Experiment 4

Sunday, 19 April 2020 6:49 PM

Mapúa University
School of Electrical, Electronics, and Computer Engineering

Experiment 4: Event-Driven and Concurrent Programming

CPE106L (Software Design Laboratory)

Gian Karlo Madrid
Rendell Sheen Suliva
Rane Gillian Villanueva

Group: 01
Section: B1

PreLab

- **Readings, Insights, and Reflection**

  Core Python Programming, Rao (9789351198918) : Chapter 21, Threads pg. 537
  Thread is a separate path of execution of a group of statements in a program. In the context of Python programming, if you code a group of statements, it is run or executed by a Phython Virtual Machine one by one, which is also known as thread. Every Pyhthon program has a thread running internally that is appointed by the Python Virtual Machine to execute the program statements. Basically threading in python is used to run multiple threads synchronically. It is very helpful when you want to run multiple statements in your program at the same time without having a problem since threading is happening. It is a way for the program to divide the tasks itself to simultaneously run tasks. A thread has a single task or multitasking. A single tasking means that the it is doing some calculations and processing, only one task is given to the processor at a time. While multitasking means the processor is doing multiple jobs or tasks at a time not just one unlike in single task you can perform in multitasking multiple jobs at the same time.

  Core Python Programming, Rao (9789351198918) : Chapter 22, GUI in Python
  On this book, it talks about GUI in python. As I'm reading the book, I thought that GUI in python is just the typical GUI most of us know which is an interface that a user can interact with wherein it is an application through graphics or images that can be interacted by a user. But GUI in Python offer tkinter module wherein we can create graphics program. It is an interface in Pyhton programming that equips the programmers to use the classes of TK module of TCL/TK language. The TCK also knwon as Tool Command Language that is capable of web and desktop application, networking administrations, testing and other useful functions. It is open source and can be used by anyone and also it uses TK Tool Kit that can create graphics. Basically GUI in python programming is user friendly and you can integrate your python programs in GUI easily.

  Fundamentals of Python: First Programs, Lambert (9781337671019): Chapter 8: Graphical User Interfaces page 245
  Chapter 8 was all about GUI, Graphical User Interface is a system of interactive visual components for a software. It can visually represent the functions of the created program and be interactive and user friendly to the users. A software is said to better if it has a GUI since it is interactive in a way you can click a button and it will have a certain function and not just a 2D representation of a computer software. Of course a GUI object created will not work if it is not specified or the program does not have a certain function for the said object, it will be useless and just be a design in the interface. To make it work, the programmer must integrate the GUI objects to the code or the software program to be fully functional GUI.

- **Answers to Questions**
  - Short answer
    1. Clicking a command button in a GUI invokes the event of OnClick function.
    2. This is a good idea to check whether the window's components are working properly according to the objective of the program. Also, this is also to check whether the controls on the window's components are responding according to which type of even is the control.

  - Multiple Choice
    1. b
    2. a
    3. b
    4. a
    5. b
    6. c
    7. a
    8. b
    9. a
    10. b

InLab

- **Objectives**
  - **To apply GUI on the applications.**
  - **To implement threads for the applications.**
  - **To implement and apply client/server programming through network.**
  - **To apply the fundamentals of strings and characters.**

- **Source Code of Machine Problem 1:**

```python
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):

    def setupUi(self, MainWindow):

        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(243, 197)
```

```python
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayoutWidget_3 = QtWidgets.QWidget(self.centralwidget)
        self.verticalLayoutWidget_3.setGeometry(QtCore.QRect(20, 20, 201, 131))
        self.verticalLayoutWidget_3.setObjectName("verticalLayoutWidget_3")
        self.verticalLayout_4 = QtWidgets.QVBoxLayout(self.verticalLayoutWidget_3)
        self.verticalLayout_4.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout_4.setObjectName("verticalLayout_4")
        self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_4.setObjectName("horizontalLayout_4")
        self.verticalLayout_10 = QtWidgets.QVBoxLayout()
        self.verticalLayout_10.setObjectName("verticalLayout_10")
        self.Label_Cel = QtWidgets.QLabel(self.verticalLayoutWidget_3)

        font = QtGui.QFont()
        font.setFamily("Segoe UI")
        font.setPointSize(13)

        self.Label_Cel.setFont(font)
        self.Label_Cel.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.Label_Cel.setObjectName("Label_Cel")
        self.verticalLayout_10.addWidget(self.Label_Cel)
        self.Label_Fah = QtWidgets.QLabel(self.verticalLayoutWidget_3)

        font = QtGui.QFont()
        font.setFamily("Segoe UI")
        font.setPointSize(13)

        self.Label_Fah.setFont(font)
```

```python
        self.Label_Fah.setFont(font)
        self.Label_Fah.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.Label_Fah.setObjectName("Label_Fah")
        self.verticalLayout_10.addWidget(self.Label_Fah)
        self.horizontalLayout_4.addLayout(self.verticalLayout_10)
        self.verticalLayout_5 = QtWidgets.QVBoxLayout()
        self.verticalLayout_5.setObjectName("verticalLayout_5")
        self.LineEdit_Cel = QtWidgets.QLineEdit(self.verticalLayoutWidget_3)

        font = QtGui.QFont()
        font.setFamily("Consolas")
        font.setPointSize(12)
        self.LineEdit_Cel.setFont(font)
        self.LineEdit_Cel.setObjectName("LineEdit_Cel")
        self.verticalLayout_5.addWidget(self.LineEdit_Cel)
        self.LineEdit_Fah = QtWidgets.QLineEdit(self.verticalLayoutWidget_3)

        font = QtGui.QFont()
        font.setFamily("Consolas")
        font.setPointSize(12)

        self.LineEdit_Fah.setFont(font)
        self.LineEdit_Fah.setObjectName("LineEdit_Fah")
        self.verticalLayout_5.addWidget(self.LineEdit_Fah)
        self.horizontalLayout_4.addLayout(self.verticalLayout_5)
        self.verticalLayout_4.addLayout(self.horizontalLayout_4)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.PushButton_FahToCel = QtWidgets.QPushButton(self.verticalLayoutWidget_3)
        self.PushButton_FahToCel.setObjectName("PushButton_FahToCel")
        self.PushButton_FahToCel.clicked.connect(self.fahToCel)
        self.horizontalLayout.addWidget(self.PushButton_FahToCel)
        self.PushButton_CelToFah = QtWidgets.QPushButton(self.verticalLayoutWidget_3)
        self.PushButton_CelToFah.setObjectName("PushButton_CelToFah")
        self.PushButton_CelToFah.clicked.connect(self.celToFah)
        self.horizontalLayout.addWidget(self.PushButton_CelToFah)
        self.verticalLayout_4.addLayout(self.horizontalLayout)
```

```python
        self.horizontalLayout.addWidget(self.PushButton_CelToFah)
```

```python
        self.verticalLayout_4.addLayout(self.horizontalLayout)
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.PushButton_Reset = QtWidgets.QPushButton(self.verticalLayoutWidget_3)
        self.PushButton_Reset.setObjectName("PushButton_Reset")
        self.PushButton_Reset.clicked.connect(self.reset)
        self.horizontalLayout_2.addWidget(self.PushButton_Reset)
        self.verticalLayout_4.addLayout(self.horizontalLayout_2)

        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 243, 21))
        self.menubar.setObjectName("menubar")

        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")

        MainWindow.setStatusBar(self.statusbar)
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Temp Converter"))
        self.Label_Cel.setText(_translate("MainWindow", "Celcius:"))
        self.Label_Fah.setText(_translate("MainWindow", "Fahrenheit:"))
        self.PushButton_FahToCel.setText(_translate("MainWindow", ">>>>"))
        self.PushButton_CelToFah.setText(_translate("MainWindow", "<<<<"))
        self.PushButton_Reset.setText(_translate("MainWindow", "Reset"))

        self.reset

    def fahToCel(self):
        cel = (float(self.LineEdit_Fah.text()) - 32) * 5/9
        fCel = round(cel, 2)
```

```python
        cel = (float(self.LineEdit_Fah.text()) - 32) * 5/9
        fCel = round(cel, 2)
        self.LineEdit_Cel.setText(str(fCel))

    def celToFah(self):
        fah = (float(self.LineEdit_Cel.text()) * 9/5) + 32
        fFah = round(fah, 2)
        self.LineEdit_Fah.setText(str(fFah))

    def reset(self):
        self.LineEdit_Cel.setText(str(0.0))
        self.LineEdit_Fah.setText(str(32.0))

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

- **Source code of Machine Problem 2:**

```python
"""
File: doctorserver.py
Server for providing non-directive psychotherapy.
Uses client handlers to handle clients' requests.
"""

from socket import *
from doctorclienthandler import DoctorClientHandler
```

```python
from doctorclienthandler import DoctorClientHandler

HOST = "localhost"
PORT = 4321
ADDRESS = (HOST, PORT)

server = socket(AF_INET, SOCK_STREAM)
server.bind(ADDRESS)
server.listen(5)

# The server now just waits for connections from clients
# and hands sockets off to client handlers
try:
    while True:
        print("Waiting for connection . . .")
        client, address = server.accept()
        print(str(client), str(address))
        print("... connected from: ", address)
        print(client)
        handler = DoctorClientHandler(client)
        handler.start()
finally:
    #dir(server.shutdown())
    server.close()
```

```python
"""
File: doctorclient.py
GUI-based view for client for non-directive psychotherapy.
"""

#fix textbox at start to avoid confusion

from socket import *
from codecs import decode
from breezypythongui import EasyFrame

HOST = "localhost"
PORT = 4321
BUFSIZE = 1024
ADDRESS = (HOST, PORT)
CODE = "ascii"

class DoctorClient(EasyFrame):
    """Represents the client's window."""

    COLOR = "#CCEEFF"      # Light blue

    def __init__(self):
        """Initialize the window and widgets."""
        EasyFrame.__init__(self, title = "Doctor",
                           background = DoctorClient.COLOR)

        self.name = ''

        # Add the labels, fields, and button
        self.drLabel = self.addLabel(text = "Want to connect?",
                                     row = 0, column = 0,
                                     columnspan = 2,
                                     background = DoctorClient.COLOR)
        self.ptField = self.addTextField(text = "", row = 1,
                                         column = 0,
                                         columnspan = 2,
                                         width = 50,
```

```python
                                         width = 50,
                                         state='disabled')
        self.sendBtn = self.addButton(row = 2, column = 0,
                                      text = "Send reply",
                                      command = self.sendReply,
                                      state = "disabled")
```

```python
        self.connectBtn = self.addButton(row = 2, column = 1,
                                         text = "Connect",
                                         command = self.connect)

        # Support the return key in the input field
        self.ptField.bind("<Return>", lambda event: self.sendReply())

    def sendReply(self):
        """Sends patient input to doctor, and receives
        and outputs doctor's reply."""

        ptInput = self.ptField.getText()
        if ptInput != "":
            self.server.send(bytes(ptInput, CODE))
            #print(self.server.recv(BUFSIZE))

            drReply = decode(self.server.recv(BUFSIZE), CODE)

            if not drReply:
                self.messageBox(message = "Doctor disconnected")
                self.disconnect()
            else:
                self.drLabel["text"] = drReply
                self.ptField.setText("")

    def connect(self):
        """Starts a new session with the doctor."""
        #print(self.drLabel['text'])
        self.server = socket(AF_INET, SOCK_STREAM)
        #print(self.server)
```

```python
        #print(self.server)
        self.server.connect(ADDRESS)
        # print(BUFSIZE)
        #print(self.server.recv(BUFSIZE))
        #self.name = self.ptField.getText()
        #print(self.name)
        self.drLabel["text"] = decode(self.server.recv(BUFSIZE), CODE)
        #print(self.drLabel['text'])

        self.connectBtn["text"] = "Disconnect"
        self.connectBtn["command"] = self.disconnect
        self.sendBtn["state"] = "normal"
        self.ptField['state'] = 'normal'

    def disconnect(self):
        """Ends the session with the doctor."""
        #self.ptField.setText(decode(self.server.recv(BUFSIZE), CODE))
        self.server.close()
        self.ptField.setText('')
        self.drLabel["text"] = "Want to connect?"
        self.connectBtn["text"] = "Connect"
        self.connectBtn["command"] = self.connect
        self.sendBtn["state"] = "disabled"
        self.ptField['state'] = 'disabled'

def main():
    """Instantiate and pop up the window."""
    DoctorClient().mainloop()

if __name__ == "__main__":
    main()
```

- **Source Code of Machine Problem 3:**

chatserver.py

chatserver.py

```python
"""
File: chatserver.py
Server for a chat room.  Handles one client at a
time and participates in the conversation.
"""

from socket import *
from codecs import decode
from chatclienthandler import ChatClientHandler


HOST = "localhost"
PORT = 5000
ADDRESS = (HOST, PORT)
BUFSIZE = 1024
CODE = "ascii"

server = socket(AF_INET, SOCK_STREAM)
server.bind(ADDRESS)
server.listen(5)

while True:
    print("Waiting for connection . . .")
    client, address = server.accept()
    print("... connected from: ", address)
    handler = ChatClientHandler(client, address)
    handler.start()
```

chatclient.py ×

chatclient.py

```python
#
# Created by: PyQt5 UI code generator 5.9.2
# Potek hirap naman neto. hakdog.
# WARNING! All changes made in this file will be lost!
from socket import *
from codecs import decode
from PyQt5 import QtCore, QtGui, QtWidgets
import ast
from threading import Thread

HOST = "localhost"
PORT = 5000
BUFSIZE = 1024
ADDRESS = (HOST, PORT)
CODE = "ascii"


#server.connect(ADDRESS)


class Ui_MainWindow(object):
    def __init__(self):
        self.connFlag = 0
        self.refFlag = 0
        self.greetings = "Welcome to the Chatroom. You have been connected to the server"
        self.server = socket(AF_INET, SOCK_STREAM)



    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 600)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout.setObjectName("verticalLayout")
        self.lblTitle = QtWidgets.QLabel(self.centralwidget)
        self.lblTitle.setObjectName("lblTitle")
```

```python
        self.lblTitle = QtWidgets.QLabel(self.centralwidget)
        self.lblTitle.setObjectName("lblTitle")
        self.verticalLayout.addWidget(self.lblTitle)
        self.textBrowser = QtWidgets.QTextBrowser(self.centralwidget)
        self.textBrowser.setObjectName("textBrowser")
        self.verticalLayout.addWidget(self.textBrowser)
        self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_3.setObjectName("horizontalLayout_3")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setObjectName("label")
        self.label.setEnabled(False)
        self.horizontalLayout_3.addWidget(self.label)
        self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit.setObjectName("lineEdit")
        self.lineEdit.setEnabled(False)
        self.lineEdit.returnPressed.connect(lambda: self.pressedSend())
        self.horizontalLayout_3.addWidget(self.lineEdit)
        self.verticalLayout.addLayout(self.horizontalLayout_3)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.btnConn = QtWidgets.QPushButton(self.centralwidget)
        self.btnConn.setObjectName("btnConn")
        self.horizontalLayout.addWidget(self.btnConn)
        spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSize
        self.horizontalLayout.addItem(spacerItem)
        self.btnSend = QtWidgets.QPushButton(self.centralwidget)
        self.btnSend.setEnabled(False)
        self.btnSend.setObjectName("btnSend")
        self.horizontalLayout.addWidget(self.btnSend)
        self.verticalLayout.addLayout(self.horizontalLayout)
        MainWindow.setCentralWidget(self.centralwidget)


        self.btnConn.clicked.connect(lambda: self.pressedConn())
        self.btnSend.clicked.connect(lambda: self.pressedSend())

        self.retranslateUi(MainWindow)
```

```python
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Multi-user Chat Room"))
        self.lblTitle.setText(_translate("MainWindow", "Want To Connect?"))
        self.label.setText(_translate("MainWindow", "Message:"))
        self.btnConn.setText(_translate("MainWindow", "Connect"))
        self.btnSend.setText(_translate("MainWindow", "Send"))

    def pressedConn(self):
        if self.connFlag == 0:
            try:
                self.server.connect(ADDRESS)
                #self.textBrowser.append(decode(self.server.recv(BUFSIZE), CODE))
                self.lblTitle.setText("Currently Connected")
                self.btnSend.setEnabled(True)
                self.label.setEnabled(True)
                self.lineEdit.setEnabled(True)
                self.btnConn.setText("Disconnect")
                self.connFlag = 1
                self.textBrowser.clear()
                #self.pressedSend()
                self.windowRefresh()
                #Thread(target=self.windowRefresh, daemon=True).start()
            except Exception as ex:
                print(ex)
        else:
            try:
                #self.textBrowser.append(decode(self.server.recv(BUFSIZE), CODE))
```

```
                self.server.send(bytes("Disconnected", CODE))
                self.textBrowser.clear()
                self.windowRefresh()
                self.server.close()
                self.server = socket(AF_INET, SOCK_STREAM)
                self.btnSend.setEnabled(False)
                self.label.setEnabled(False)
```

```
                self.btnSend.setEnabled(False)
                self.label.setEnabled(False)
                self.lblTitle.setText("Want To Connect?")
                self.lineEdit.setEnabled(False)
                self.btnConn.setText("Connect")
                #self.textBrowser.append("You've disconnected to the chat room")
                self.connFlag = 0
                #Thread.exit()
            except Exception as ex:
                print(ex)


    def pressedSend(self):
        #print("Message: %s" % self.lineEdit.text())
        #self.textBrowser.append("From: %s" % self.lineEdit.text())
        if self.lineEdit.text() == "":
            pass
        else:
            self.server.send(bytes(self.lineEdit.text(), CODE))
            #for i in range(decode(self.server.recv(BUFSIZE), CODE))
            #print(message)
            #self.textBrowser.append(decode(self.server.recv(BUFSIZE), CODE))
            self.textBrowser.clear()
            self.lineEdit.setText("")
            self.windowRefresh()

    def windowRefresh(self):
        message = ast.literal_eval(decode(self.server.recv(BUFSIZE), CODE))
        # if self.refFlag == 0:
        #     self.textBrowser.append("%s: %s" % (message[0][0], message[0][1]))
        # else:
            #print(decode(self.server.recv(BUFSIZE), CODE) )
            #type(message)
            #self.textBrowser.clear()
        for i in range(len(message)):
            if message != []:
                self.textBrowser.append("%s: %s" % (message[i][0], message[i][1]))
```

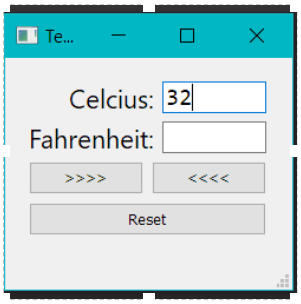

```
    def windowRefresh(self):
        message = ast.literal_eval(decode(self.server.recv(BUFSIZE), CODE))
        # if self.refFlag == 0:
        #     self.textBrowser.append("%s: %s" % (message[0][0], message[0][1]))
        # else:
            #print(decode(self.server.recv(BUFSIZE), CODE) )
            #type(message)
            #self.textBrowser.clear()
        for i in range(len(message)):
            if message != []:
                self.textBrowser.append("%s: %s" % (message[i][0], message[i][1]))
```


```


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    #Thread(target = ui.windowRefresh, daemon=True).start()
    MainWindow.show()
    sys.exit(app.exec_())
```

PostLab

- **Machine Problems**
  1. Write a GUI based program that allows the user to convert temperature values between degrees Fahrenheit and degrees Celsius. The interface should have labeled entry fields for these two values. These components should be arranged in a grid where the labels occupy the first row and the corresponding fields occupy the second row. At start corresponding fields occupy the second row. At start--up, the Fahrenheit field should contain 32.0, and up, the Fahrenheit field should contain 32.0, and the Celsius field should contain 0.0. The third row in the window contains two command buttons, the Celsius field should contain 0.0. The third row in the window contains two command buttons, labeled >>>> and <<<<. Whenlabeled >>>> and <<<<. When the user presses the first button, the program should use the data in the user presses the first button, the program should use the data in the Fahrenheit field to compute the Celsius value, which should then be output to the Celsius field. the Fahrenheit field to compute the Celsius value, which should then be output to the Celsius field. The second button should perform the inverse function.The second button should perform the inverse function.



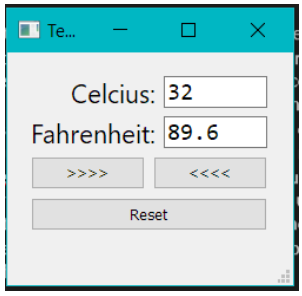
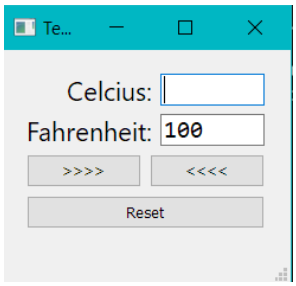


Figure 1 Figure 2
The figures above show the conversion of Celsius to Fahrenheit



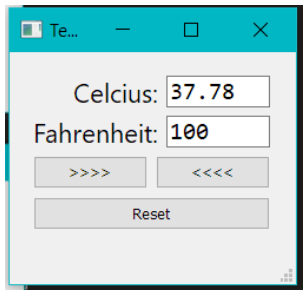


Figure 3 Figure 4
The figures above show the conversion of Fahrenheit to Celsius

  2. On the machine problem 6, this is dice game that player 1 and player 2 plays. The program is running perfectly. The goal in the machine problem is to create a user interaction that the user can play single games and multiple games. I created a payOneGame function for the one play function to work as well as getting the winner and loser by coding getWinner and getLoser function respectively as well as the play function for the single play only in the Player Class. As shown on the output, it tells the user to input 1 if single play only and 2 for multiple plays then it ask the user for how many plays that the program will do. In addition, this is a continuous loop meaning the loop in asking the user for input will not end until the appropriate numbers has been inputted. To exit the program prematurely, simply the user will give 99 value so the program can be exited without doing anything. Also the umlet diagram has been created to show its relationships.

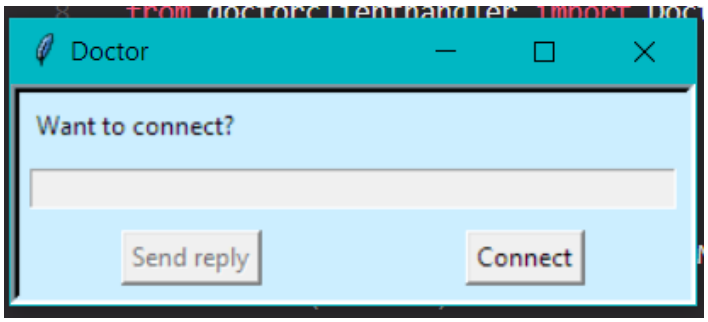


Figure 1
The figure shows the initial setup of the application. If the connect button is pressed, it will proceed to the next step.

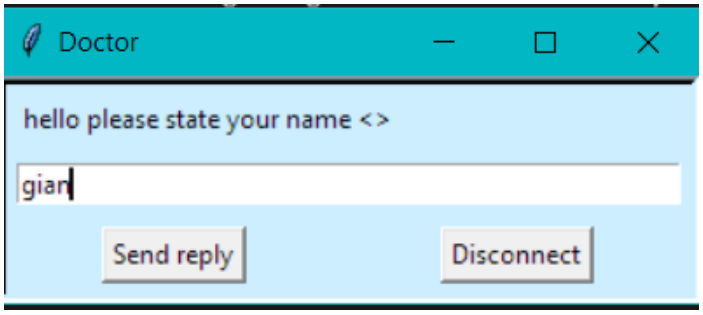


Figure 2
The image above shows the next step is to enter a name and then click send reply button to send the details to the application.

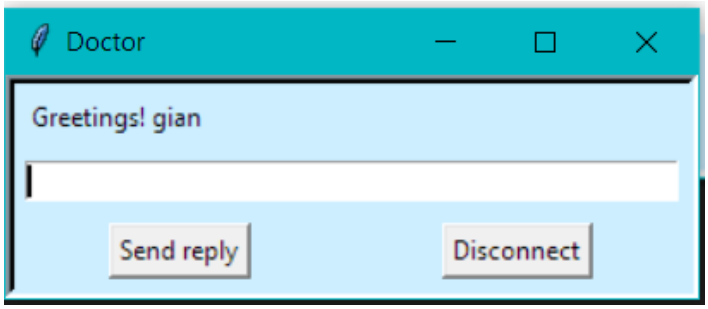


Figure3
This image shows the first actual communication from a user to the doctor, in which the doctor can now accept messages.

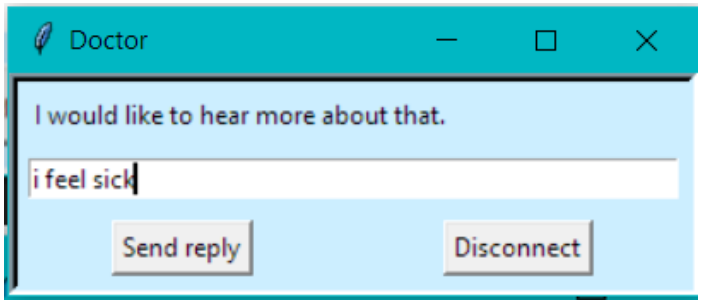


Figure 4
This image shows the ability of the user to say something to the doctor

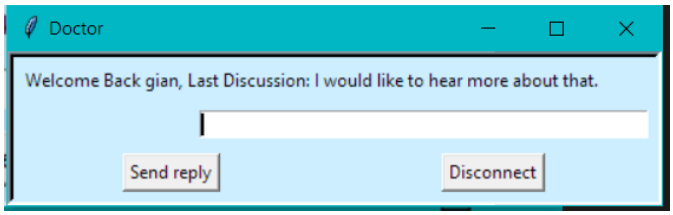


Figure 5
This image shows that if an existing user has been connected before, the doctor will check the records and will know the last message to the user.

3. A crude multi client chat room allows two or more users to converse by sending and receiving messages. On the client side, a user connects to the chat room as in the ATM application, by clicking a Connect button. At that point, a transcript of the conversa tion thus far appears in a text area. At any time, the user can send a message to the chat room by entering it as input and clicking a Send button. When the user sends a message, the chat room returns another transcript of the entire conversation to displa y in the text area. The user disconnects by clicking the Disconnect button.
On the server side, there are five resources: a server, a client handler, a transcript, a thread safe transcript, and a shared cell. Their roles are much the same as they are in t he ATM application of Project 8. The server creates a thread safe transcript at start up, listens for client connections, and passes a client's socket and the thread safe transcript to a client handler when a client connects. The client handler receives th e client's name from the client socket, adds this name and the connection time to the thread safe transcript, sends the thread safe transcript's string to the client, and waits for a reply. When the client's reply comes in, the client handler adds the clie nt's name and time to it, adds the result to the thread safe transcript, and sends the thread safe transcript's string back to the client.

When the client disconnects, her name and a message to that effect are added to the thread safe transcript. The SharedCell class includes the usual read and write methods for a readers and writers protocol, and the SharedTranscript and Transcript classes include an add method and an __str__ method. The add method adds a string to a list of strings, while __str__ ret urns the join of this list, separated by newlines.

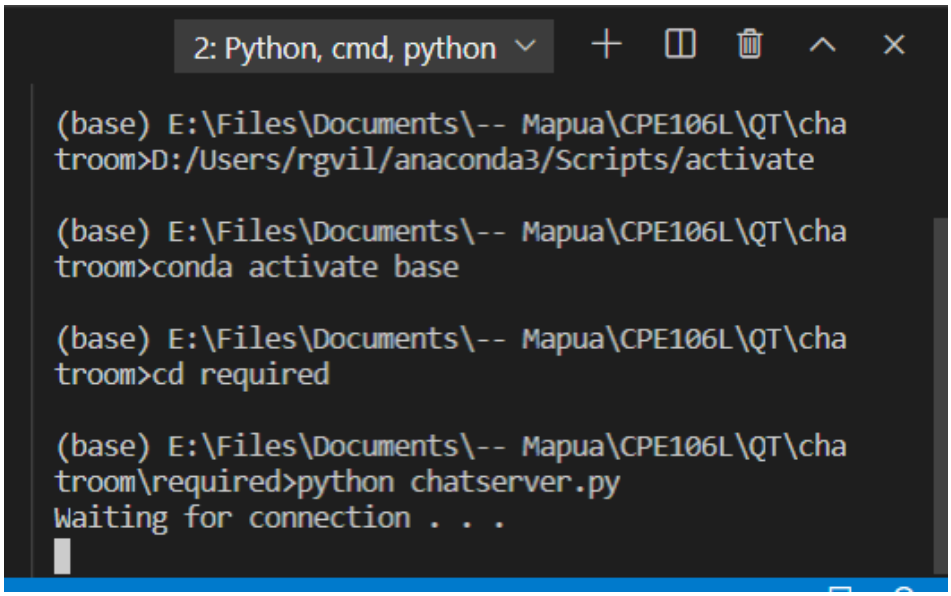


Figure 3.1
The figure above shows that the chatserver.py is run. The server is required in order for chatroom users to enter the room. Without the server, there will be no exchange of communication between users.

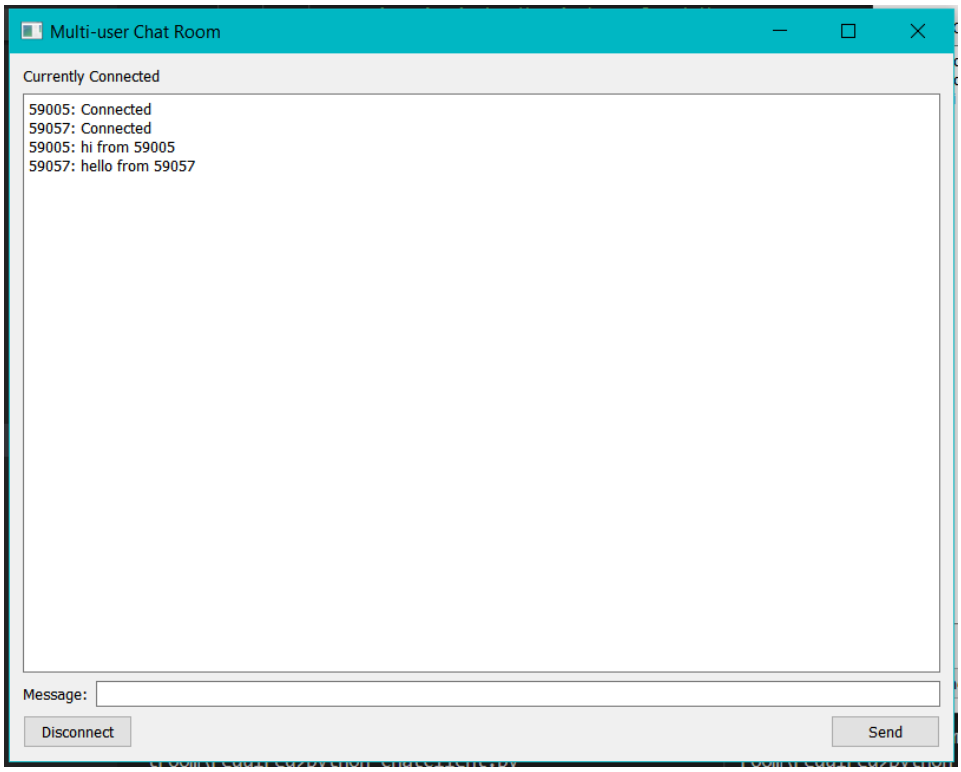
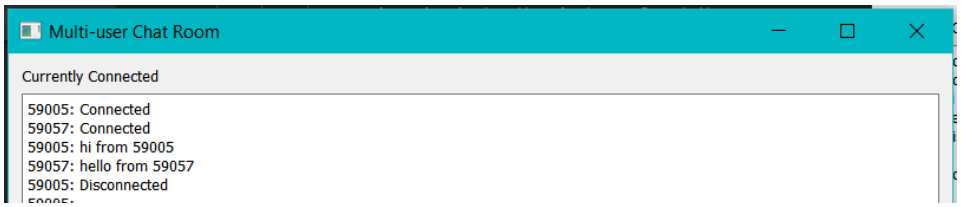


Figure 3.2
This shows the GUI of the users. The users will first connect to the server and when they are successfully connected, chats will be displayed on the screen as well as confirmation that the user has successfully connected on the chatroom.



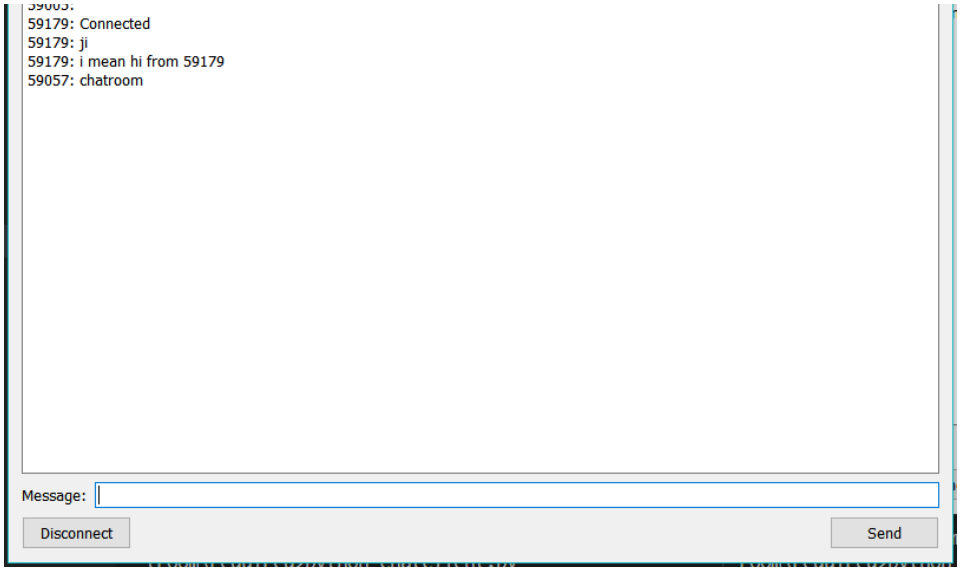


Figure 3.3
The above image shows the conversation of 3 users, 59005,59057,59179. It also shows when a user has connected or disconected from the room.

- **Github:** http://bit.ly/2Ivstu4
- **OneDrive Files:** https://bit.ly/3bshOxm
- **OneDrive PDF:**