PRACTICAL 01

Aim:

Design a distributed application using socket in which a client program sends some numbers to the server. Server reads these number and performs some computations e.g. basic arithmetic operations, comparisons, any scientific computation of these numbers or any other computation. Use any programming language that permits socket primitives to design this application.

Code:

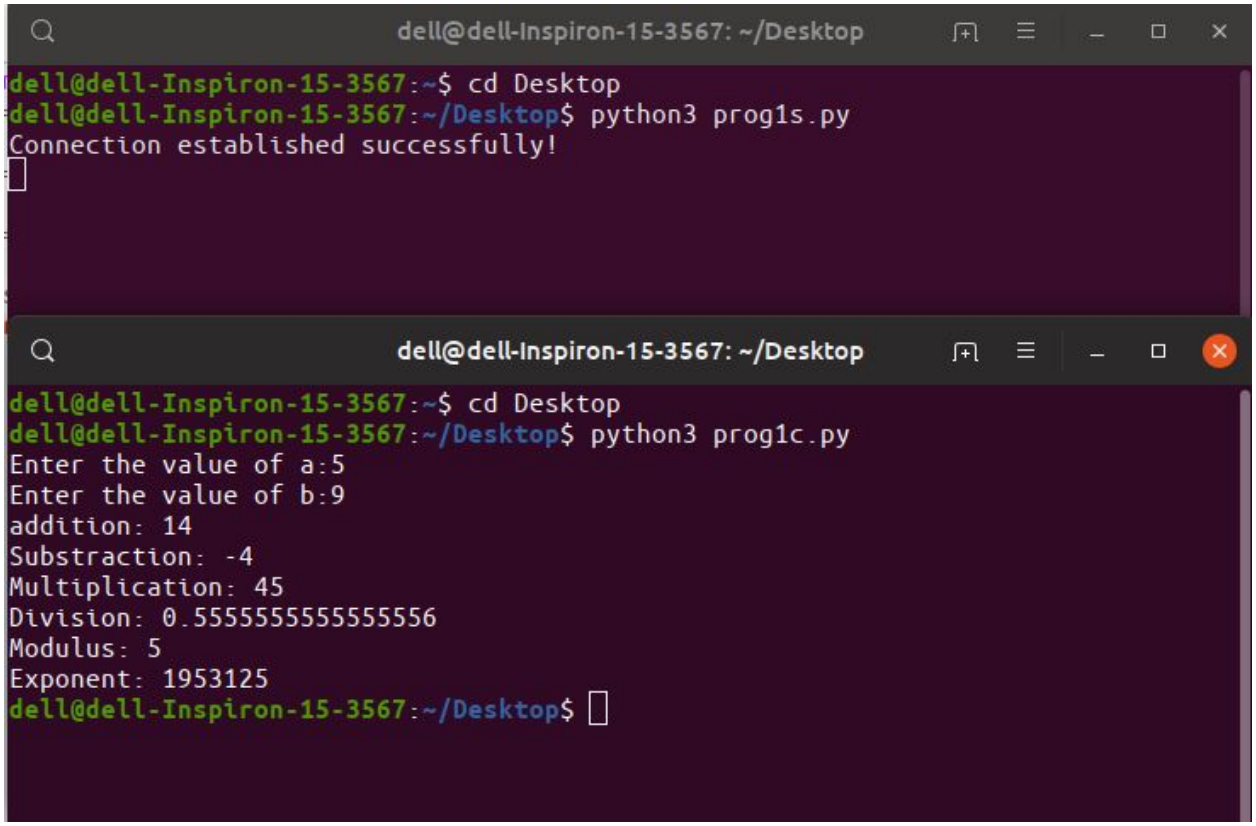
1. Server Program:

```
import socket
s= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((socket.gethostname() ,1024))
s.listen(5)
while True:
  clt, adr = s.accept()
  print(f"Connection established successfully!")
  a=clt.recv(1024).decode("utf-8")
  b=clt.recv(1024).decode("utf-8")
  add=int(a)+int(b)
  sub=int(a)-int(b)
  mul=int(a)*int(b)
  div=int(a)/int(b)
  mod=int(a)%int(b)
  exp=int(a)**int(b)
  clt.send(("addition: "+str(add)+"\nSubstraction:
"+str(sub)+"\nMultiplication: "+str(mul)+"\nDivision:
"+str(div)+"\nModulus: "+str(mod)+"\nExponent: "+str(exp)).encode())
```

2. Client Program:

```
import socket
s= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((socket.gethostname() ,1024))
a=input("Enter the value of a:")
s.send(a.encode("utf-8"))
b=input("Enter the value of b:")
s.send(b.encode("utf-8"))
msg=s.recv(1024)
print(msg.decode("utf-8"))
```

Output:



PRACTICAL 02

Aim:

Design a distributed client server application using socket.The server offers many functionalities using threads or multiple processes.

Client program :

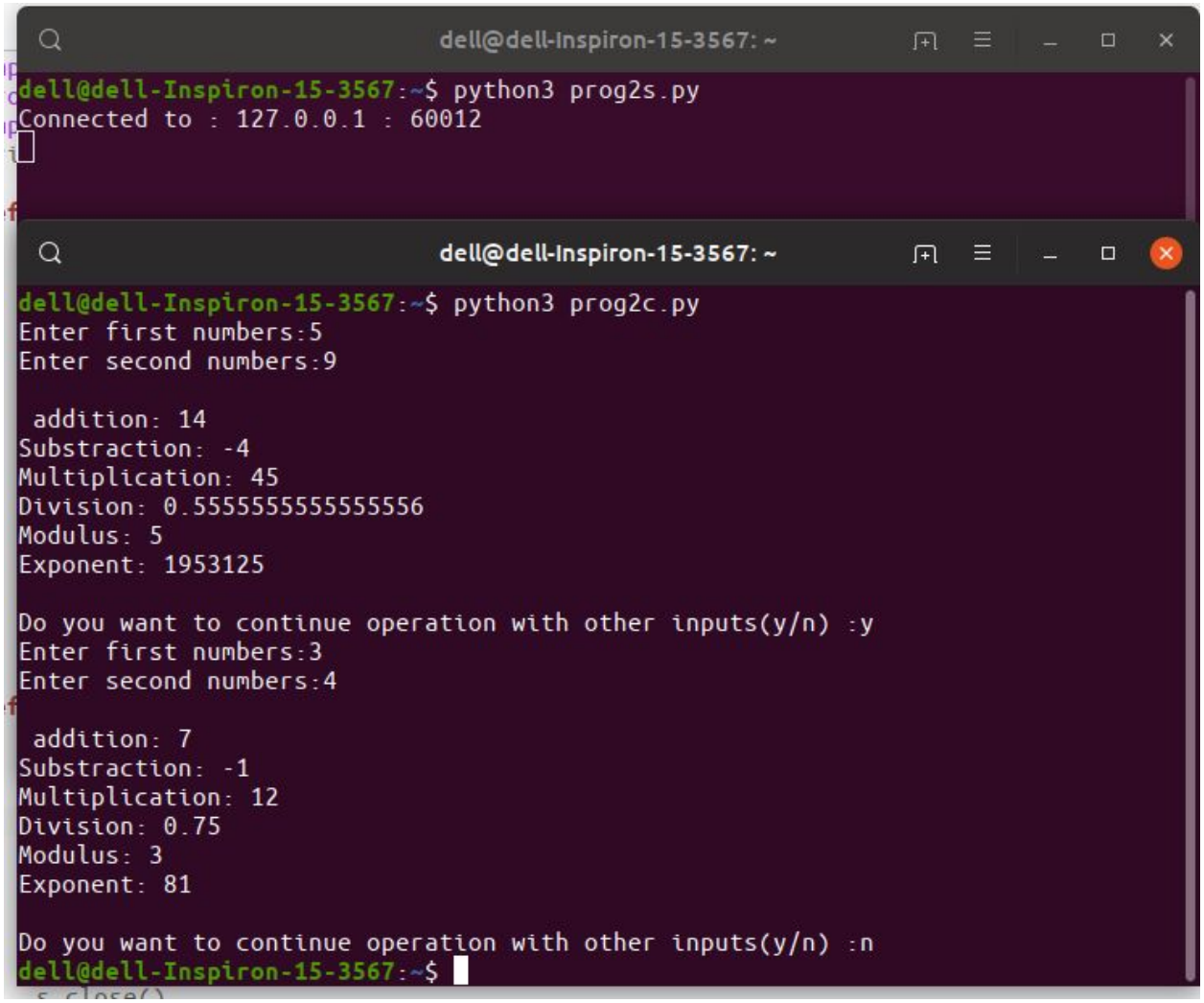
```
import socket
def Main():
        s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect((socket.gethostname() ,1024))
        while True:

        num1 =input("Enter first numbers:")
        s.send(num1.encode('ascii'))
        num2 =input("Enter second numbers:")
        s.send(num2.encode('ascii'))
        data = s.recv(1024)
        print('\n',str(data.decode('ascii')))
        ans = input('\nDo you want to continue operation with other
inputs(y/n) :')
        if ans == 'y':
        continue
        else:
        break
        s.close()

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

Server Program:

```
import socket
from _thread import *
import threading
print_lock = threading.Lock()

def threaded(c):
	while True:

	num1 = c.recv(1024)
	num2 = c.recv(1024)
	if not (num1 or num2):
	print_lock.release()
	break

	add=int(num1)+int(num2)
	sub=int(num1)-int(num2)
	mul=int(num1)*int(num2)
	div=int(num1)/int(num2)
	mod=int(num1)%int(num2)
	exp=int(num1)**int(num2)
	c.send(("addition: "+str(add)+"\nSubstraction:
"+str(sub)+"\nMultiplication: "+str(mul)+"\nDivision: "+str(div)+"\nModulus:
"+str(mod)+"\nExponent: "+str(exp)).encode())
	c.close()
def Main():
	s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
	s.bind((socket.gethostname(),1024))
	s.listen(5)

	while True:
	c, addr = s.accept()
```

```
        print_lock.acquire()
        print('Connected to :', addr[0], ':', addr[1])
        start_new_thread(threaded, (c,))
        s.close()
if __name__ == '__main__':
        Main()
```

Output:



PRACTICAL 05

1.ArCompt.java

```
import java.rmi.*;
import java.util.*;
import java.lang.*;
// Creating an Interface
public interface ArCompt
	extends java.rmi.Remote {

	// Declaring the method
	public int factorial(int x) throws java.rmi.RemoteException;
	public double calfun(double x,int n) throws java.rmi.RemoteException;


}
```

2.ArComptImpl.java

```
import java.util.Scanner;
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.lang.*;
public class ArComptImpl
	extends java.rmi.server.UnicastRemoteObject
	implements ArCompt {

	public ArComptImpl ()
	throws java.rmi.RemoteException
	{
	super();
	}
	public int  factorial(int x)
	throws java.rmi.RemoteException
	{   if (x==0){return 1;}
	else{
	return x*factorial(x-1);
```

```
        }
        }
        public double calfun(double x1 , int y) throws java.rmi.RemoteException
        {   double radians = Math.toRadians(x1);
        if(y==1){return Math.sin(radians);}
        else if(y==2){return Math.cos(radians);}
        else if(y==3){return Math.tan(radians);}
        else{return 0;}
        }
}
```

3.ArComptClient.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.io.*;
import java.lang.*;
import java.net.*;
public class ArComptClient {
        public static void main(String[] args)
        {
        Scanner s= new Scanner(System.in);
        try {

        ArCompt c = (ArCompt)Naming.lookup("rmi://localhost/ArCompt");
        System.out.print("\nEnter the Number to find factorial:");
        int num=s.nextInt();
        System.out.print("\nFactorial: "+ c.factorial(num));
        System.out.print("\nEnter the value (in degree) to find sin():");
        double a=s.nextDouble();
        System.out.print("\nSin(): "+ c.calfun(a,1));
        System.out.print("\nEnter the value (in degree) to find cos():");
        double b=s.nextDouble();
        System.out.print("\ncos(): "+ c.calfun(b, 2));
        System.out.print("\nEnter the value (in degree) to find tan():");
        double t=s.nextDouble();
        System.out.print("\ntan(): "+ c.calfun(t, 3));
        }
```

```
catch(Exception e){System.out.print(e);}
        }

}
```

4.ArComptServer.java

```
import java.rmi.*;
import java.rmi.server.*;
public class ArComptServer {
        public static void main(String[] args)
        {
        try{
        ArComptImpl stub= new ArComptImpl();
        Naming.rebind("rmi:///ArCompt",stub);
}
catch(Exception e){System.out.print(e);}
        }
}
```

Output:

```
dell@dell-Inspiron-15-3567: ~/Desktop
dell@dell-Inspiron-15-3567:~$ cd Desktop
dell@dell-Inspiron-15-3567:~/Desktop$ javac ArCompt.java
dell@dell-Inspiron-15-3567:~/Desktop$ javac ArComptImpl.java
dell@dell-Inspiron-15-3567:~/Desktop$ javac ArComptServer.java
dell@dell-Inspiron-15-3567:~/Desktop$ javac ArComptClient.java
dell@dell-Inspiron-15-3567:~/Desktop$ rmic ArComptImpl
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
dell@dell-Inspiron-15-3567:~/Desktop$ rmiregistry
```

```
dell@dell-Inspiron-15-3567: ~/Desktop
dell@dell-Inspiron-15-3567:~$ cd Desktop
dell@dell-Inspiron-15-3567:~/Desktop$ javac ArComptServer.java
dell@dell-Inspiron-15-3567:~/Desktop$ java ArComptServer
```

```
dell@dell-Inspiron-15-3567: ~/Desktop
dell@dell-Inspiron-15-3567:~$ cd Desktop
dell@dell-Inspiron-15-3567:~/Desktop$ javac ArComptClient.java
dell@dell-Inspiron-15-3567:~/Desktop$ java ArComptClient

Enter the Number to find factorial:3

Factorial: 6
Enter the value (in degree) to find sin():30

Sin(): 0.49999999999999994
Enter the value (in degree) to find cos():60

cos(): 0.5000000000000001
Enter the value (in degree) to find tan():45

tan(): 0.9999999999999999dell@dell-Inspiron-15-3567:~/Desktop$
```