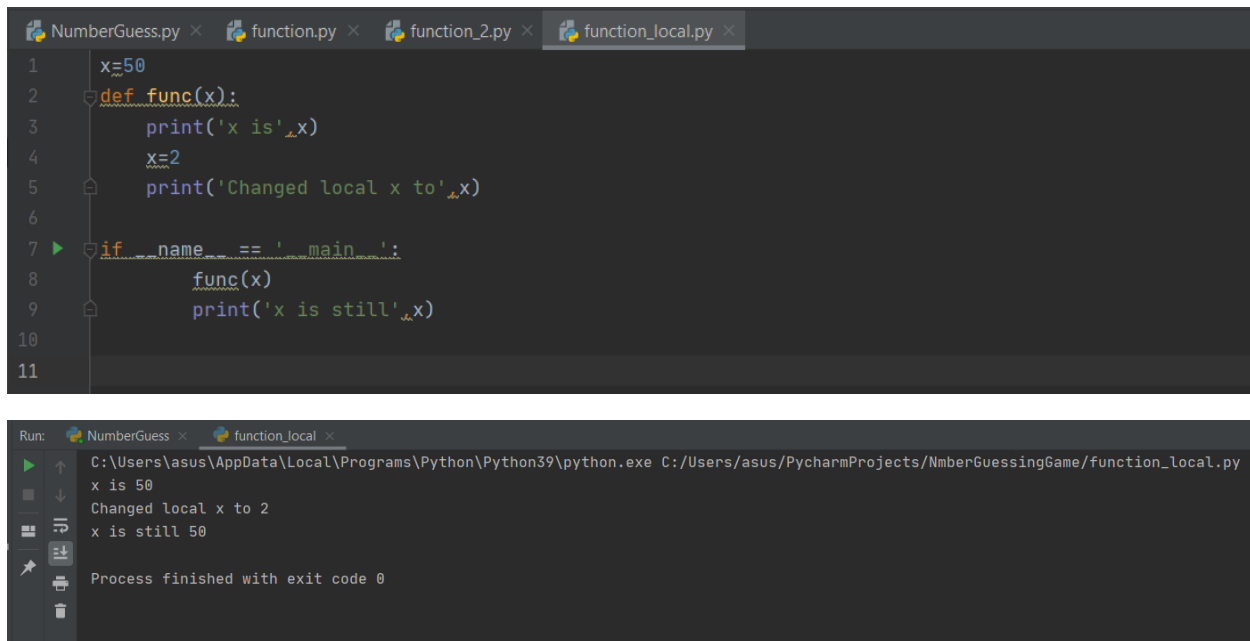
def func(x):
    print('x is', x)
    x = 2
    print('Changed local x to', x)

if __name__ == '__main__':
    func(x)
    print('x is still', x)

```

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

Answer:



The screenshot shows a Python IDE with four tabs: NumberGuess.py, function.py, function_2.py, and function_local.py. The active tab is function_local.py, which contains the following code:

```
1 x=50
2 def func(x):
3     print('x is',x)
4     x=2
5     print('Changed local x to',x)
6
7 if __name__ == '__main__':
8     func(x)
9     print('x is still',x)
10
11
```

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

```
Run: C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/NmberGuessingGame/function_local.py
x is 50
Changed local x to 2
x is still 50
Process finished with exit code 0
```

The final value of variable x is 50.

The first time that we print the value of the name x with the first line in the function's body, Python uses the value of the parameter declared in the main block, above the function definition. Next, we assign the value 2 to x. The name x is local to our function. So, when we change the value of x in the function, the x defined in the main block remains unaffected. With the last print function call, we display the value of x as defined in the main block, thereby confirming that it is actually unaffected by the local assignment within the previously called function.

Exercise 4.1.5: Global variable (save as function_global.py) Create python scrip using the syntax provided below.


```

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)

```

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

Answer:

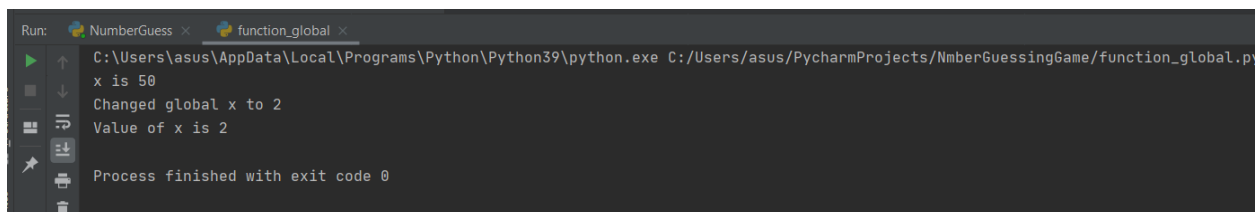


```

1  x=50
2  def func():
3      global x
4      print('x is',x)
5      x=2
6      print('Changed global x to',x)
7
8  if __name__ == '__main__':
9      func()
10     print('Value of x is',x)
11

```

Output:



```

Run: C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/NmberGuessingGame/function_global.py
x is 50
Changed global x to 2
Value of x is 2
Process finished with exit code 0

```

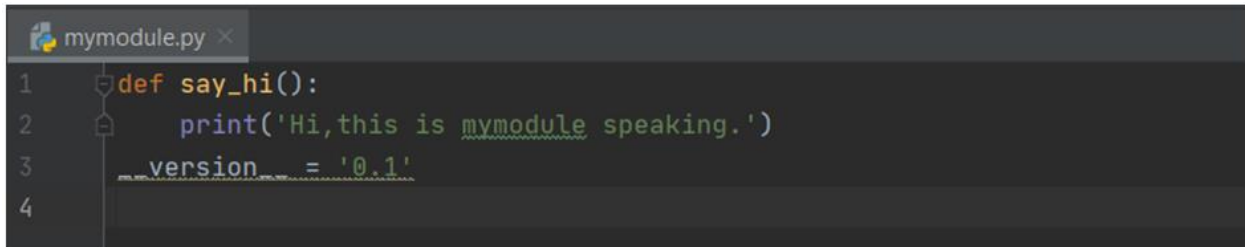
The final value of variable x is 2.

The global statement is used to declare that x is a global variable – hence, when we assign a value to x inside the function, that change is reflected when we use the value of x in the main block.

- **Exercise 4.1.6:** Python modules Create python scrip using the syntax provided below (save as mymodule.py).

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

Answer:



A screenshot of a code editor window titled 'mymodule.py'. The code inside is as follows:

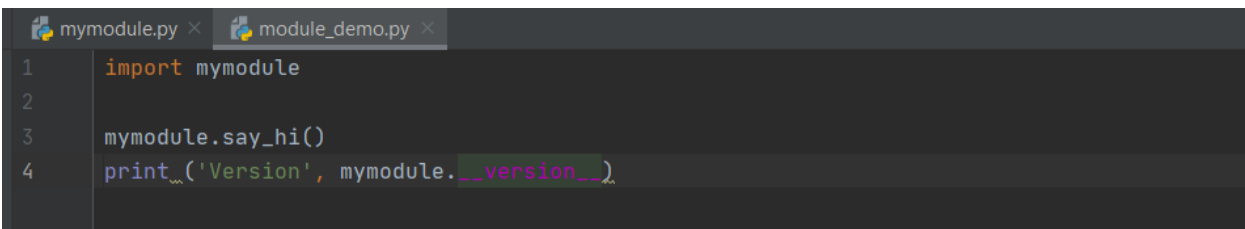
```
1 def say_hi():  
2     print('Hi, this is mymodule speaking.')  
3     __version__ = '0.1'  
4
```

Create python scrip using the syntax provided below (save as module_demo.py).

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

Run the script, which is the role of import?

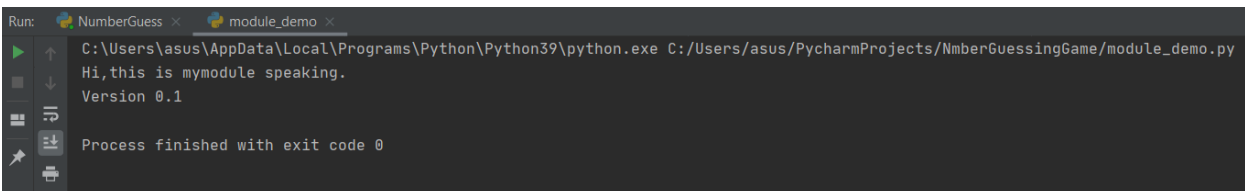
Answer:



A screenshot of a code editor window showing two files: 'mymodule.py' and 'module_demo.py'. The 'module_demo.py' file is active and contains the following code:

```
1 import mymodule  
2  
3 mymodule.say_hi()  
4 print('Version', mymodule.__version__)
```

Output:



A screenshot of a terminal window titled 'Run:'. It shows the execution of the 'module_demo.py' script. The output is as follows:

```
C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/NmberGuessingGame/module_demo.py  
Hi, this is mymodule speaking.  
Version 0.1  
  
Process finished with exit code 0
```

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.

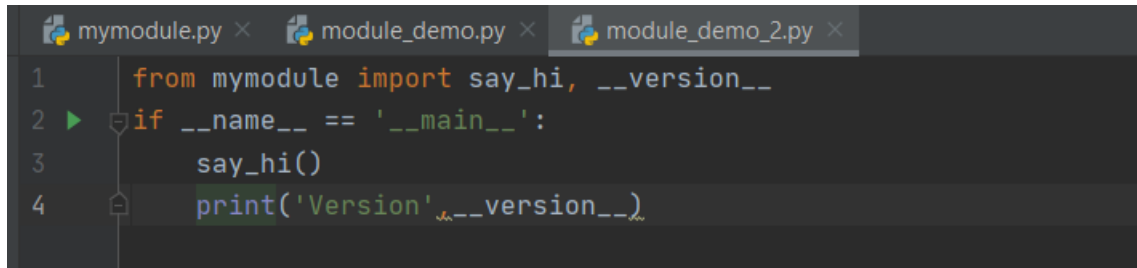
Create python scrip using the syntax provided below (save as module_demo2.py)

```
from mymodule import say_hi, __version__

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

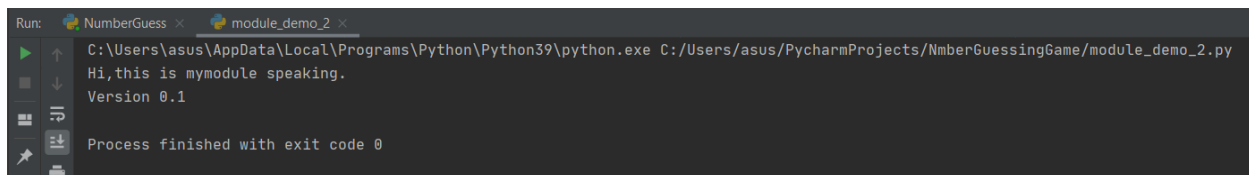
Run the script, which is the role of from, import?

Answer:



```
1 from mymodule import say_hi, __version__
2 if __name__ == '__main__':
3     say_hi()
4     print('Version', __version__)
```

Output:



```
Run: C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/NmberGuessingGame/module_demo_2.py
Hi, this is mymodule speaking.
Version 0.1
Process finished with exit code 0
```

We can import only a small part of the module i.e., only the required functions and variable names from the module instead of importing full code. When we want only specific things to be imported, we can make use of "from" keyword to import what we want.

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

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

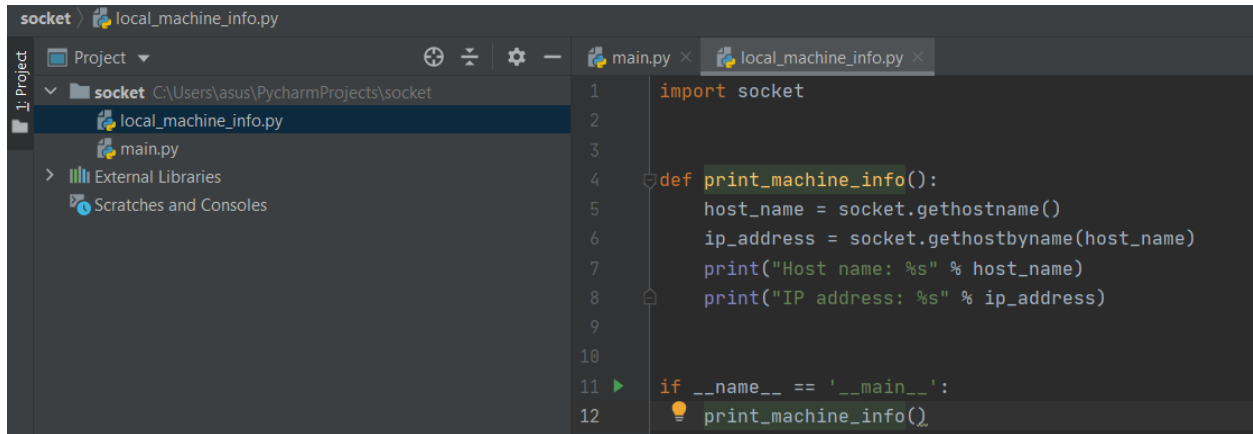
Create python scrip using the syntax provided below (save as local_machine_info.py):

```
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()
```

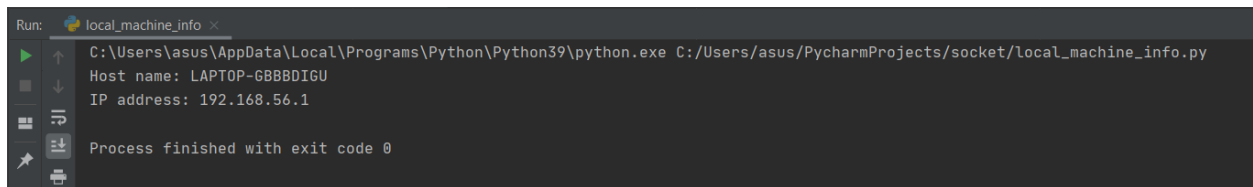
Run the script, which module the program uses? Provide two additional functions of socket?

Answer:



```
1 import socket
2
3
4 def print_machine_info():
5     host_name = socket.gethostname()
6     ip_address = socket.gethostbyname(host_name)
7     print("Host name: %s" % host_name)
8     print("IP address: %s" % ip_address)
9
10
11 if __name__ == '__main__':
12     print_machine_info()
```

Output:



```
Run: local_machine_info x
C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/socket/local_machine_info.py
Host name: LAPTOP-GBBBDIGU
IP address: 192.168.56.1
Process finished with exit code 0
```

The import socket statement imports one of Python's core networking libraries. Then, we use the two utility functions, gethostname() and gethostbyname(host_name). The first function takes no parameter and returns the current or localhost name. The second function takes a single hostname parameter and returns its IP address.

• **Exercise 4.2.2:** Retrieving a remote machine's IP address Create python script using the syntax provided below (save as remote_machine_info.py):

```

import socket

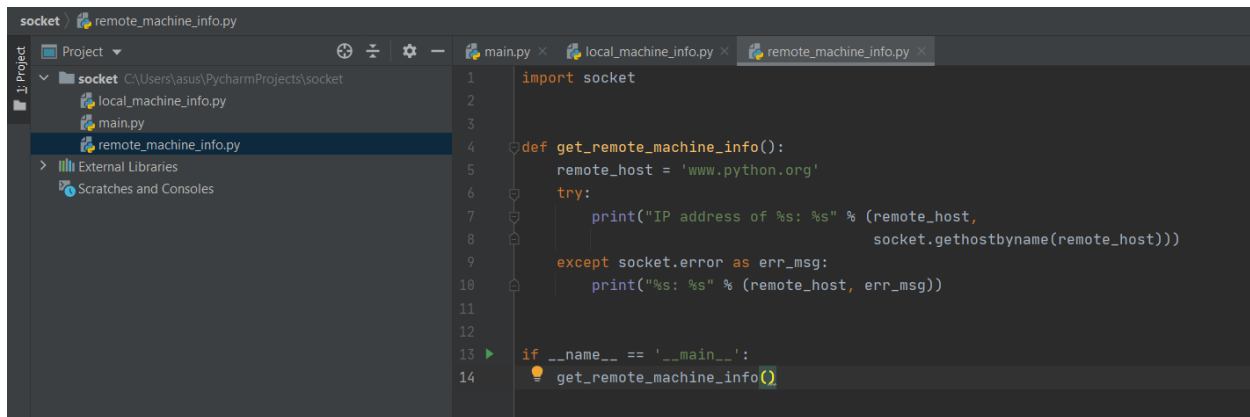
def get_remote_machine_info():
    remote_host = 'www.python.org'
    try:
        print ("    Remote host name: %s" % remote_host)
        print ("    IP address: %s" % socket.gethostbyname(remote_host))
    except socket.error as err_msg:
        print ("Error accesing %s: error number and detail %s"
              %(remote_host, err_msg))

if __name__ == '__main__':
    get_remote_machine_info()

```

Run the script, which is the output? Modify the code for getting the RMIT website info.

Answer:

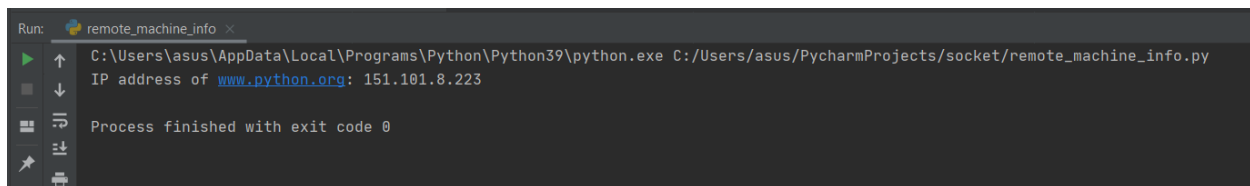


```

socket / remote_machine_info.py
Project
  socket
    local_machine_info.py
    main.py
    remote_machine_info.py
  External Libraries
  Scratches and Consoles
main.py x local_machine_info.py x remote_machine_info.py x
1 import socket
2
3
4 def get_remote_machine_info():
5     remote_host = 'www.python.org'
6     try:
7         print("IP address of %s: %s" % (remote_host,
8                                         socket.gethostbyname(remote_host)))
9     except socket.error as err_msg:
10        print("%s: %s" % (remote_host, err_msg))
11
12
13 if __name__ == '__main__':
14     get_remote_machine_info()

```

Output:



```

Run: remote_machine_info x
C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/socket/remote_machine_info.py
IP address of www.python.org: 151.101.8.223
Process finished with exit code 0

```

• **Exercise 4.2.3:** Converting an IPv4 address to different formats Create python scrip using the syntax below (save as ip4_address_conversion.py):

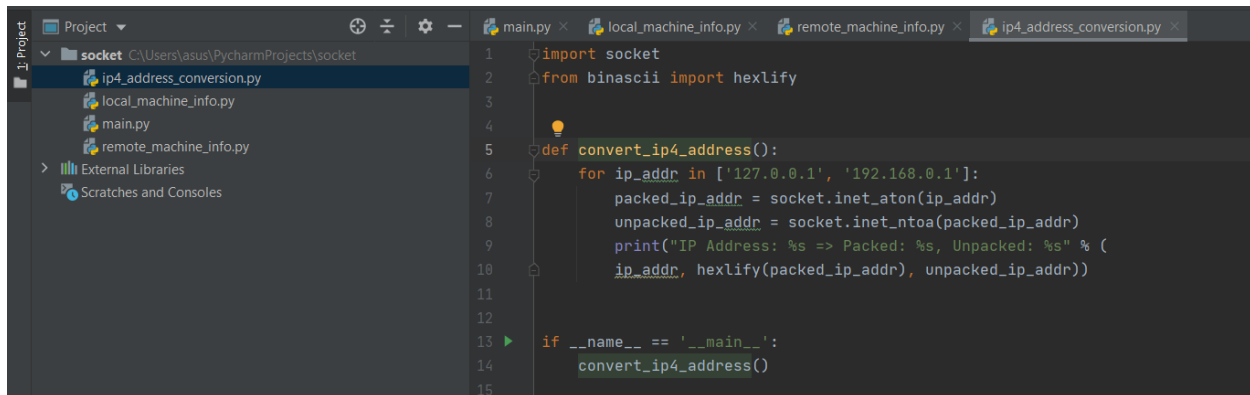
```
import socket
from binascii import hexlify

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))

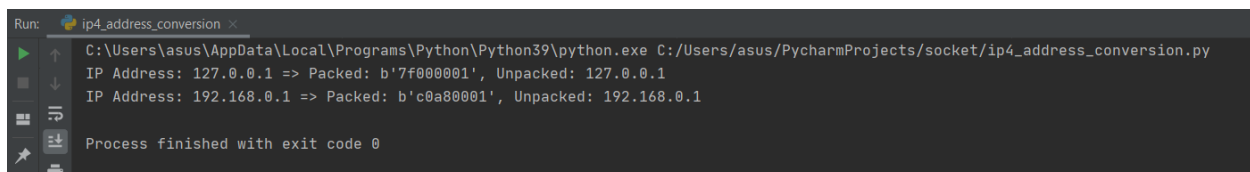
if __name__ == '__main__':
    convert_ip4_address()
```

Run the script, which is the output? How binascii works?

Answer:



Output:



In this recipe, the two IP addresses have been converted from a string to a 32-bit packed format using a for-in statement. Additionally, the Python hexlify function is called from the binascii module. This helps to represent the binary data in a hexadecimal format.

• **Exercise 4.2.4:** Finding a service name, given the port and protocol Create python scrip using the syntax below (save as finding_service_name.py):

```
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()
```

Run the script, which is the output? Modify the code for getting complete the table:

Port	Protocol Name
21	
22	
110	

Answer:

```
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()
```

Output:

```
Run: finding_service_name x
C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/socket/finding_service_name.py
Port: 80 => service name: http
Port: 25 => service name: smtp
Port: 53 => service name: domain
Process finished with exit code 0
```

- Exercise 4.2.5: Setting and getting the default socket timeout Create python scrip using the syntax below (save as socket_timeout.py):

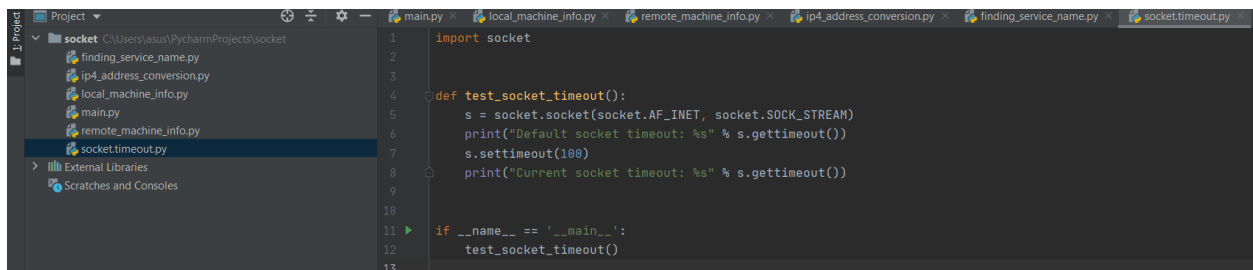
```
import socket

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()
```

Run the script, which is the role of socket timeout in real applications?

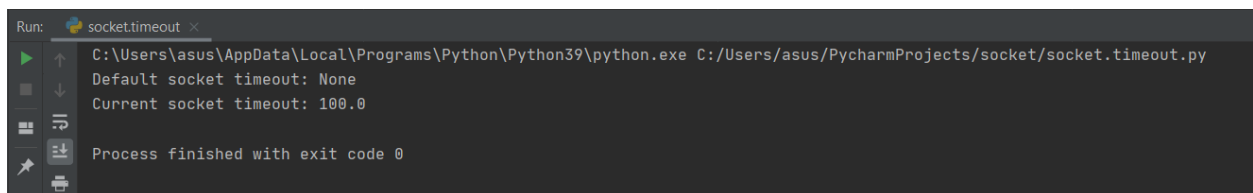
Answer:



The screenshot shows the PyCharm IDE with a project named 'socket' located at 'C:\Users\asus\PycharmProjects\socket'. The file 'socket.timeout.py' is selected in the project view. The code editor displays the following Python script:

```
1 import socket
2
3
4 def test_socket_timeout():
5     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6     print("Default socket timeout: %s" % s.gettimeout())
7     s.settimeout(100)
8     print("Current socket timeout: %s" % s.gettimeout())
9
10
11 if __name__ == '__main__':
12     test_socket_timeout()
13
```

Output:



The screenshot shows the PyCharm Run console for the 'socket.timeout.py' script. The output is as follows:

```
Run: socket.timeout x
C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/socket/socket.timeout.py
Default socket timeout: None
Current socket timeout: 100.0
Process finished with exit code 0
```

In order to avoid the annoying problem of waiting for an undetermined amount of time, sockets (which are responsible for network communication) support a timeout option which raises an error based on a time limit. Unfortunately, when developers develop their amazing network libraries, they may omit such non-obvious cases and forget to provide socket timeout settings, despite of the docs recommendation.

- **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):


```

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

```

```

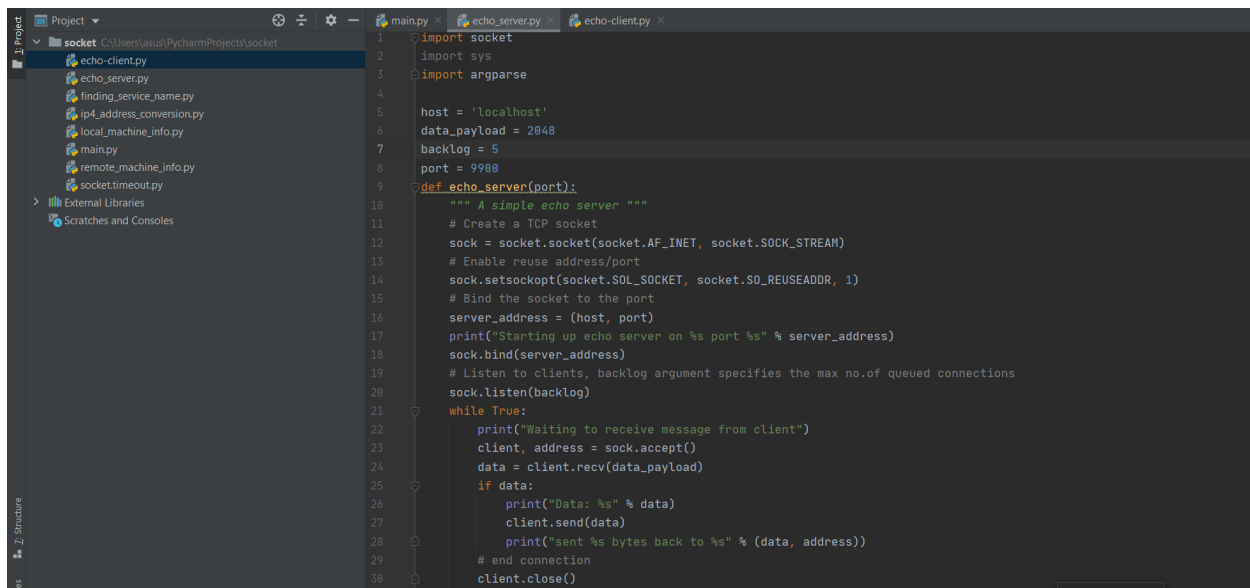
server_address = (host, port)
print ("Starting up echo server on %s port %s" %server_address)
sock.bind(server_address)
# Listen to clients, backlog argument specifies the max no. of queued
connections
sock.listen(backlog)

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()

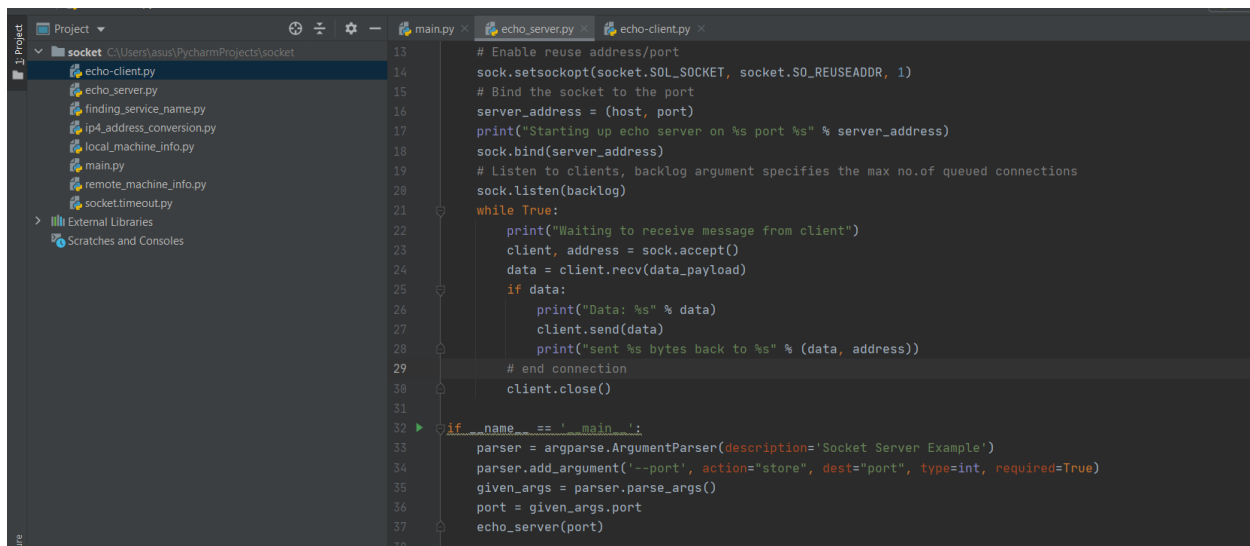
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int,
required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)

```

Answer:

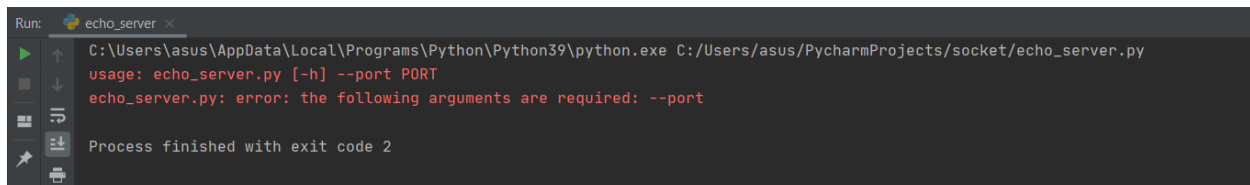


```
1 import socket
2 import sys
3 import argparse
4
5 host = 'localhost'
6 data_payload = 2048
7 backlog = 5
8 port = 9900
9
10 def def_echo_server(port):
11     """ A simple echo server """
12     # Create a TCP socket
13     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14     # Enable reuse address/port
15     sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
16     # Bind the socket to the port
17     server_address = (host, port)
18     print("Starting up echo server on %s port %s" % server_address)
19     sock.bind(server_address)
20     # Listen to clients, backlog argument specifies the max no.of queued connections
21     sock.listen(backlog)
22     while True:
23         print("Waiting to receive message from client")
24         client, address = sock.accept()
25         data = client.recv(data_payload)
26         if data:
27             print("Data: %s" % data)
28             client.send(data)
29             print("sent %s bytes back to %s" % (data, address))
30         # end connection
31         client.close()
```



```
13 # Enable reuse address/port
14 sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
15 # Bind the socket to the port
16 server_address = (host, port)
17 print("Starting up echo server on %s port %s" % server_address)
18 sock.bind(server_address)
19 # Listen to clients, backlog argument specifies the max no.of queued connections
20 sock.listen(backlog)
21 while True:
22     print("Waiting to receive message from client")
23     client, address = sock.accept()
24     data = client.recv(data_payload)
25     if data:
26         print("Data: %s" % data)
27         client.send(data)
28         print("sent %s bytes back to %s" % (data, address))
29     # end connection
30     client.close()
31
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)
```

Output:



```
Run: echo_server
C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/socket/echo_server.py
usage: echo_server.py [-h] --port PORT
echo_server.py: error: the following arguments are required: --port
Process finished with exit code 2
```

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

```
#!/usr/bin/env python

import socket
import sys
import argparse
import codecs

from codecs import encode, decode

host = 'localhost'

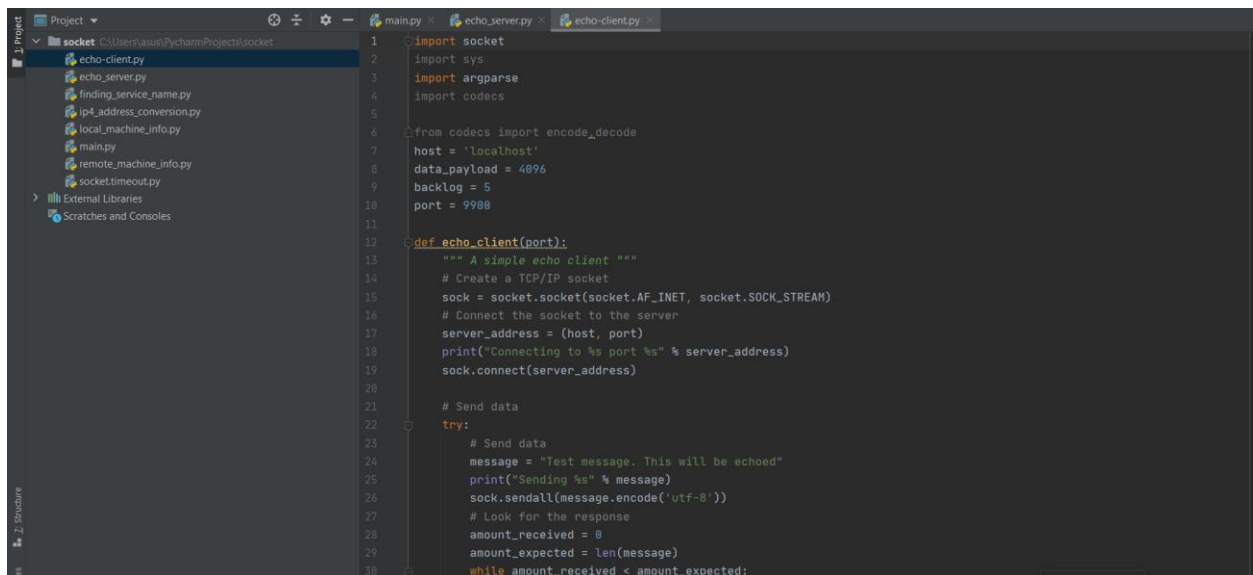
def echo_client(port):
    """ A simple echo client """
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Connect the socket to the server
    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    sock.connect(server_address)
    # 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()

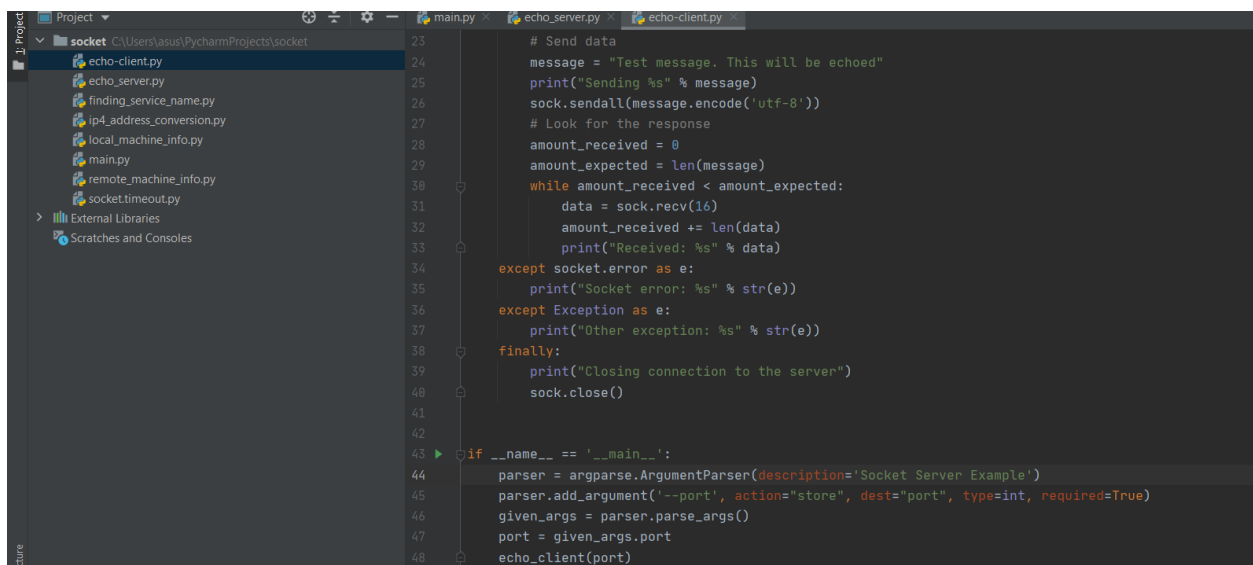
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int,
required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```

- Run the script, which is the output?

Answer:

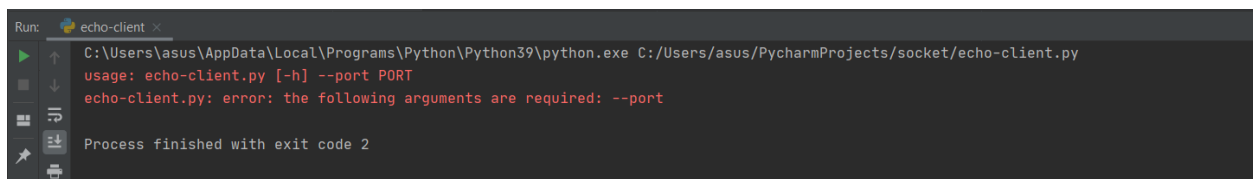


```
1 import socket
2 import sys
3 import argparse
4 import codecs
5
6 from codecs import encode_decode
7
8 host = 'localhost'
9 data_payload = 4096
10 backlog = 5
11 port = 9988
12
13 def echo_client(port):
14     """ A simple echo client """
15     # Create a TCP/IP socket
16     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17     # Connect the socket to the server
18     server_address = (host, port)
19     print("Connecting to %s port %s" % server_address)
20     sock.connect(server_address)
21
22     # Send data
23     try:
24         # Send data
25         message = "Test message. This will be echoed"
26         print("Sending %s" % message)
27         sock.sendall(message.encode('utf-8'))
28         # Look for the response
29         amount_received = 0
30         amount_expected = len(message)
31         while amount_received < amount_expected:
```



```
23
24 # Send data
25 message = "Test message. This will be echoed"
26 print("Sending %s" % message)
27 sock.sendall(message.encode('utf-8'))
28 # Look for the response
29 amount_received = 0
30 amount_expected = len(message)
31 while amount_received < amount_expected:
32     data = sock.recv(16)
33     amount_received += len(data)
34     print("Received: %s" % data)
35 except socket.error as e:
36     print("Socket error: %s" % str(e))
37 except Exception as e:
38     print("Other exception: %s" % str(e))
39 finally:
40     print("Closing connection to the server")
41     sock.close()
42
43 if __name__ == '__main__':
44     parser = argparse.ArgumentParser(description='Socket Server Example')
45     parser.add_argument('--port', action='store', dest='port', type=int, required=True)
46     given_args = parser.parse_args()
47     port = given_args.port
48     echo_server(port)
```

Output:



```
Run: echo-client x
C:\Users\asus\AppData\Local\Programs\Python\Python39\python.exe C:/Users/asus/PycharmProjects/socket/echo-client.py
usage: echo-client.py [-h] --port PORT
echo-client.py: error: the following arguments are required: --port
Process finished with exit code 2
```

- What you need to do for running the program?

Answer:

We need to install Python on our machine before we start coding. Python comes preinstalled in most of the Linux distributions. Then we need to install pycharm for coding and running the program.

- Which program you need to run first client of server?

Answer:

We need to run Python socket server program executes at first and wait for any request. Python socket client program will initiate the conversation at first. Then server program will response accordingly to client requests

- Explain how the program works?

Answer:

The program is not working perfectly. Though I have used port 9900, it gives an error.

- What you need to do for communicating with another server in the classroom?

Answer:

We need to use socket to connect another server for communicating with another server in the classroom.

5. 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 us many built-in functions like print(), etc. but we can also create our 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.

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

Answer:

Global variables are declared outside any function, and they can be accessed (used) on any function in the program. Local variables are declared inside a function, and can be used only inside that function.

• **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.

In computer network interface, usually chosen at random from the available non-well-known ports.

• **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.