COM302 - COMPUTER NETWORKS

FINAL COURSE PROJECT

Group 1

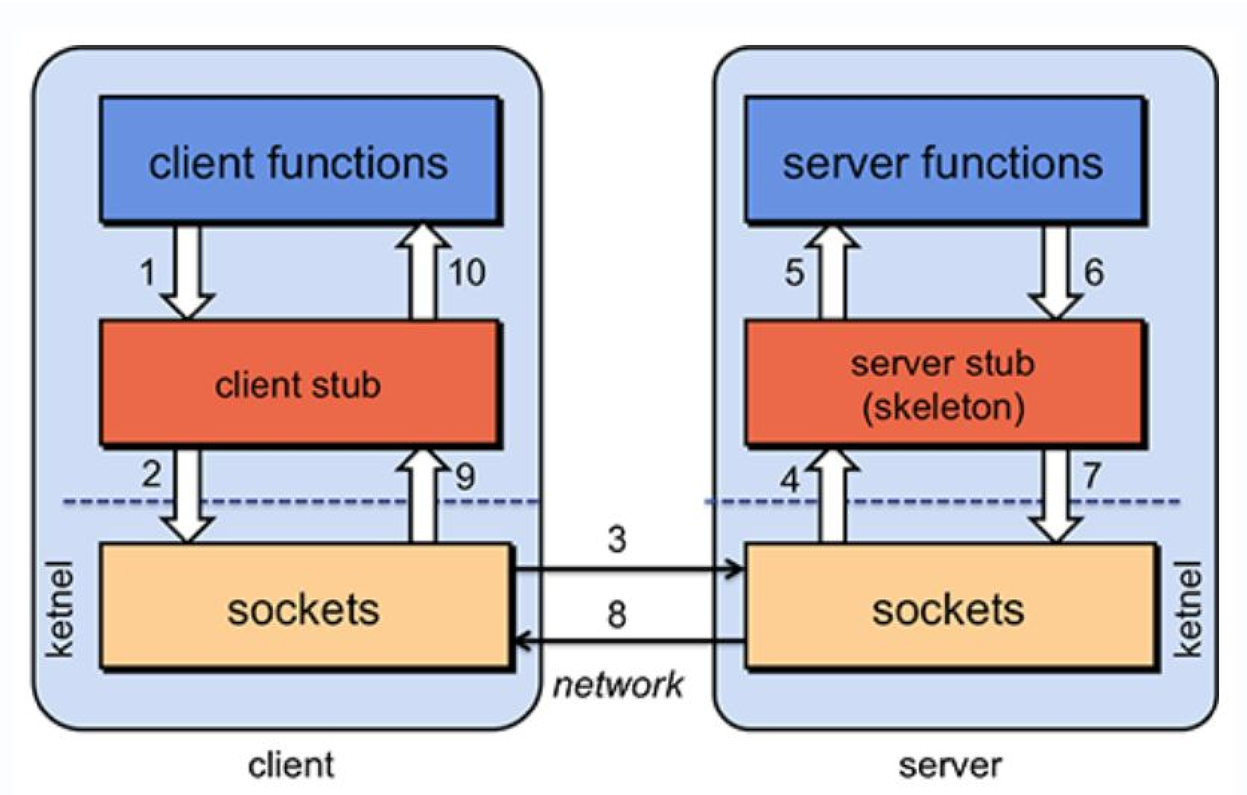
COE18B024	Hrishikesh Rajesh Menon
COE18B028	Katepalli Naga Sai Bhargav
COE18B056	Thigulla Vamsi Krishna
COE18B065	Srinivasan R Sharma
CED18I039	Paleti Krishnasai
CED18I056	Darshan VSS

Remote Procedure Call

Remote Procedure Call (RPC) system enables you to call a function available on a remote server using the same syntax which is used when calling a function in a local library. This is useful in two situations.

- You can utilize the processing power from multiple machines using rpc without changing the code for making the call to the programs located in the remote systems.
- The data needed for the processing is available only in the remote system.

So in python we can treat one machine as a server and another machine as a client which will make a call to the server to run the remote procedure.



Steps in executing a remote procedure call

1. The client calls a local procedure, called the **client stub**. To the client process, this appears to be the actual procedure, because it is a regular local procedure. It just does something different since the real procedure is on the server. The client stub packages the parameters to the remote procedure (this may involve converting them to a standard format) and builds one or more network messages. The packaging of arguments into a network message is called **marshaling** and requires **serializing** all the data elements into a flat array-of-bytes format.
2. Network messages are sent by the client stub to the remote system (via a system call to the local kernel using *sockets* interfaces).
3. Network messages are transferred by the kernel to the remote system via some protocol (either connectionless or connection-oriented).
4. A **server stub**, sometimes called the **skeleton**, receives the messages on the server. It **unmarshals** the arguments from the messages and, if necessary, converts them from a standard network format into a machine-specific form.
5. The server stub calls the server function (which, to the client, is the remote procedure), passing it the arguments that it received from the client.
6. When the server function is finished, it returns to the server stub with its return values.
7. The server stub converts the return values, if necessary, and marshals them into one or more network messages to send to the client stub.
8. Messages get sent back across the network to the client stub.
9. The client stub reads the messages from the local kernel.
10. The client stub then returns the results to the client function, converting them from the network representation to a local one if necessary. The client code then continues its execution.

Overview :

The Implementation that has been done is using a monte carlo estimation of PI , which is cpu intensive .

The computation is on the server stub which is hosted on Google Cloud , hence the clients need not waste cpu power to compute the intensive calculations.

The estimated value of PI is returned to the respective client(multi threaded) and the client uses it to compute any further calculations needed.

(demonstration has been done with the area and perimeter of a circle with a fixed radius = 6 , with which the difference can be seen with the different values of PI) .

The server and client skeletons are mostly generic with an added control over the subroutine calling through different threads to avoid data security issues.

The server function comprises of the Monte Carlo method of estimation of the value of PI , the input is sent over from the client side who inputs the number of points needed to take for the estimation. The server then computes the cpu intensive calculations and sends the output back to the client through a socket. The client then uses the received value for its continued execution.

While the server is running on a specific thread's request , that specific thread will be in *sleep* state.

The project has a graphical user interface (**GUI**) written on tkinter and integrated to the client main window.

Technologies used

- Python
- Sockets
- Tkinter

Features that this project has

- User Friendly gui
- Theme change feature in the gui
- Data security (each client is communicated via unique thread)
- Complete gui based usage .
- Efficient and clean implementation of RPC.

Challenges faced

- Data security over threads and clients.
- Complete gui based usage.

How we tackled

- Each thread is controlled with an IP mapping so that only the client that has made a request is served and served the intended data .
- Integrated the gui in the client main window with the client skeleton code as part of the gui boilerplate.

Advantages Of RPC :

- You don’t have to worry about getting a unique transport address (picking a unique port number for a socket on a machine). The server can bind to any available port and then register that port with an RPC name server. The client will contact this name server to find the port number that corresponds to the program it needs. All this will be invisible to the programmer.
- The system can be independent of transport providers. The automatically-generated server stub can make itself available over every transport provider on a system, both TCP and UDP. The client can choose dynamically and no extra programming is required since the code to send and receive messages is automatically generated.
- Applications on the client only need to know one transport address: that of the name server that is responsible for telling the application where to connect for a given set of server functions.
- The function-call model can be used instead of the send/receive (read/write) interface provided by sockets. Users don't have to deal with marshaling parameters and then parsing them out on the other side.

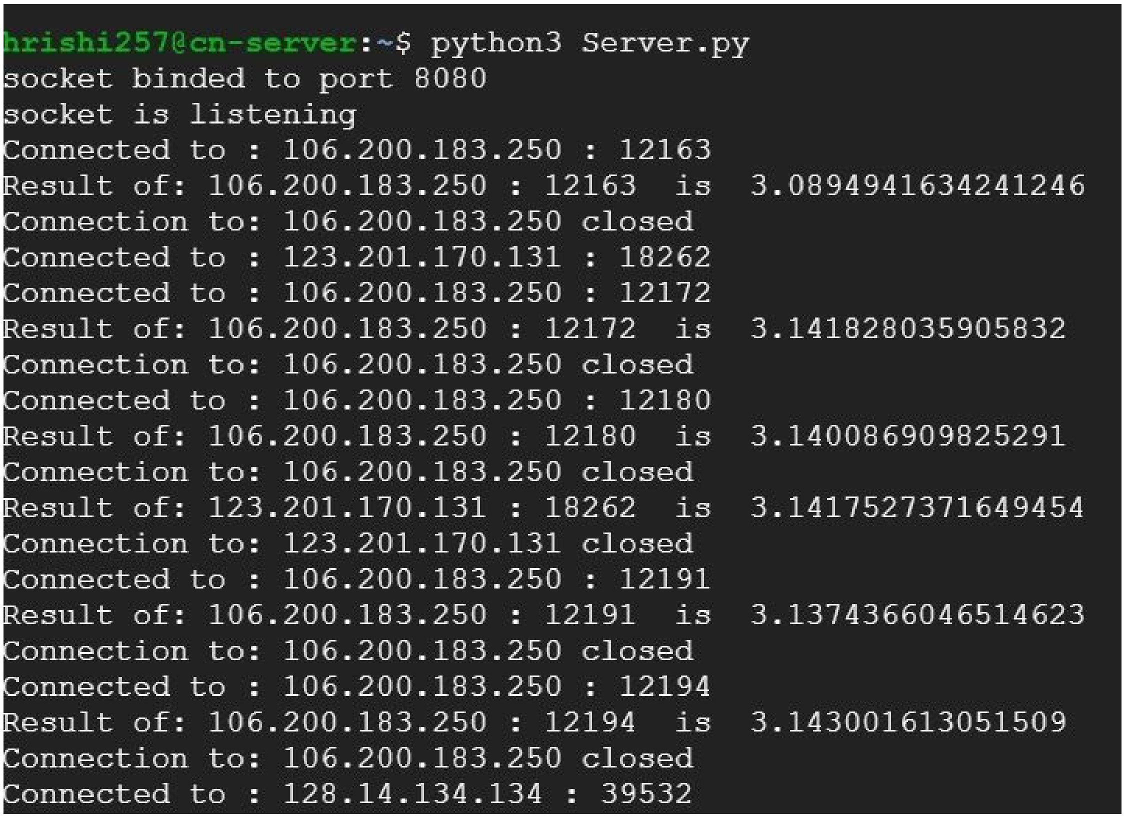
Instructions to test drive the project :

1. Start the server hosted on google cloud (for local testing change the intended IP Address and Port).
2. Run the Client.py on your system.
3. Select ***Action*** -> ***Enter a command.***
4. Input the number of points in the dialog box.
5. Press Ok.
6. The result is shown on the main window .
7. Select **Yes** to continue , **No** to stop the process.

The Code files can be accessed through the following repository

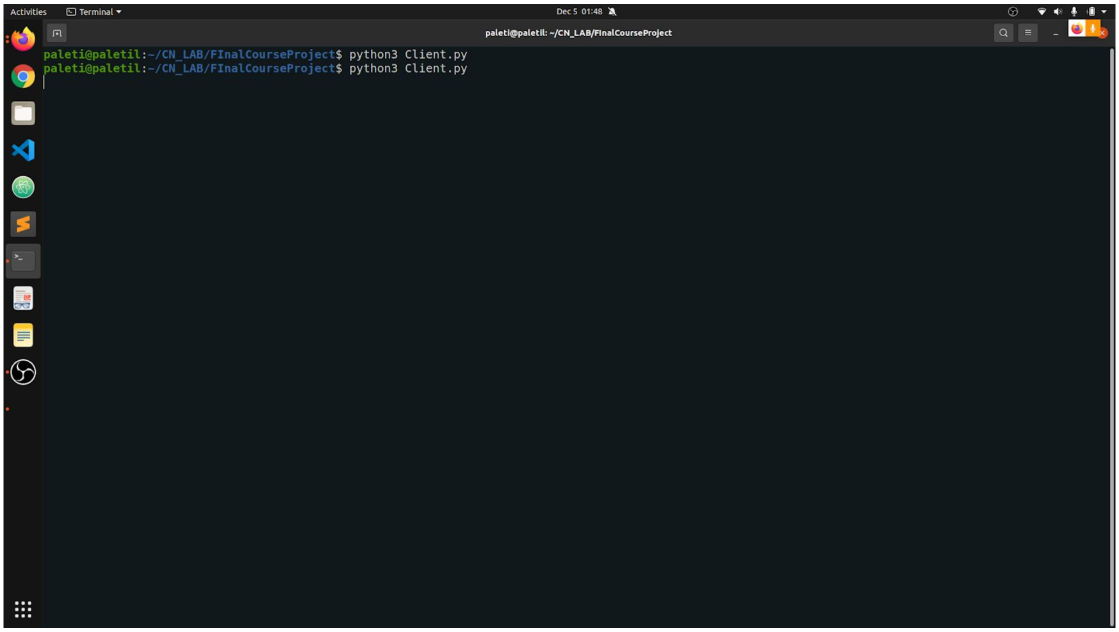
Remote-Procedural-Call

Cloud Hosted Server :

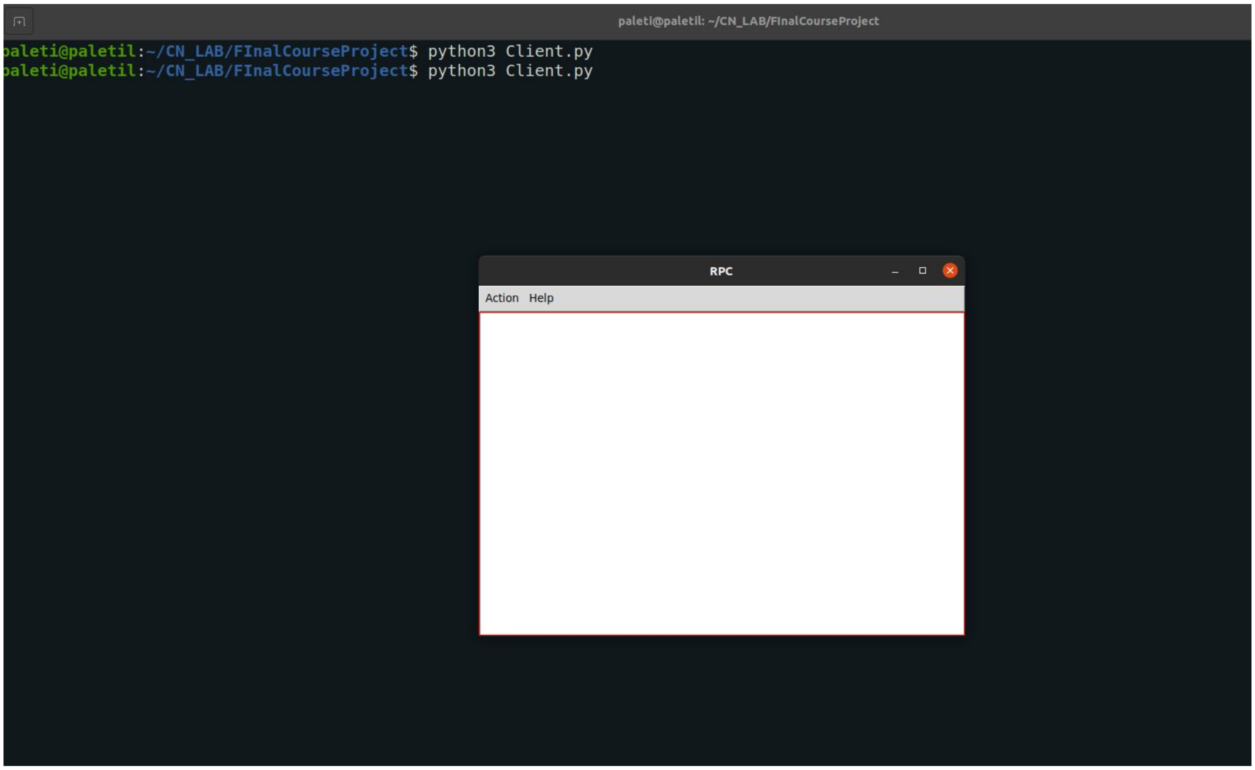


```
hrishi257@cn-server:~$ python3 Server.py
socket binded to port 8080
socket is listening
Connected to : 106.200.183.250 : 12163
Result of: 106.200.183.250 : 12163  is  3.0894941634241246
Connection to: 106.200.183.250 closed
Connected to : 123.201.170.131 : 18262
Connected to : 106.200.183.250 : 12172
Result of: 106.200.183.250 : 12172  is  3.141828035905832
Connection to: 106.200.183.250 closed
Connected to : 106.200.183.250 : 12180
Result of: 106.200.183.250 : 12180  is  3.140086909825291
Connection to: 106.200.183.250 closed
Result of: 123.201.170.131 : 18262  is  3.1417527371649454
Connection to: 123.201.170.131 closed
Connected to : 106.200.183.250 : 12191
Result of: 106.200.183.250 : 12191  is  3.1374366046514623
Connection to: 106.200.183.250 closed
Connected to : 106.200.183.250 : 12194
Result of: 106.200.183.250 : 12194  is  3.143001613051509
Connection to: 106.200.183.250 closed
Connected to : 128.14.134.134 : 39532
```

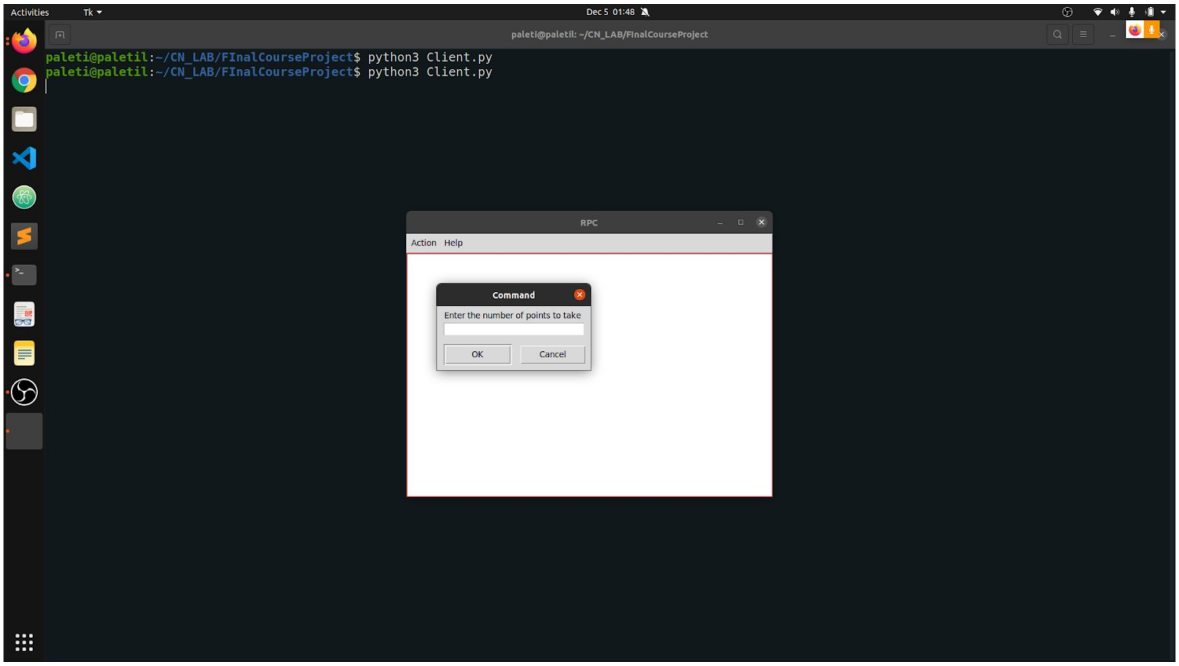
Multiple clients screenshots: -



```
paleti@paletil:~/CN_LAB/FInalCourseProject$ python3 Client.py
paleti@paletil:~/CN_LAB/FInalCourseProject$ python3 Client.py
```

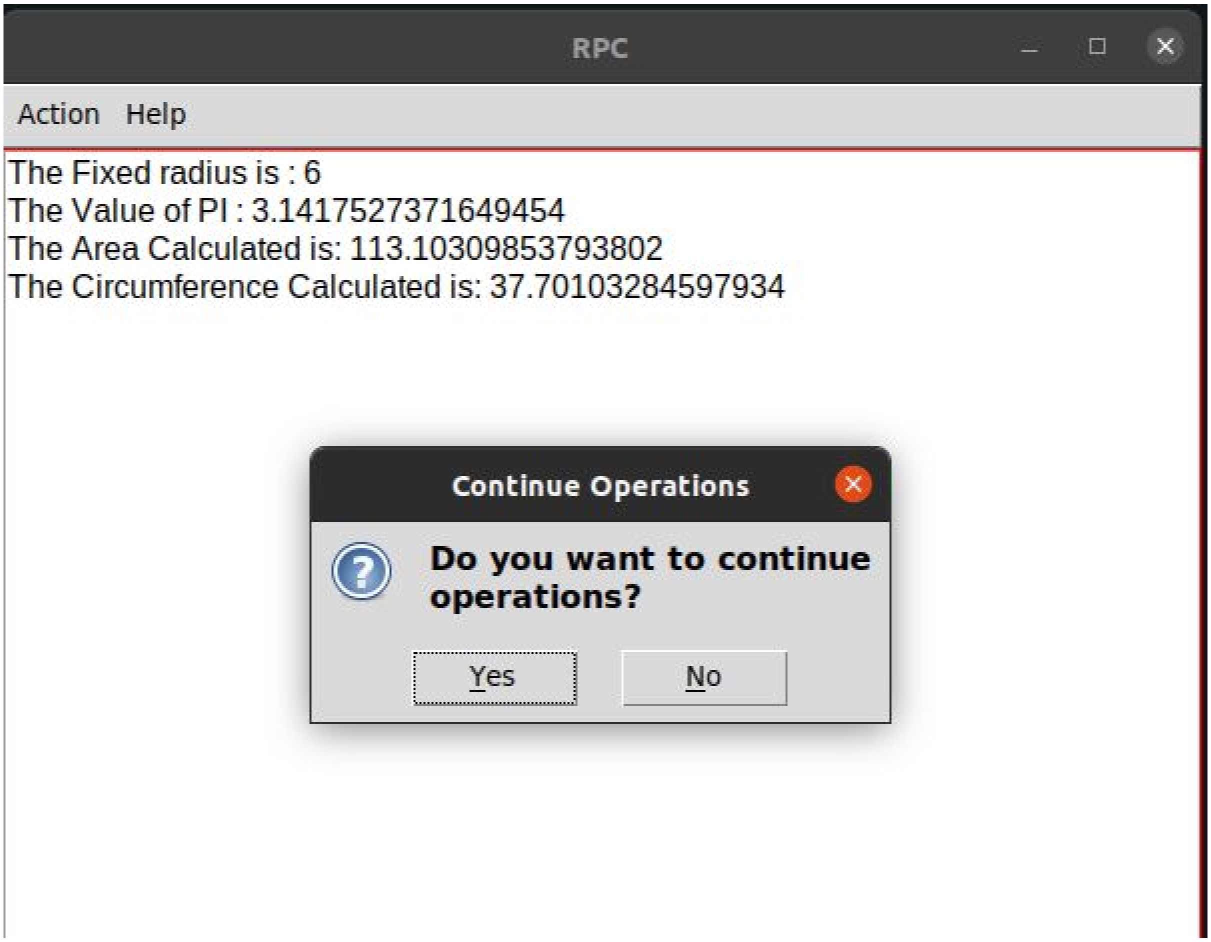


```
paleti@paletil:~/CN_LAB/FInalCourseProject$ python3 Client.py
paleti@paletil:~/CN_LAB/FInalCourseProject$ python3 Client.py
```



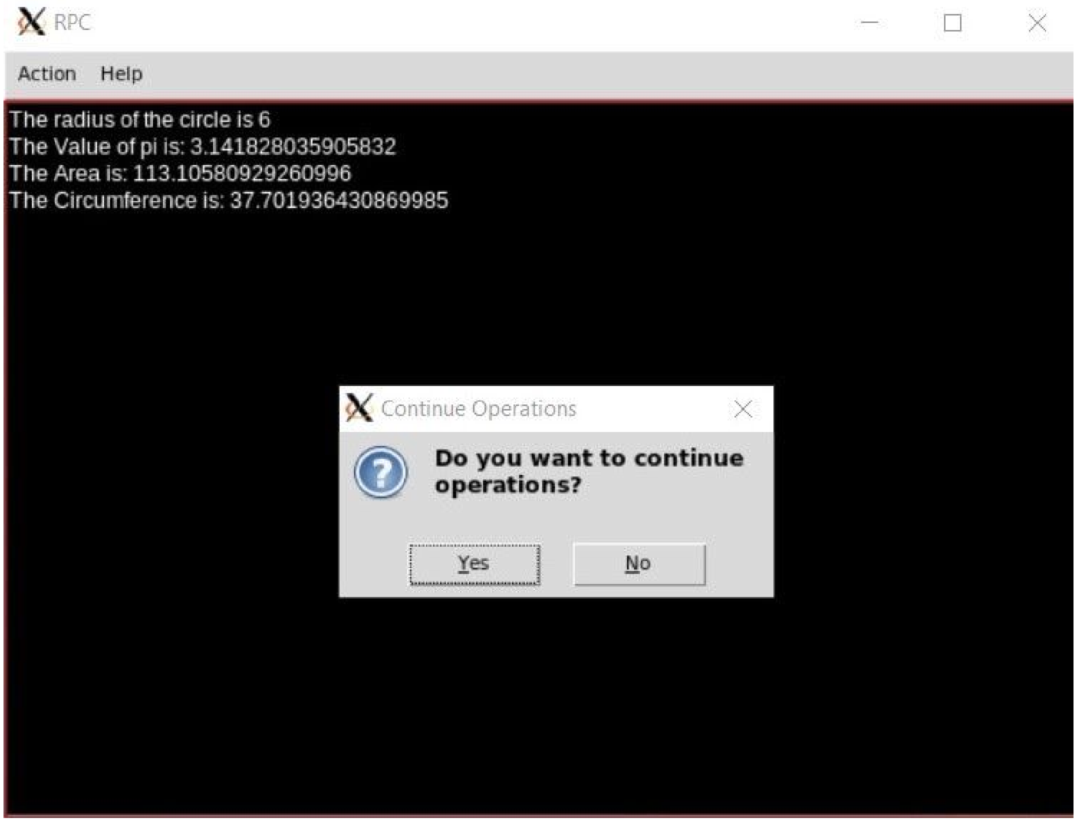
```
paleti@paletil:~/CN_LAB/FInalCourseProject$ python3 Client.py
paleti@paletil:~/CN_LAB/FInalCourseProject$ python3 Client.py
```

Enter the number of points to take



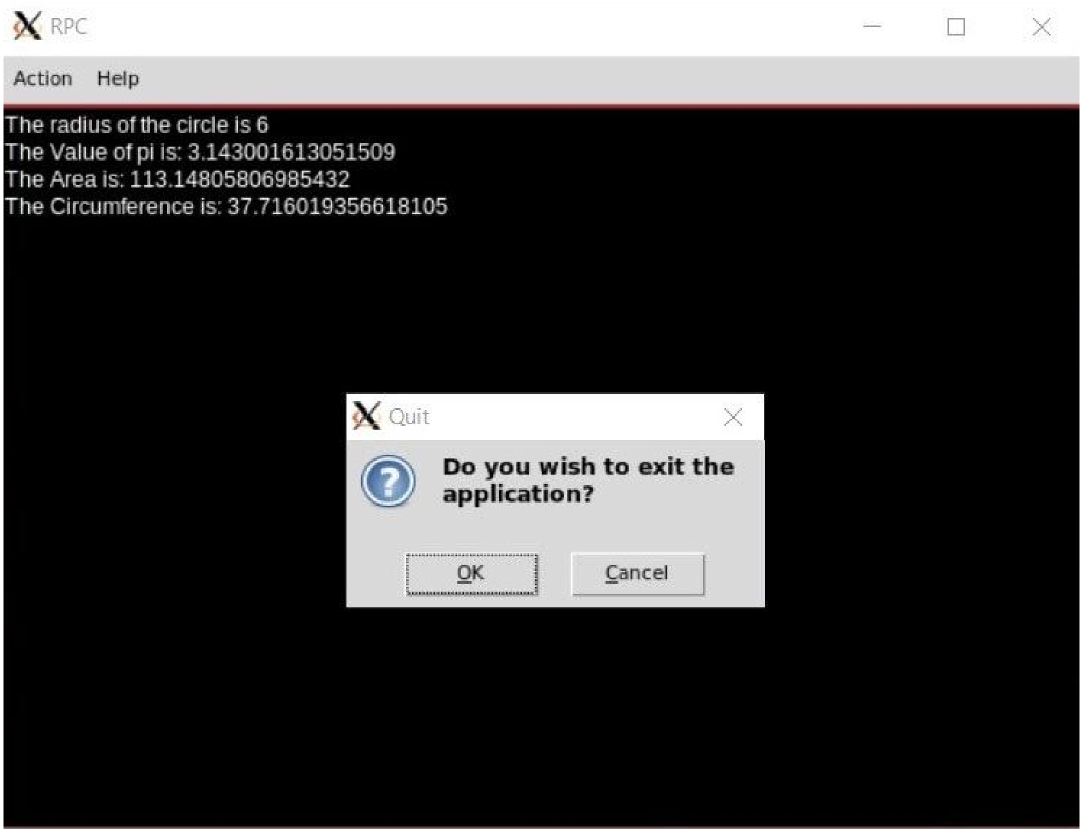
```
The Fixed radius is : 6
The Value of PI : 3.1417527371649454
The Area Calculated is: 113.10309853793802
The Circumference Calculated is: 37.70103284597934
```

Do you want to continue operations?



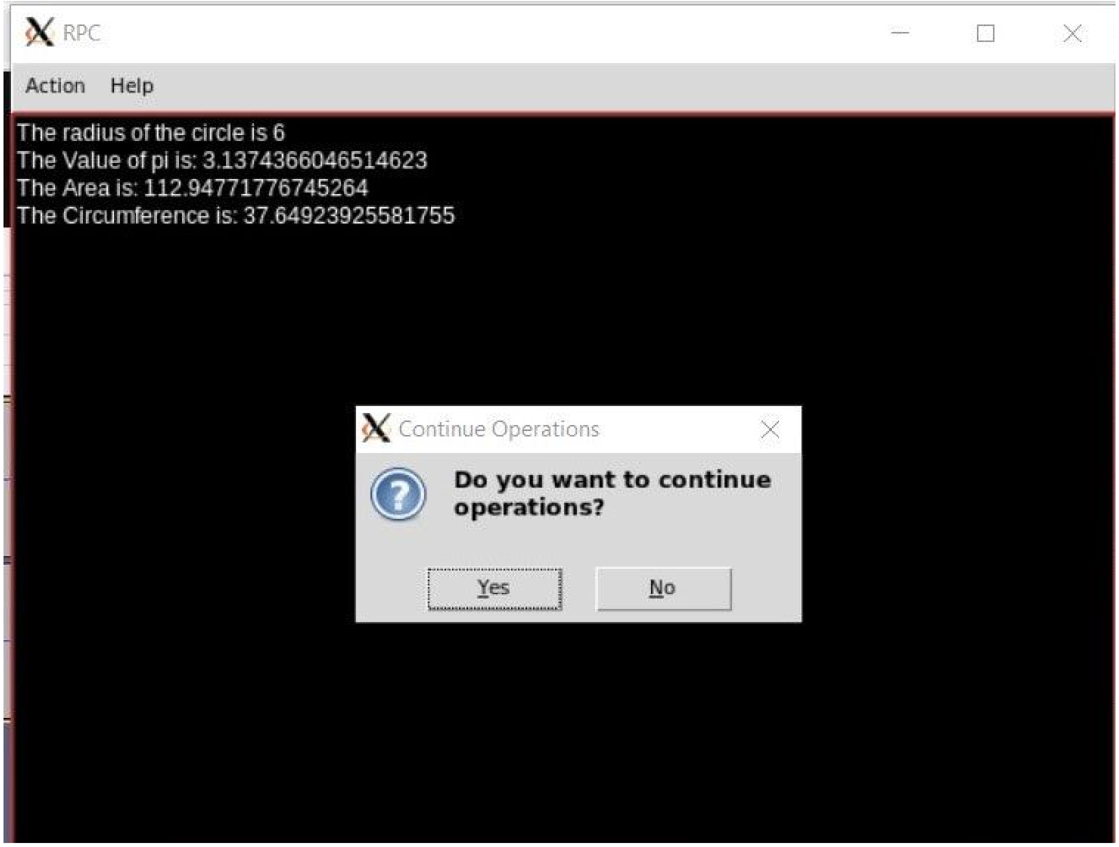
```
The radius of the circle is 6
The Value of pi is: 3.141828035905832
The Area is: 113.10580929260996
The Circumference is: 37.701936430869985
```

Do you want to continue operations?



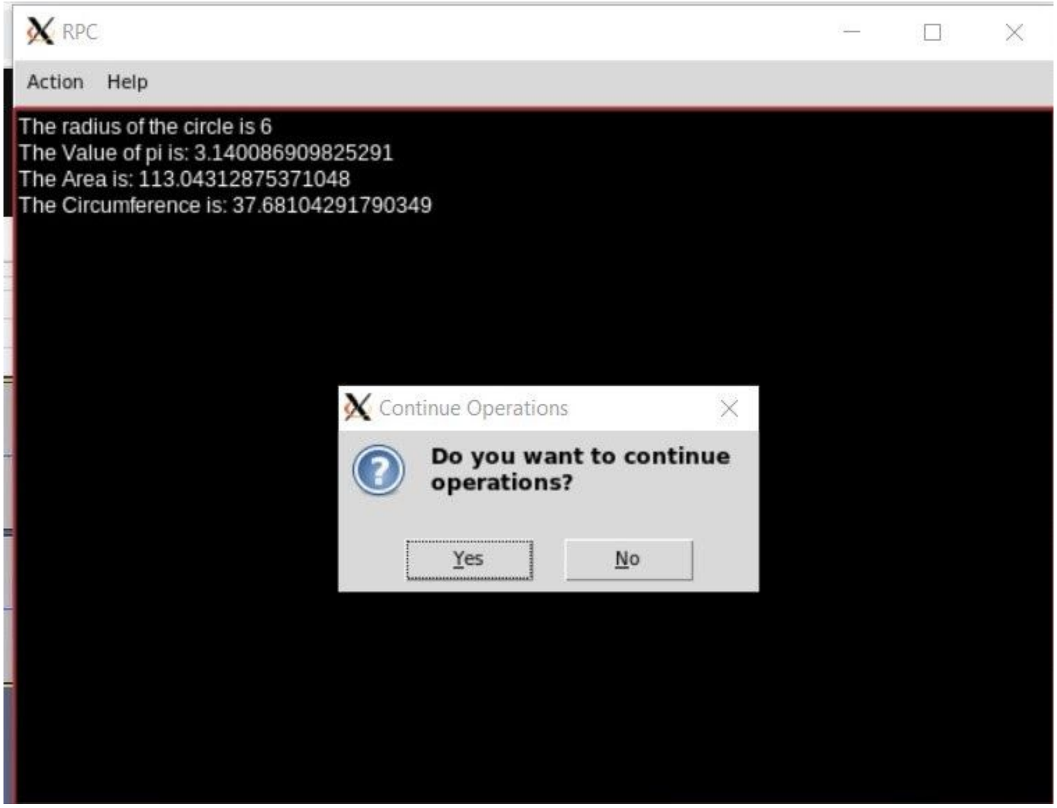
```
The radius of the circle is 6
The Value of pi is: 3.143001613051509
The Area is: 113.14805806985432
The Circumference is: 37.716019356618105
```

Do you wish to exit the application?



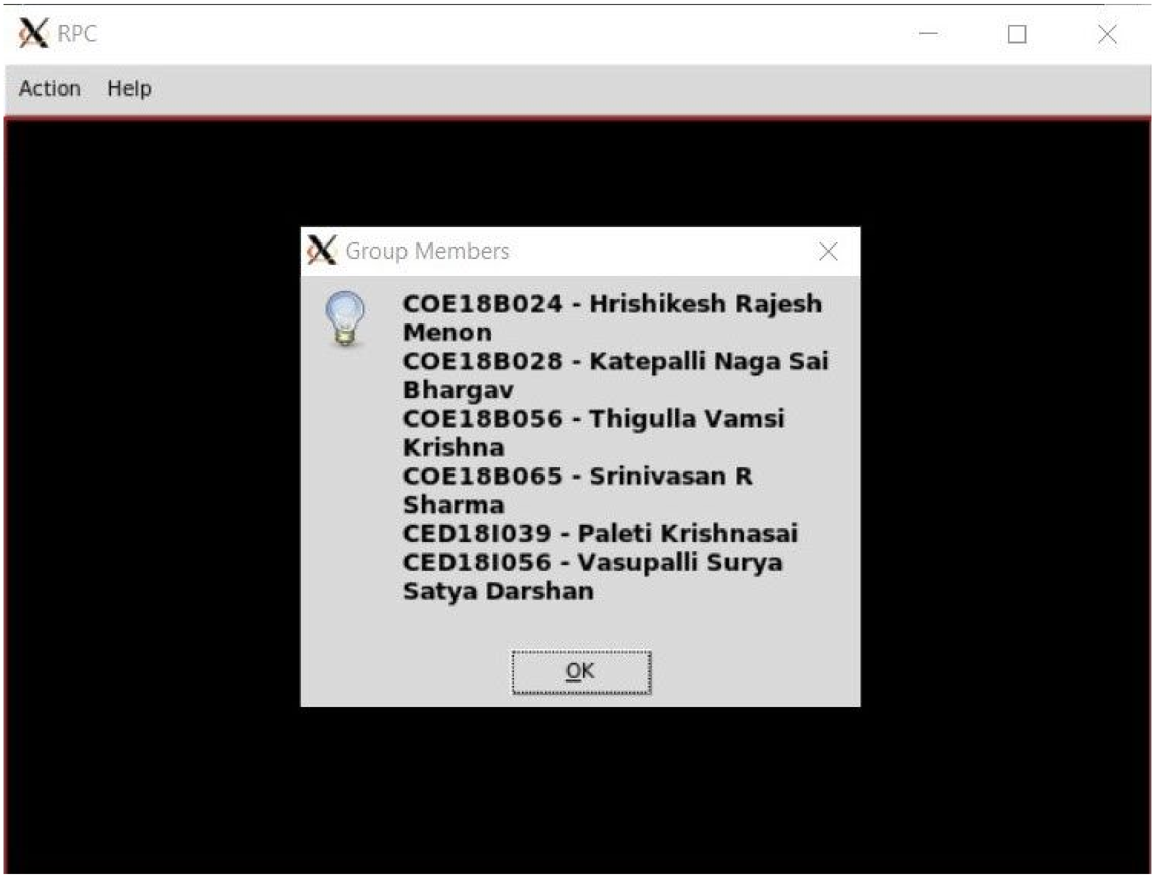
```
The radius of the circle is 6
The Value of pi is: 3.1374366046514623
The Area is: 112.94771776745264
The Circumference is: 37.64923925581755
```

Do you want to continue operations?



```
The radius of the circle is 6
The Value of pi is: 3.140086909825291
The Area is: 113.04312875371048
The Circumference is: 37.68104291790349
```

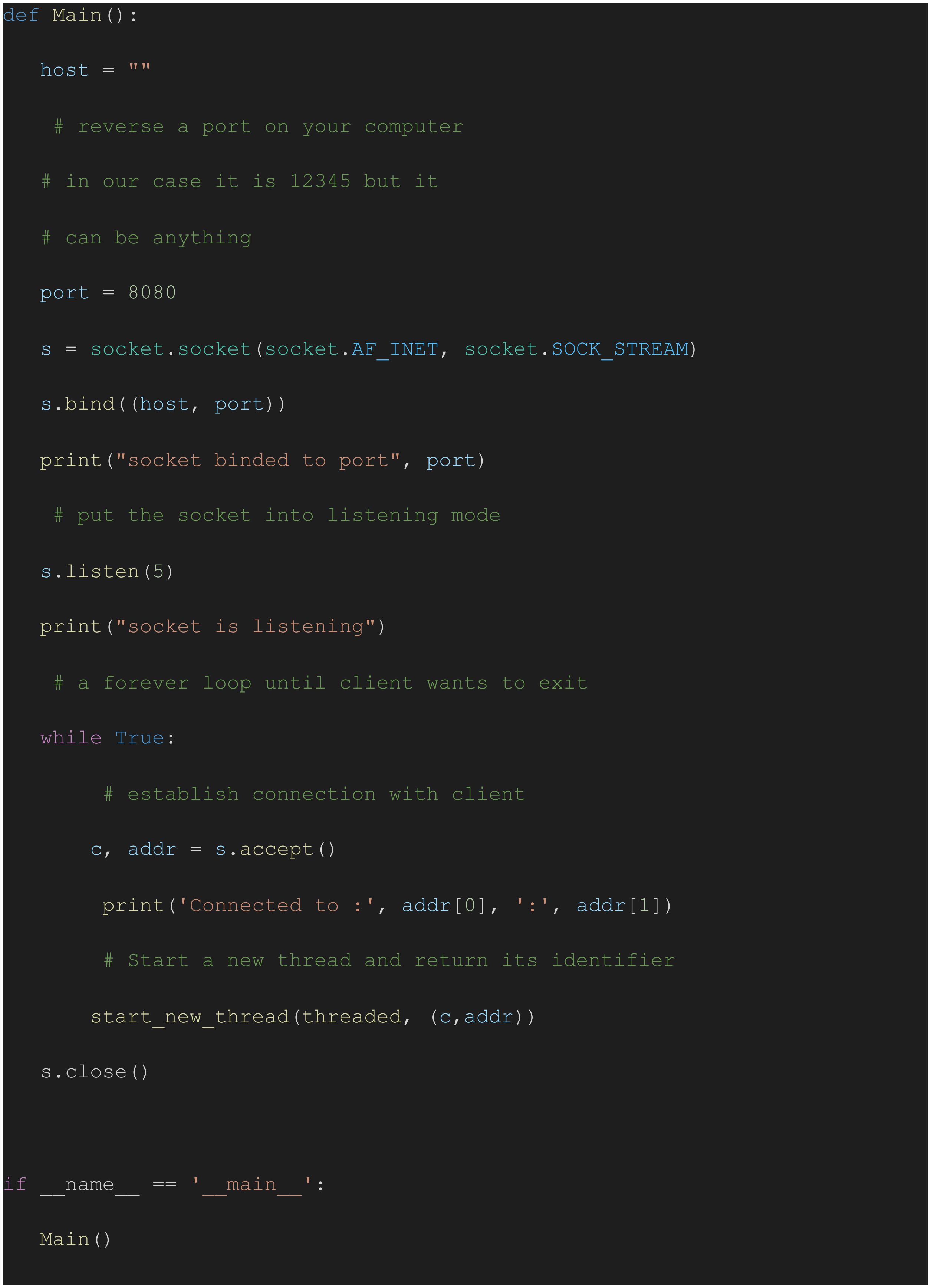
Do you want to continue operations?



COE18B024 - Hrishikesh Rajesh Menon
COE18B028 - Katepalli Naga Sai Bhargav
COE18B056 - Thigulla Vamsi Krishna
COE18B065 - Srinivasan R Sharma
CED18I039 - Paleti Krishnasai
CED18I056 - Vasupalli Surya Satya Darshan

Server :

```
# import socket programming library
import socket
 # import thread module
from _thread import *
import threading
import subprocess
import random

#function to run
def FindPi(interval):
   circle_points= 0
   square_points= 0



   for i in range(interval):
       rand_x= random.uniform(-1, 1)
       rand_y= random.uniform(-1, 1)

       origin_dist= rand_x**2 + rand_y**2

       if origin_dist<= 1:
           circle_points+= 1
       square_points+= 1


   pi = 4* circle_points/ square_points

   return(str(pi))



# thread function

def threaded(c,addr):

   while True:

        # comand received from client

       data = c.recv(1024)

       if not data:

           print('Connection to:',addr[0],"closed")

           break


       out = FindPi(int(str(data.decode('utf-8'))))

       print('Result of:',addr[0],":",addr[1]," is ",out)


       # send back reversed string to client

       c.send(bytes(out,'utf-8'))

    # connection closed

   c.close()
```



```
def Main():

   host = ""

    # reverse a port on your computer

   # in our case it is 12345 but it

   # can be anything

   port = 8080

   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

   s.bind((host, port))

   print("socket binded to port", port)

    # put the socket into listening mode

   s.listen(5)

   print("socket is listening")

    # a forever loop until client wants to exit

   while True:

        # establish connection with client

       c, addr = s.accept()

        print('Connected to :', addr[0], ':', addr[1])

        # Start a new thread and return its identifier

       start_new_thread(threaded, (c,addr))

   s.close()


if __name__ == '__main__':

   Main()
```

Client :

```
import tkinter

import os

from tkinter import *

from tkinter.messagebox import *

from tkinter.filedialog import *

from tkinter import simpledialog

from tkinter import ttk

import threading

import sys

import faulthandler

import socket

sys.setrecursionlimit(10**6)



def calculate(command):

   host = '34.90.102.225'

   port = 8080

   s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

   s.connect((host,port))

   while True:

        # command sent to server

           #message = simpledialog.askstring(title="Client",prompt="Enter
the number of points to take : ")
```

```
        #message = input("Enter the Number of points to take: ")

        s.send(bytes(command, 'utf-8'))


        # recieved response

        data = s.recv(1024)


        # print the received response

        pi = float(str(data.decode('ascii')))

        #print("Pi recieved as:",pi)


        r = 6

        area = pi*r*r

        circum = 2*pi*r

        command_output = "The Fixed radius is : "+str(r)+"\nThe Value of PI : "+str(pi)+"\nThe Area Calculated is: "+str(area)+"\nThe Circumference Calculated is: "+str(circum)+"\n"

        return command_output




class RPC:


    __root = Tk()


    __thisWidth = 300
    __thisHeight = 300


    __thisTextArea = Text(__root,bg="white",font="Arial",fg="black",highlightbackground="red",highlightcolor="green",insertbackground="black",selectbackground="cyan",wrap=WORD)

    __thisMenuBar = Menu(__root)

    __thisActionMenu = Menu(__thisMenuBar, tearoff = 0)

    __thisHelpMenu = Menu(__thisMenuBar, tearoff = 0)


    counter = 0


    def __init__(self,**kwargs):

        try:

            self.__root.wm_iconbitmap("Notepad.ico")

        except:

            pass


        try:

            self.__thisWidth = kwargs['width']

        except KeyError:

            pass


        try:
            self.__thisHeight = kwargs['height']
        except KeyError:
            pass

        self.__root.title("RPC")

        bindtags = list(self.__thisTextArea.bindtags())
        bindtags.remove("Text")
        self.__thisTextArea.bindtags(tuple(bindtags))

        """If the height is not specified in the keyword arguments given, then use default values specified"""

        screenWidth = self.__root.winfo_screenwidth()
        screenHeight = self.__root.winfo_screenheight()

        left = (screenWidth / 2) - (self.__thisWidth / 2)

        top = (screenHeight / 2) - (self.__thisHeight /2)

        self.__root.geometry('%dx%d+%d+%d' % (self.__thisWidth,self.__thisHeight,left,top))

        self.__root.grid_rowconfigure(0,weight = 1)
        self.__root.grid_columnconfigure(0,weight = 1)


        self.__thisTextArea.grid(sticky = N + E + S + W)



        self.__thisActionMenu.add_command(label = "Switch Theme",command = self.__theme)


        self.__thisActionMenu.add_command(label = "Enter Command",command = self.__command)


        self.__thisActionMenu.add_separator()
        self.__thisActionMenu.add_command(label = "Exit",command = self.__quitApplication, accelerator = "Ctrl + Q")


        self.__thisMenuBar.add_cascade(label = "Action",menu = self.__thisActionMenu)


        self.__thisHelpMenu.add_command(label = "About",command = self.__showAbout)


        self.__thisHelpMenu.add_command(label = "Group Members",command = self.__showGroup)


        self.__thisMenuBar.add_cascade(label = "Help",menu = self.__thisHelpMenu)
        self.__root.config(menu = self.__thisMenuBar)



    def __showAbout(self):
        showinfo("RPC","Made by Group 1")

    def __showGroup(self):
        showinfo("Group Members","COE18B024 - Hrishikesh Rajesh Menon\nCOE18B028 - Katepalli Naga Sai Bhargav\nCOE18B056 - Thigulla Vamsi Krishna\nCOE18B065 - Srinivasan R Sharma\nCED18I039 - Paleti Krishnasai\nCED18I056 - Vasupalli Surya Satya Darshan")

    def __theme(self):
        if self.counter!=0:
            self.__thisTextArea.config(bg="white",font="Arial",fg="black",highlightbackground="red",highlightcolor="green",insertbackground="black",selectbackground="cyan",wrap=WORD)
            self.counter = 0
        else:
            self.__thisTextArea.config(bg="black",font="Arial",fg="white",highlightbackground="red",highlightcolor="green",insertbackground="white",selectbackground="yellow",wrap=WORD)
            self.counter = 1
```

```python
    def __quitApplication(self,event = None):

        if messagebox.askokcancel("Quit","Do you wish to exit the application?"):

            self.__root.destroy()



    def run(self):

        self.__root.mainloop()



    def __command(self):

        command = simpledialog.askstring("Command","Enter the number of points to take ")

        command_output = "Output"

        if command is not None:

            """Code for output from socket here"""

            """

            Insert all code required to be put in main here

            This part contains code to send the variable 'command' to the socket

            On receiving the output, it is stored in 'command_output'

            """



            command_output = calculate(command)
            self.__thisTextArea.delete(1.0,END)

            #command_output = "The Area is: "+str(area)+"\nThe Circumference is: "+str(circum)+"\n"

            self.__thisTextArea.insert(1.0,command_output)



        op_continue = messagebox.askyesno("Continue Operations","Do you want to continue operations?")

        if op_continue:

            self.__command()

        else:

            self.__quitApplication()




RP = RPC(width=600,height=400)

RP.run()
```

Team Contributions :

Hrishikesh Rajesh Menon - Cloud Hosting and VM setup

Katepalli Naga Sai Bhargav - GUI

Darshan VSS - GUI

Vamsi Krishna Thigulla - GUI

Srinivasan Sharma - Client and Server stubs and functions (multithreading).

Paleti Krishnasai - Client and Server stubs and functions (multithreading).

END
