MAPÚA UNIVERSITY
SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Lab 3: Event-Driven and Concurrent Programming

CPE106L (Software Design Laboratory)

Hannah Mae Antaran

Darrel Umali Virtusio

Kathleen Joy Tupas

Group: 01

Section: E01

PreLab

Readings, Insights, and Reflection

Lambert, K. A. (2019). Graphical User Interfaces. *Fundamentals of Python: First Programs and Data Structures*. pg. 244.

This chapter teaches us how to develop GUIs and new classes of objects such as application windows, by extending or repurposing existing classes. A GUI displays text as well as small images that represent objects such as folders, files of different types, command buttons, and drop-down menus. It displays all information, including test, graphically to its users and allows them to manipulate this information directly with a pointing device. Learning GUI programming is beneficial for us since it plays a prominent role in modern software development.

Rao, N. R. (2018). Threads. *Core Python Programming*. pg. 537.

Threads are used as a component for a process where statements are executed. Running of statements can be in two types, these are single tasking and multitasking. In Single tasking, processors only accept one job at a time while in Multitasking, several jobs are executed at the same time. Both can be executed through process based or thread-based multitasking. However, when it comes to numerous jobs, or simple statements, thread-based multitasking is highly recommended. In Python, a group of statements running in a python virtual machine are also called threads where it is executed one by one.

Lambert, K. A. (2019). Multithreading, Networks, and Client/Server Programming. *Fundamentals of Python: First Programs and Data Structures*. pg. 352.

This chapter offers an introduction to multithreading, networks, and client/server programming. As reflection to what we read, we learned that multithreading allows the execution of multiple parts of a program at the same time which leads to maximum utilization of the CPU. We could utilize this lesson to create codes efficiently as it is more economical to use threads than processes since they share the same process resources.

Rao, N. R. (2018). Graphical User Interface. *Core Python Programming*. pg. 569.

A Graphical User Interface, also known as GUI, enables a user and a software to communicate and interact using windows, buttons, icons, symbols, and the like. GUI has made the difficult command line interfaces of early made systems into an aesthetically simpler interface. Note that today, a software's user interface is competitively compared and used based on the judgement of its appearance and how it is presented while using it. Users favor those interfaces that they will not need to learn any complicated commands or syntax, just those programs that would get them closer to their goal just by clicking buttons or other visual cues.

Qt Documentation. (n.d.). Retrieved June 27, 2020, from https://doc.qt.io/qtforpython/

In this website, a collection of tutorials with walkthrough guides are provided with Qt for Python to help us get stated. These documents were ported from C++ to Python and cover a range of topics, from basic use of widgets to step by step tutorials that show how an application is put together. This is useful as building GUI applications using PyQt is comparatively less time consuming than code the widgets. It is one of the fastest and easiest ways to create GUIs.

Threading - Thread-based Parallelism. (n.d.). Retrieved June 27, 2020, from https://docs.python.org/3/library/threading.html

This website shows Python's threading module which includes resources for creating threads and managing multi-threaded applications. There are two ways to specify the activity and that is by passing a callable object to the constructor and the other is by overring the run() method in a subclass. This topic is useful since threading is still an appropriate model to run multiple input/output bound tasks simultaneously.

Answers to Questions

1. Explain what happens when a user clicks a command button in a fully functioning GUI program?

   Clicking the command button in a fully functioning GUI will prompt the event of OnClick function.

2. Why is it a good idea to write and test the code for laying out a window's components before you add the methods that perform computations in response to events?

   Window components are often laid and tested before adding methods that perform computations in response to events so that its feel and look can be checked individually, and the corresponding event handler's performance can be reviewed.

Multiple Choice

1. B
2. A
3. B
4. A
5. B
6. C
7. A
8. B
9. A
10. B

InLab

- ## Objectives

1. To learn about event-driven programming.
2. To apply GUI and threads in applications.
3. To implement client/server programming.
4. To reinforce learning of strings in Python.

- ## Tools Used

1. Visual Studio Code
2. Spyder
3. QtDesigner

- ## Procedure

In this part of the experiment, we will provide the source code of the machine problem. The programs apply the learnings from last experiment which are classes and objects and PyQt5 which is a library which contains different classes and function for creating GUI in your program.

Machine Problem 1

tempconverter.py

```python
import sys
from PyQt5 import QtCore, QtGui, QtWidgets, uic
from decimal import Decimal

qtcreator_file = 'TempConvert.ui'  # ui file here
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtcreator_file)


class MyWindow(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        self.FahConvert.clicked.connect(self.Fahconvert)
        self.CelConvert.clicked.connect(self.Celconvert)

    def Fahconvert(self):
        tempC = Decimal(self.Fahrenheit.toPlainText())
        tempC = round(((tempC)-32)*5/9, 3)
        self.Celsius.setPlainText(str(tempC))

    def Celconvert(self):
        tempF = Decimal(self.Celsius.toPlainText())
        tempF = round((tempF)*9/5+32, 3)

        self.Fahrenheit.setPlainText(str(tempF))


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyWindow()
    window.show()
    sys.exit(app.exec_())

```

Figure 1.1 Source code of tempconverter.py.

```python
import sys
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtWidgets import QMainWindow, QWidget, QLabel, QLineEdit
from PyQt5.QtWidgets import QPushButton
from PyQt5.QtCore import QSize

class MainWindow(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)

        self.setMinimumSize(QSize(240, 200))
        self.setWindowTitle("Temperature Converter")

        self.c_to_f_label = QLabel(self)
        self.c_to_f_label.setText('Celsius')
        self.c_to_f_label.move(40, 20)
        self.c_to_f_input = QLineEdit(self)

        self.c_to_f_input.move(30, 60)
        self.c_to_f_input.resize(70, 32)

        self.f_to_c_label = QLabel(self)
        self.f_to_c_label.setText('Fahrenheit')
        self.f_to_c_label.move(140, 20)
        self.f_to_c_input = QLineEdit(self)

        self.f_to_c_input.move(130, 60)
        self.f_to_c_input.resize(70, 32)

        self.setStyleSheet("QLabel {font: 12pt Arial}")
        self.c_to_f_input.setText('0')
        self.f_to_c_input.setText('32')

        pybutton1 = QPushButton('>>>>', self)
        pybutton1.clicked.connect(self.clickMethod1)
        pybutton1.resize(50,32)
        pybutton1.move(40, 100)

        pybutton2 = QPushButton('<<<<', self)
        pybutton2.clicked.connect(self.clickMethod2)
        pybutton2.resize(50,32)
        pybutton2.move(140, 100)

        Clear = QPushButton('Clear', self)
        Clear.clicked.connect(self.clickMethod3)
        Clear.resize(50,32)
        Clear.move(95, 140)

    def clickMethod1(self):
        self.f_to_c_input.setText(str(round(float(self.c_to_f_input.text())*9/5+32,2)))

    def clickMethod2(self):
        self.c_to_f_input.setText(str(round((float(self.f_to_c_input.text())-32)*5/9,2)))

    def clickMethod3(self):
        self.c_to_f_input.clear()
        self.f_to_c_input.clear()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    mainWin = MainWindow()
    mainWin.show()
    sys.exit( app.exec_() )
```

Figure 1.2 Source code of temp_converter.py.

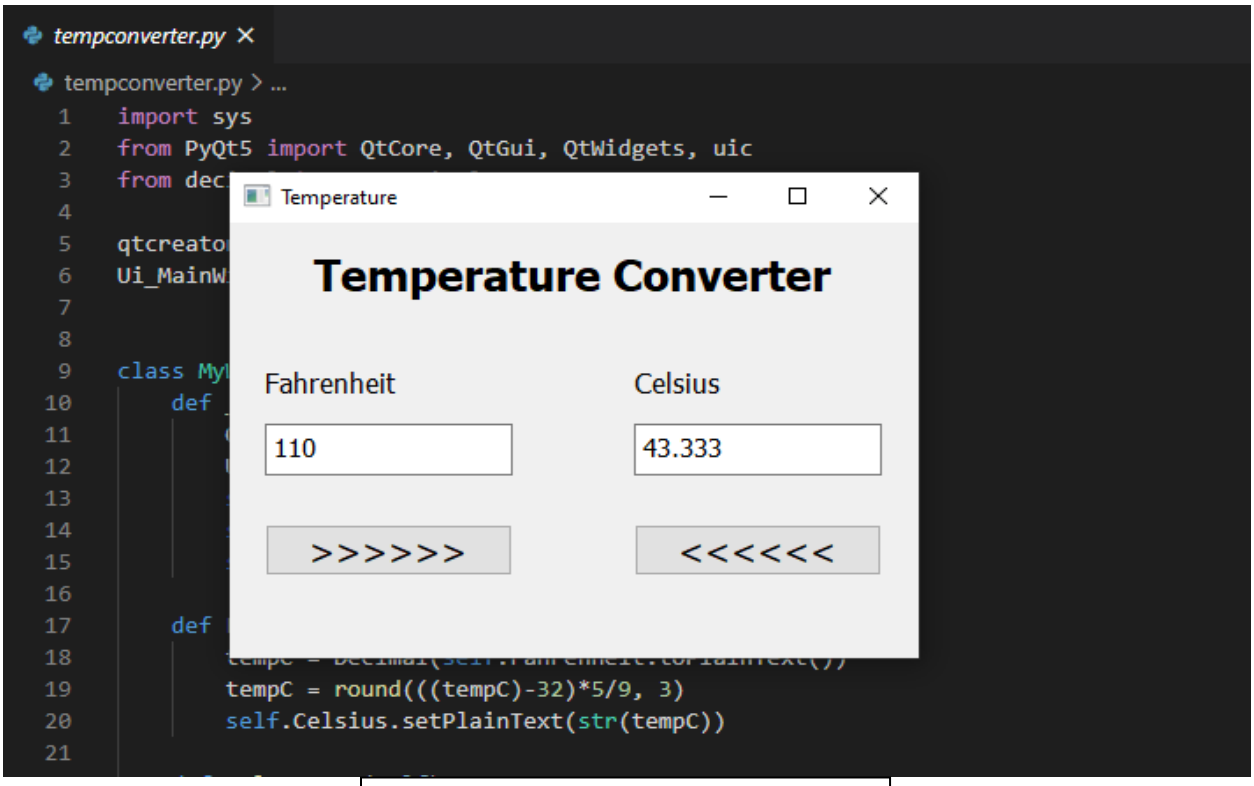


Figure 1.3 Sample run of tempconverter.py.

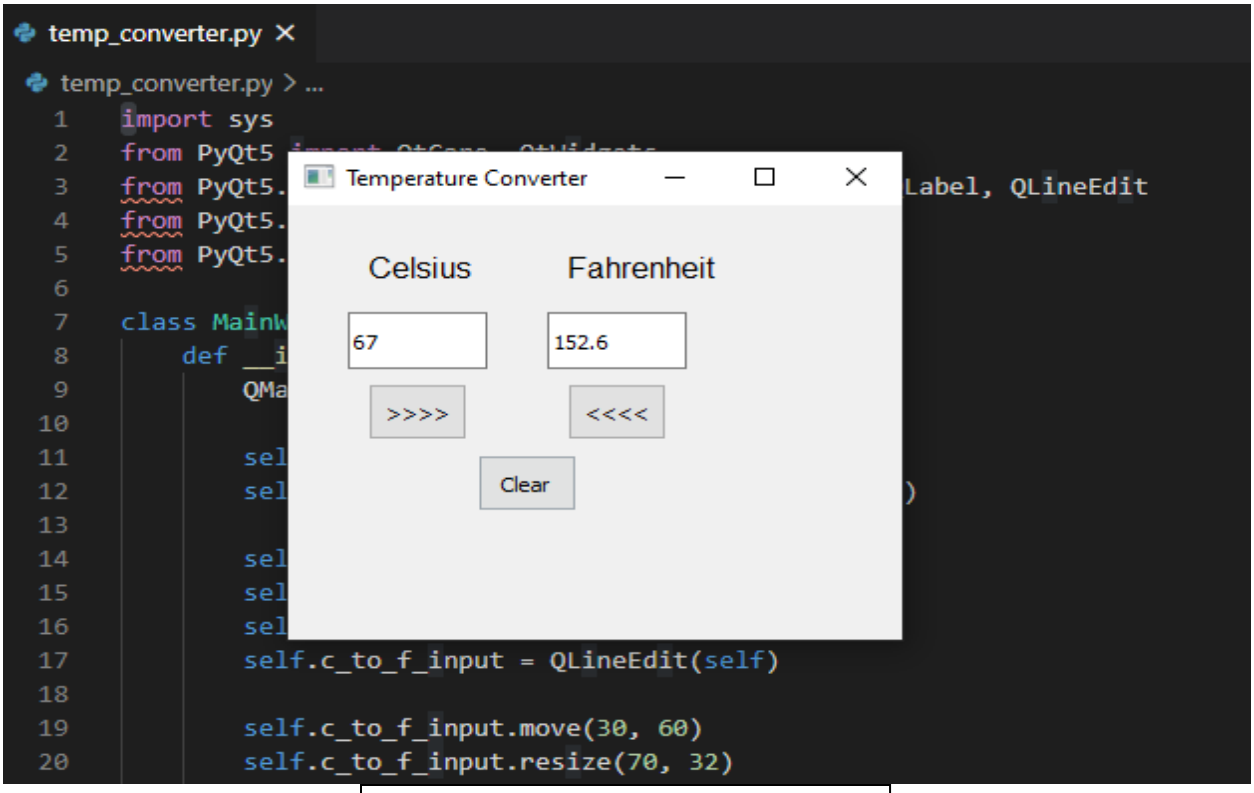
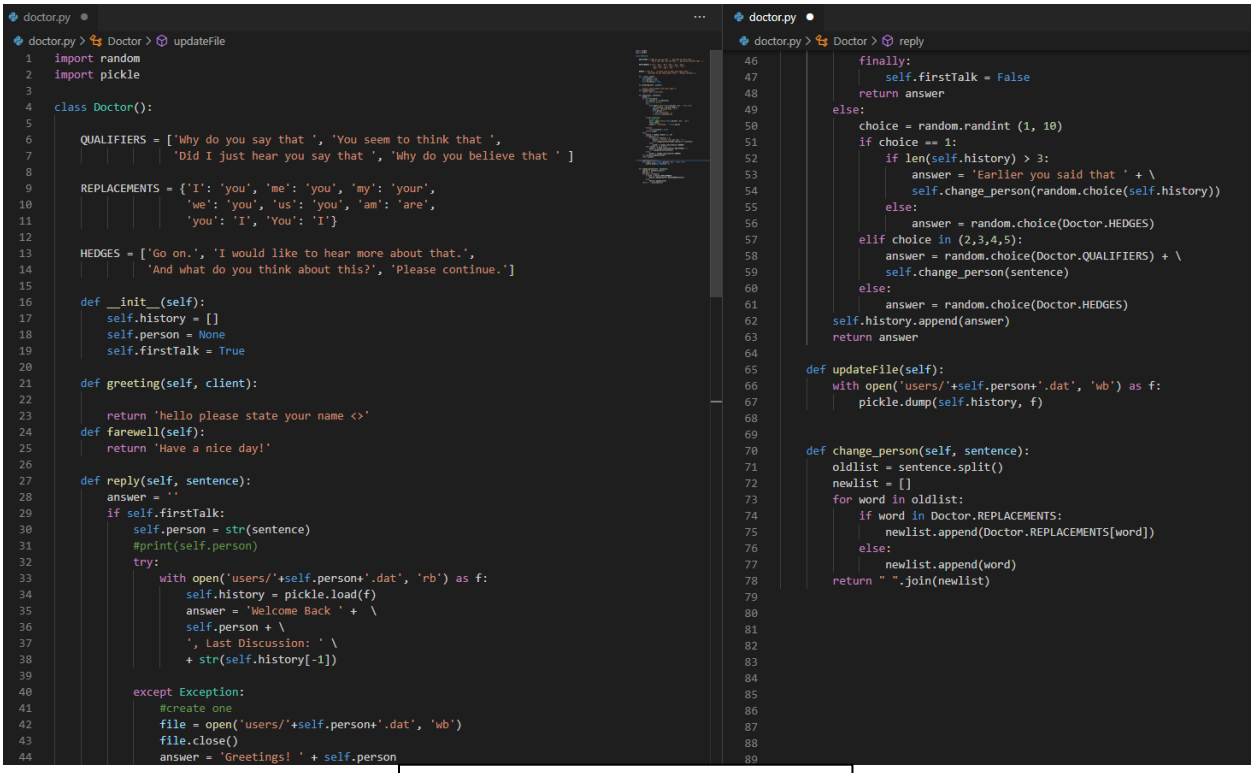


Figure 1.4 Sample run of temp_converter.py.

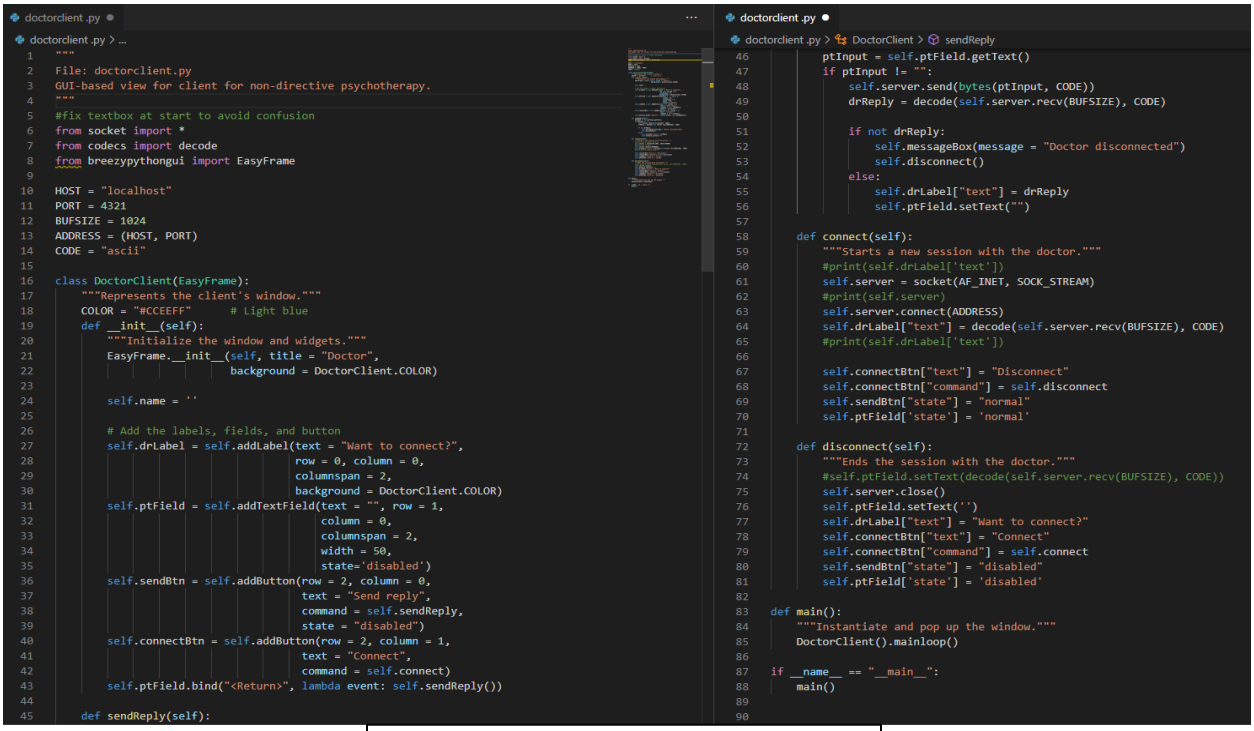
This program incorporates event-driven programming in a temperature converter program. As you can see, both python files are GUI based and has the same function. The difference between the two programs is, temp_converter.py is made out of PyQt5 library. The code is written using the classes of PyQt5. While the tempconverter.py GUI is created using Qt Designer which makes it easier to create the interface because of drag and drop function instead of writing the class itself.

Machine Problem 2



```python
import random
import pickle

class Doctor():

    QUALIFIERS = ['Why do you say that ', 'You seem to think that ',
                  'Did I just hear you say that ', 'Why do you believe that ' ]

    REPLACEMENTS = {'I': 'you', 'me': 'you', 'my': 'your',
                    'we': 'you', 'us': 'you', 'am': 'are',
                    'you': 'I', 'You': 'I'}

    HEDGES = ['Go on.', 'I would like to hear more about that.',
              'And what do you think about this?', 'Please continue.']

    def __init__(self):
        self.history = []
        self.person = None
        self.firstTalk = True

    def greeting(self, client):

        return 'hello please state your name <>'
    def farewell(self):
        return 'Have a nice day!'

    def reply(self, sentence):
        answer = ''
        if self.firstTalk:
            self.person = str(sentence)
            #print(self.person)
            try:
                with open('users/'+self.person+'.dat', 'rb') as f:
                    self.history = pickle.load(f)
                    answer = 'Welcome Back ' +  \
                    self.person + \
                    ', Last Discussion: ' \
                    + str(self.history[-1])

            except Exception:
                #create one
                file = open('users/'+self.person+'.dat', 'wb')
                file.close()
                answer = 'Greetings! ' + self.person
            finally:
                self.firstTalk = False
            return answer
        else:
            choice = random.randint (1, 10)
            if choice == 1:
                if len(self.history) > 3:
                    answer = 'Earlier you said that ' + \
                    self.change_person(random.choice(self.history))
                else:
                    answer = random.choice(Doctor.HEDGES)
            elif choice in (2,3,4,5):
                answer = random.choice(Doctor.QUALIFIERS) + \
                self.change_person(sentence)
            else:
                answer = random.choice(Doctor.HEDGES)
        self.history.append(answer)
        return answer

    def updateFile(self):
        with open('users/'+self.person+'.dat', 'wb') as f:
            pickle.dump(self.history, f)


    def change_person(self, sentence):
        oldlist = sentence.split()
        newlist = []
        for word in oldlist:
            if word in Doctor.REPLACEMENTS:
                newlist.append(Doctor.REPLACEMENTS[word])
            else:
                newlist.append(word)
        return " ".join(newlist)
```

Figure 1.5 Source code of doctor.py.



```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,
                                         state='disabled')
        self.sendBtn = self.addButton(row = 2, column = 0,
                                      text = "Send reply",
                                      command = self.sendReply,
                                      state = "disabled")
        self.connectBtn = self.addButton(row = 2, column = 1,
                                      text = "Connect",
                                      command = self.connect)
        self.ptField.bind("<Return>", lambda event: self.sendReply())

    def sendReply(self):
        ptInput = self.ptField.getText()
        if ptInput != "":
            self.server.send(bytes(ptInput, CODE))
            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)
        self.server.connect(ADDRESS)
        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()
```

Figure 1.6 Source code of doctorclient.py.

doctorclienthandler.py

```python
"""
File: doctorclienthandler.py
Client handler for providing non-directive psychotherapy.
"""

from codecs import decode
from threading import Thread
from doctor import Doctor

BUFSIZE = 1024
CODE = "ascii"

class DoctorClientHandler(Thread):
    """Handles a session between a doctor and a patient."""
    def __init__(self, client):
        Thread.__init__(self)
        #print(str(client))
        self.client = client
        self.dr = Doctor()

    def run(self):
        self.client.send(bytes(self.dr.greeting(self.client),
                               CODE))
        while True:
            message = decode(self.client.recv(BUFSIZE),
                             CODE)

            if not message:
                self.dr.updateFile()
                print("Client disconnected")
                self.client.close()
                break
            else:
                self.client.send(bytes(self.dr.reply(message),
                                       CODE))

```

Figure 1.7 Source code of doctorclienthandler.py.

doctorserver.py

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

from socket import *
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()
```

Figure 1.8 Source code of doctorserver.py.

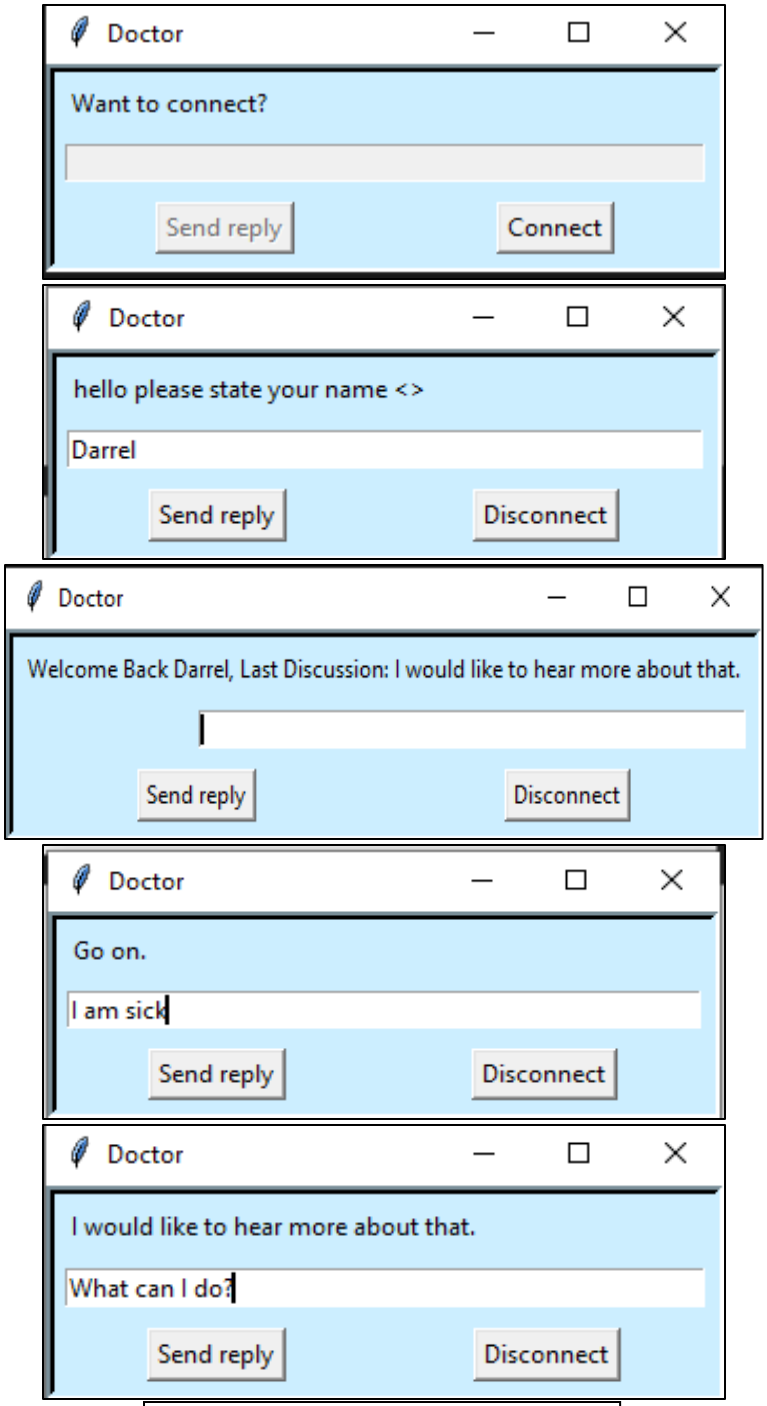


Figure 1.9 Sample run of the program.

This program uses GUI and applies the concept of threading. As you can see, this is an interactive program where a user can login and talk to a virtual doctor.

Machine Problem 3

chatclient.py ✕ chatserver.py ✕

chatserver.py > ...

```python
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()
```

Figure 1.10 Source code of chatserver.py.

chatclient.py ✕

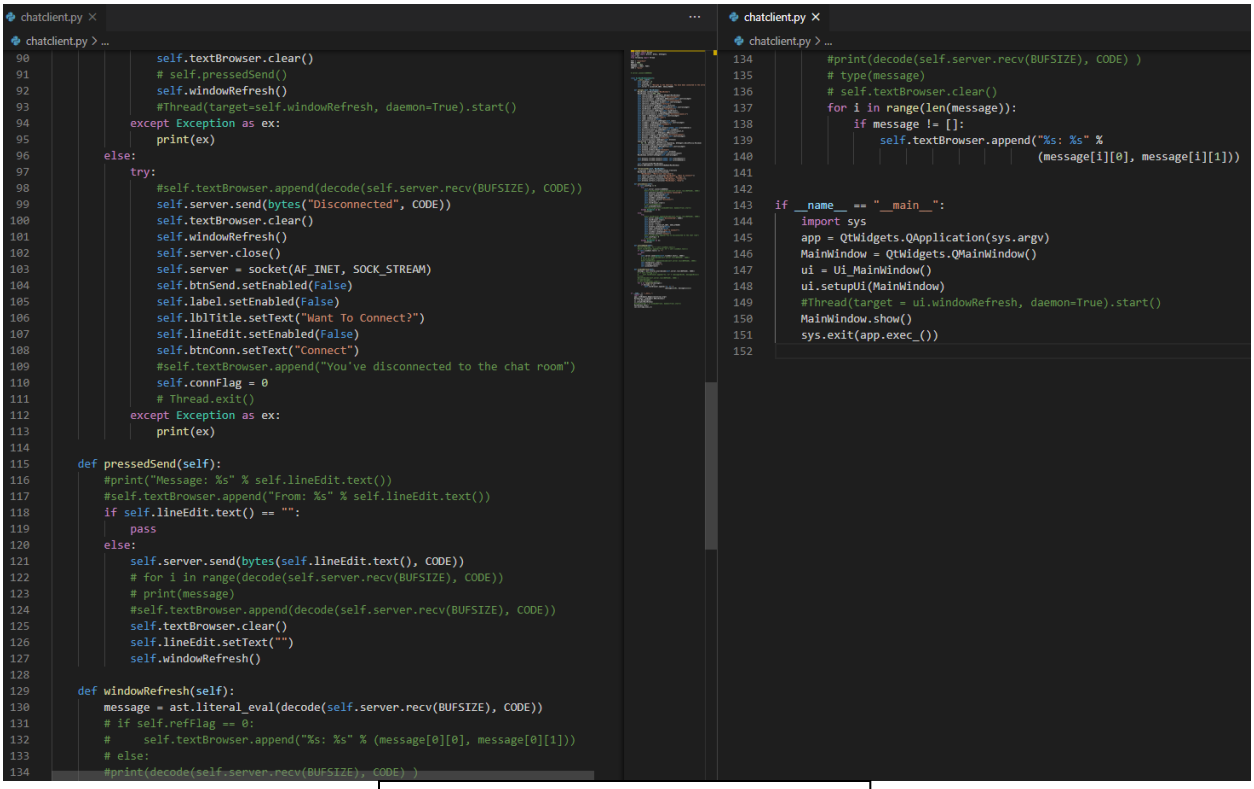
chatclient.py > ...

```python
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 serve
        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")
        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)
```

chatclient.py ✕

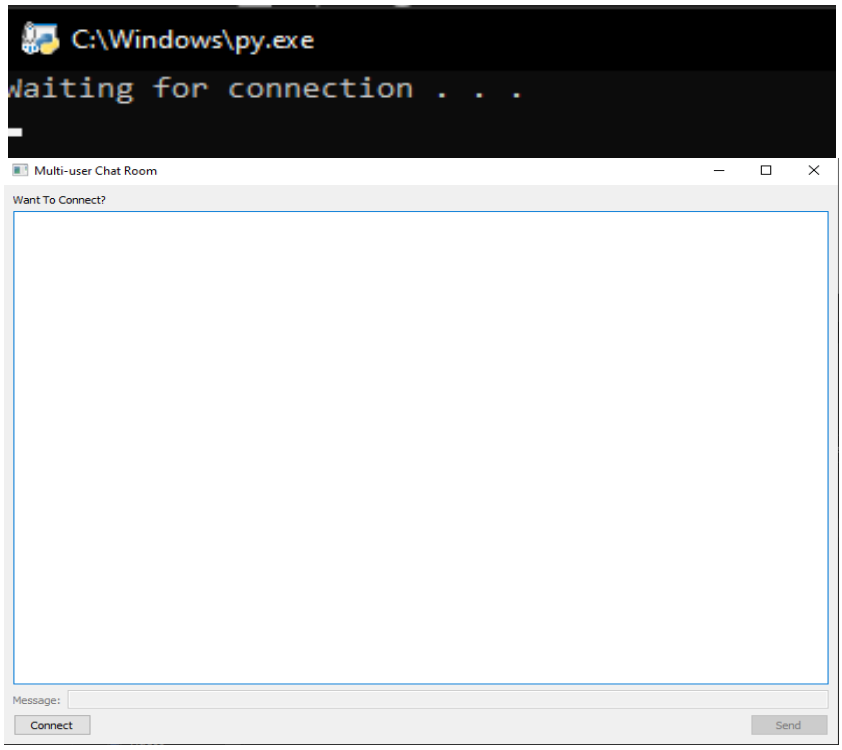
chatclient.py > ...

```python
        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.QSizePolicy.Minimum)
        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)
        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.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]))


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_())
```

Figure 1.11 Source code of chatclient.py.



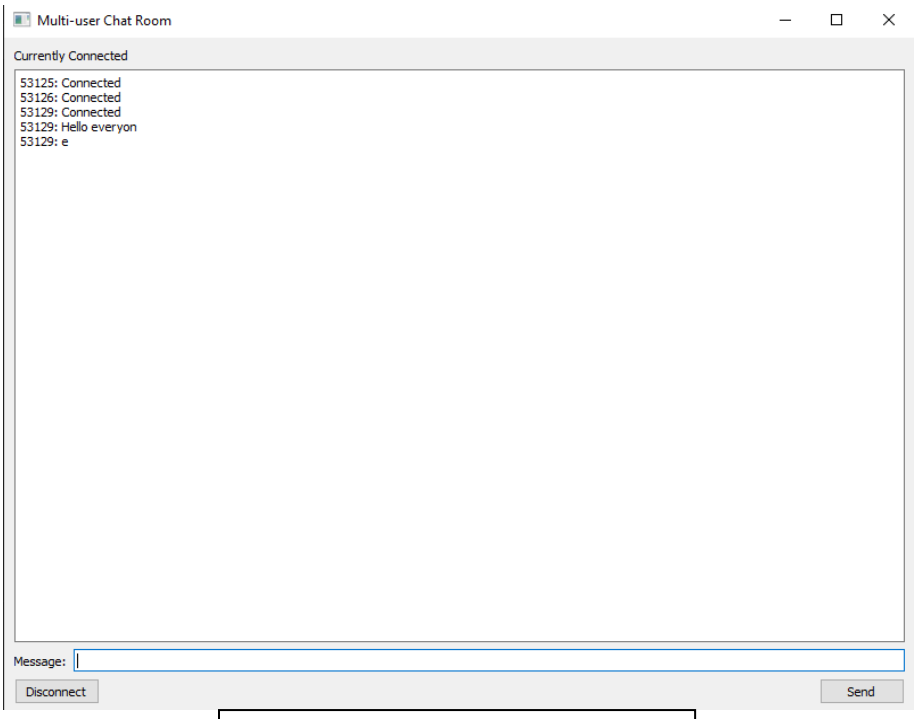


Figure 1.12 Sample run of the program.

This program also applies GUI and threading concept in python. The program can accommodate multiple users who wants to enter the chat room. It has a server so that the users can communicate between them. In the sample run, it shows that there are three users connected to the chat room. User 53125, 53126, and 53129. Everyone in the chat room will also be notified when someone disconnected.

PostLab

Machine Problem 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 <<<<. When labeled >>>> 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.

tempconverter.py

```python
import sys
from PyQt5 import QtCore, QtGui, QtWidgets, uic
from decimal import Decimal

qtcreator_file = 'TempConvert.ui'  # ui file here
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtcreator_file)


class MyWindow(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        self.FahConvert.clicked.connect(self.Fahconvert)
        self.CelConvert.clicked.connect(self.Celconvert)

    def Fahconvert(self):
        tempC = Decimal(self.Fahrenheit.toPlainText())
        tempC = round(((tempC)-32)*5/9, 3)
        self.Celsius.setPlainText(str(tempC))

    def Celconvert(self):
        tempF = Decimal(self.Celsius.toPlainText())
        tempF = round((tempF)*9/5+32, 3)

        self.Fahrenheit.setPlainText(str(tempF))


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyWindow()
    window.show()
    sys.exit(app.exec_())
```

Figure 2.1 Source code of tempconverter.py.

```python
import sys
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtWidgets import QMainWindow, QWidget, QLabel, QLineEdit
from PyQt5.QtWidgets import QPushButton
from PyQt5.QtCore import QSize

class MainWindow(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)

        self.setMinimumSize(QSize(240, 200))
        self.setWindowTitle("Temperature Converter")

        self.c_to_f_label = QLabel(self)
        self.c_to_f_label.setText('Celsius')
        self.c_to_f_label.move(40, 20)
        self.c_to_f_input = QLineEdit(self)

        self.c_to_f_input.move(30, 60)
        self.c_to_f_input.resize(70, 32)

        self.f_to_c_label = QLabel(self)
        self.f_to_c_label.setText('Fahrenheit')
        self.f_to_c_label.move(140, 20)
        self.f_to_c_input = QLineEdit(self)

        self.f_to_c_input.move(130, 60)
        self.f_to_c_input.resize(70, 32)

        self.setStyleSheet("QLabel {font: 12pt Arial}")
        self.c_to_f_input.setText('0')
        self.f_to_c_input.setText('32')

        pybutton1 = QPushButton('>>>>', self)
        pybutton1.clicked.connect(self.clickMethod1)
        pybutton1.resize(50,32)
        pybutton1.move(40, 100)

        pybutton2 = QPushButton('<<<<', self)
        pybutton2.clicked.connect(self.clickMethod2)
        pybutton2.resize(50,32)
        pybutton2.move(140, 100)

        Clear = QPushButton('Clear', self)
        Clear.clicked.connect(self.clickMethod3)
        Clear.resize(50,32)
        Clear.move(95, 140)

    def clickMethod1(self):
        self.f_to_c_input.setText(str(round(float(self.c_to_f_input.text())*9/5+32,2)))

    def clickMethod2(self):
        self.c_to_f_input.setText(str(round((float(self.f_to_c_input.text())-32)*5/9,2)))

    def clickMethod3(self):
        self.c_to_f_input.clear()
        self.f_to_c_input.clear()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    mainWin = MainWindow()
    mainWin.show()
    sys.exit( app.exec_() )
```

Figure 2.2 Source code of temp_converter.py.

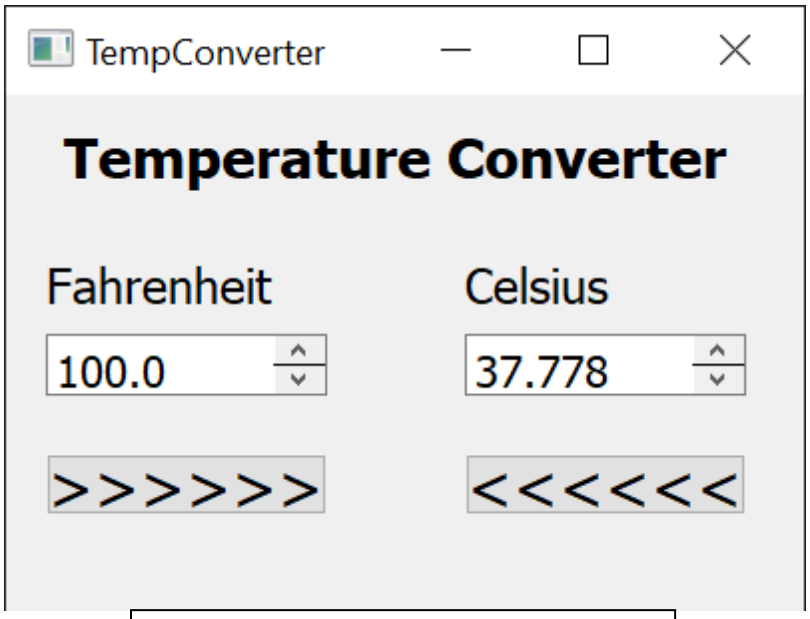


Figure 2.3 Sample run of Fahrenheit to Celsius.

The figure above shows the conversion of Fahrenheit to Celsius through a GUI based program, named TempConverter. To convert Fahrenheit to Celsius, the user must click the push button aligned below of the Fahrenheit column.

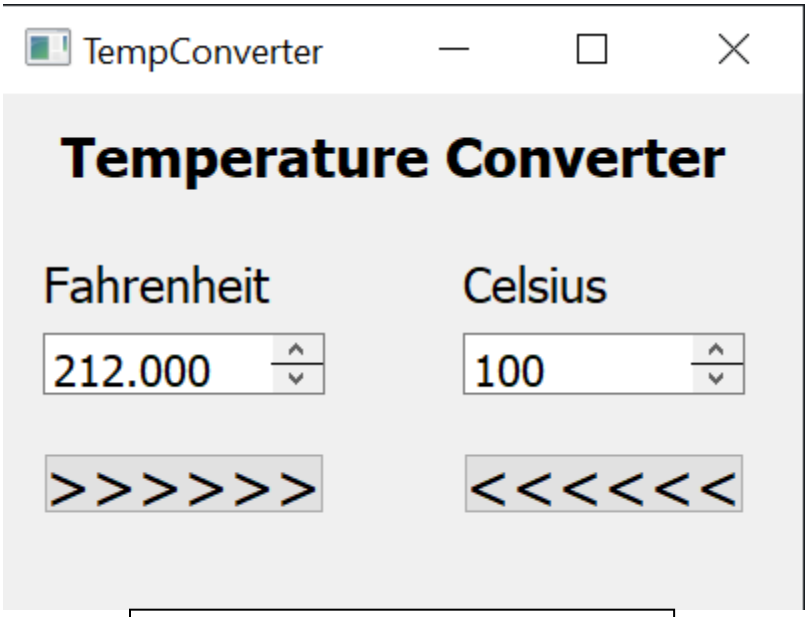
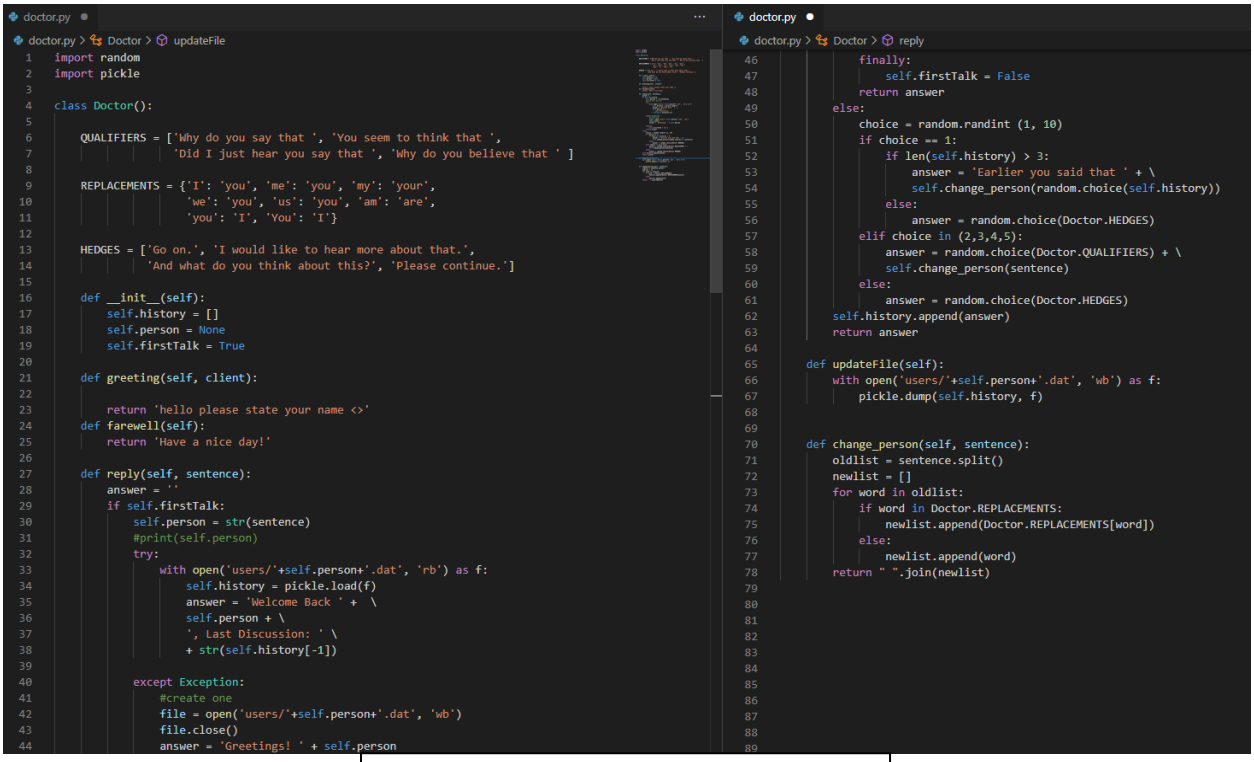


Figure 2.4 Sample run of Celsius to Fahrenheit.

This figure shows the conversion of Celsius to Fahrenheit through the same GUI based program shown in Figure 13. To convert Celsius to Fahrenheit, the user must click the push button aligned below of the Celsius column.

Machine Problem 2



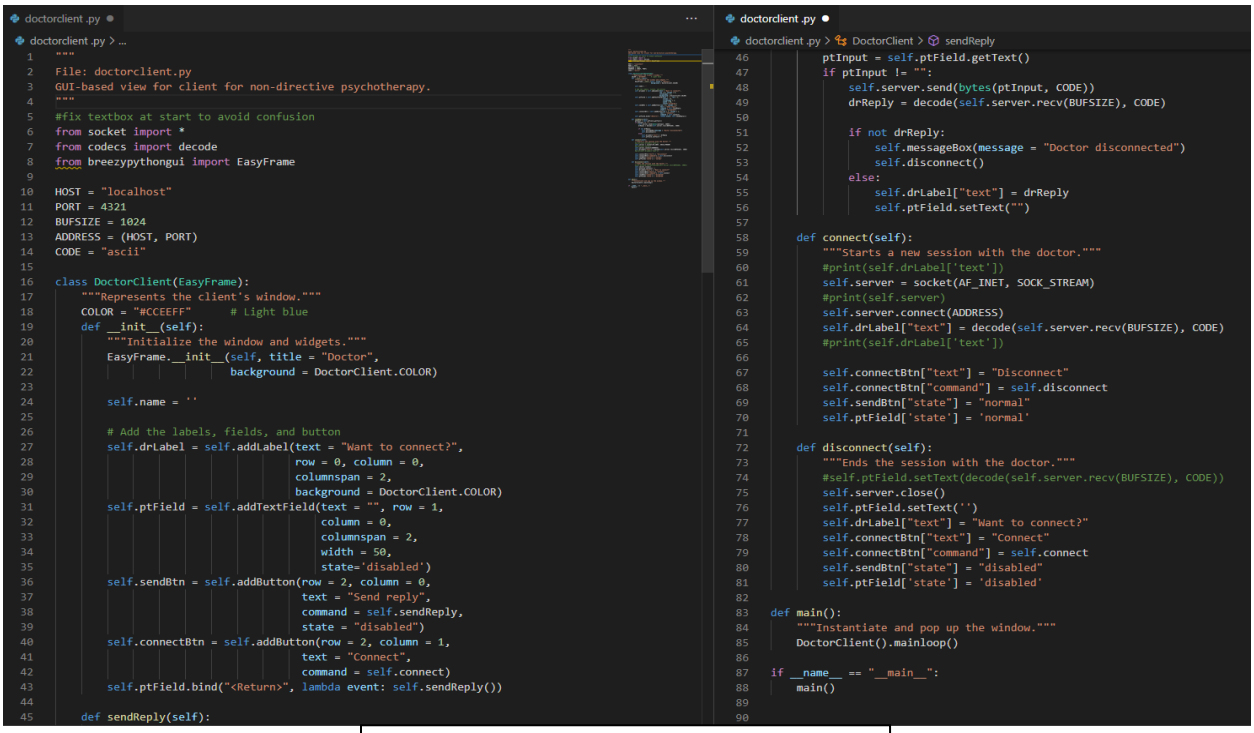
doctor.py

```
import random
import pickle

class Doctor():

    QUALIFIERS = ['Why do you say that ', 'You seem to think that ',
                  'Did I just hear you say that ', 'Why do you believe that ' ]

    REPLACEMENTS = {'I': 'you', 'me': 'you', 'my': 'your',
                    'we': 'you', 'us': 'you', 'am': 'are',
                    'you': 'I', 'You': 'I'}

    HEDGES = ['Go on.', 'I would like to hear more about that.',
              'And what do you think about this?', 'Please continue.']

    def __init__(self):
        self.history = []
        self.person = None
        self.firstTalk = True

    def greeting(self, client):

        return 'hello please state your name <>'
    def farewell(self):
        return 'Have a nice day!'

    def reply(self, sentence):
        answer = ''
        if self.firstTalk:
            self.person = str(sentence)
            #print(self.person)
            try:
                with open('users/'+self.person+'.dat', 'rb') as f:
                    self.history = pickle.load(f)
                    answer = 'Welcome Back ' +  \
                    self.person + \
                    ', Last Discussion: ' \
                    + str(self.history[-1])

            except Exception:
                #create one
                file = open('users/'+self.person+'.dat', 'wb')
                file.close()
                answer = 'Greetings! ' + self.person
```

doctor.py

```
                finally:
                    self.firstTalk = False
                return answer
            else:
                choice = random.randint (1, 10)
                if choice == 1:
                    if len(self.history) > 3:
                        answer = 'Earlier you said that ' + \
                        self.change_person(random.choice(self.history))
                    else:
                        answer = random.choice(Doctor.HEDGES)
                elif choice in (2,3,4,5):
                    answer = random.choice(Doctor.QUALIFIERS) + \
                    self.change_person(sentence)
                else:
                    answer = random.choice(Doctor.HEDGES)
            self.history.append(answer)
            return answer

        def updateFile(self):
            with open('users/'+self.person+'.dat', 'wb') as f:
                pickle.dump(self.history, f)


        def change_person(self, sentence):
            oldlist = sentence.split()
            newlist = []
            for word in oldlist:
                if word in Doctor.REPLACEMENTS:
                    newlist.append(Doctor.REPLACEMENTS[word])
                else:
                    newlist.append(word)
            return " ".join(newlist)
```

Figure 2.5 Source code of tempconverter.py.



doctorclient.py

```
"""
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,
                                         state='disabled')
        self.sendBtn = self.addButton(row = 2, column = 0,
                                      text = "Send reply",
                                      command = self.sendReply,
                                      state = "disabled")
        self.connectBtn = self.addButton(row = 2, column = 1,
                                         text = "Connect",
                                         command = self.connect)
        self.ptField.bind("<Return>", lambda event: self.sendReply())

    def sendReply(self):
```

doctorclient.py

```
        ptInput = self.ptField.getText()
        if ptInput != "":
            self.server.send(bytes(ptInput, CODE))
            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)
        self.server.connect(ADDRESS)
        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()
```

Figure 2.6 Source code of tempconverter.py.

doctorclienthandler.py

```python
"""
File: doctorclienthandler.py
Client handler for providing non-directive psychotherapy.
"""

from codecs import decode
from threading import Thread
from doctor import Doctor

BUFSIZE = 1024
CODE = "ascii"

class DoctorClientHandler(Thread):
    """Handles a session between a doctor and a patient."""
    def __init__(self, client):
        Thread.__init__(self)
        #print(str(client))
        self.client = client
        self.dr = Doctor()

    def run(self):
        self.client.send(bytes(self.dr.greeting(self.client),
                               CODE))
        while True:
            message = decode(self.client.recv(BUFSIZE),
                             CODE)

            if not message:
                self.dr.updateFile()
                print("Client disconnected")
                self.client.close()
                break
            else:
                self.client.send(bytes(self.dr.reply(message),
                                       CODE))

```

Figure 2.7 Source code of tempconverter.py.

doctorserver.py

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

from socket import *
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()
```

Figure 2.8 Source code of tempconverter.py.

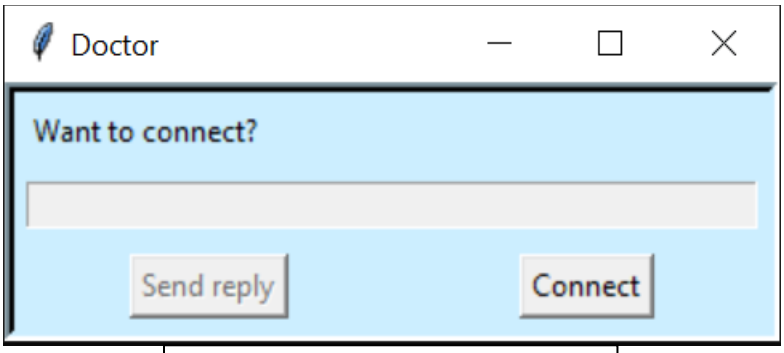


Figure 2.9 Start-up Doctor Window.

This figure shows the starting window of the Doctor App where the patient or user would connect prior to talking to the automated Doctor in the program.

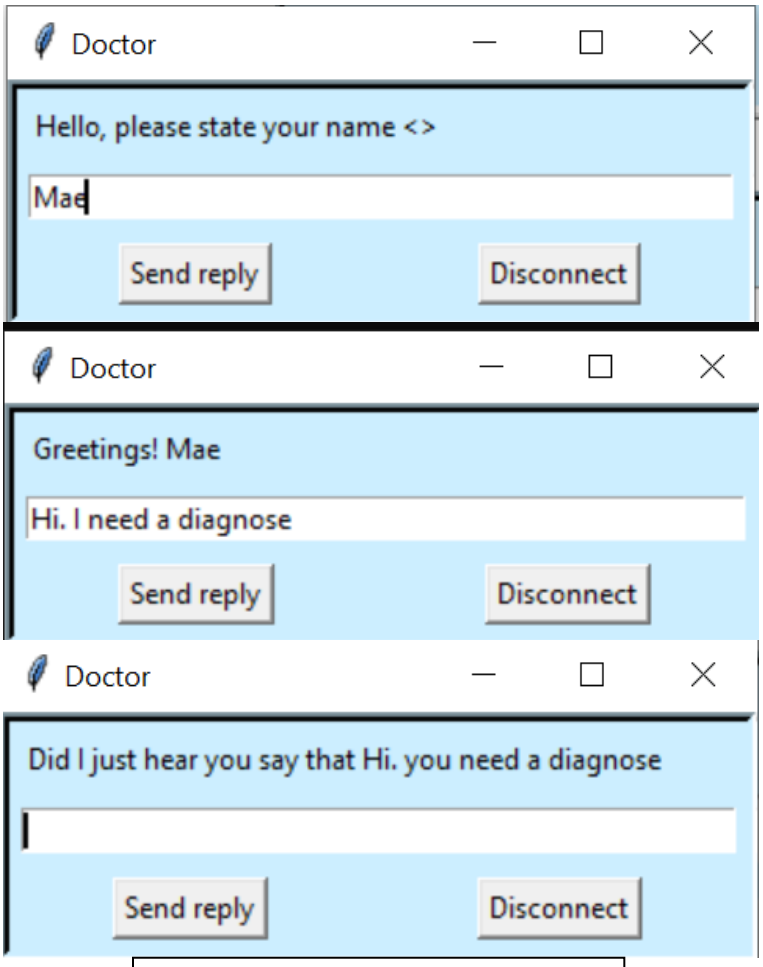


Figure 2.10 First run of the Doctor App.

In the first run of the Doctor App, the patient has connected to the app and holds a conversation with a Doctor that automatically replies to its patient.

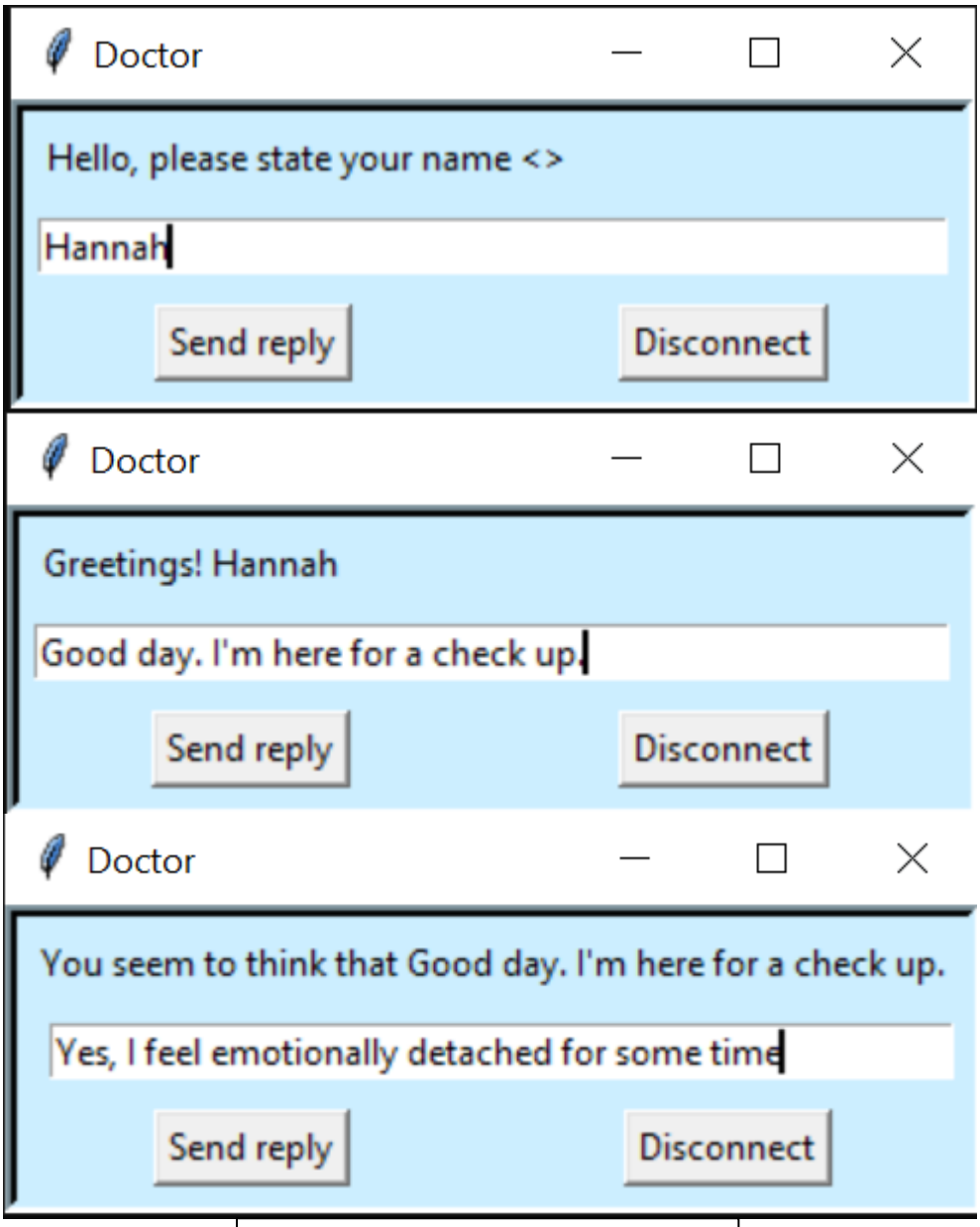


Figure 2.11 Second run of the Doctor App.

In the second run of the Doctor App, a new patient has connected to the app and holds a new conversation with the Doctor.

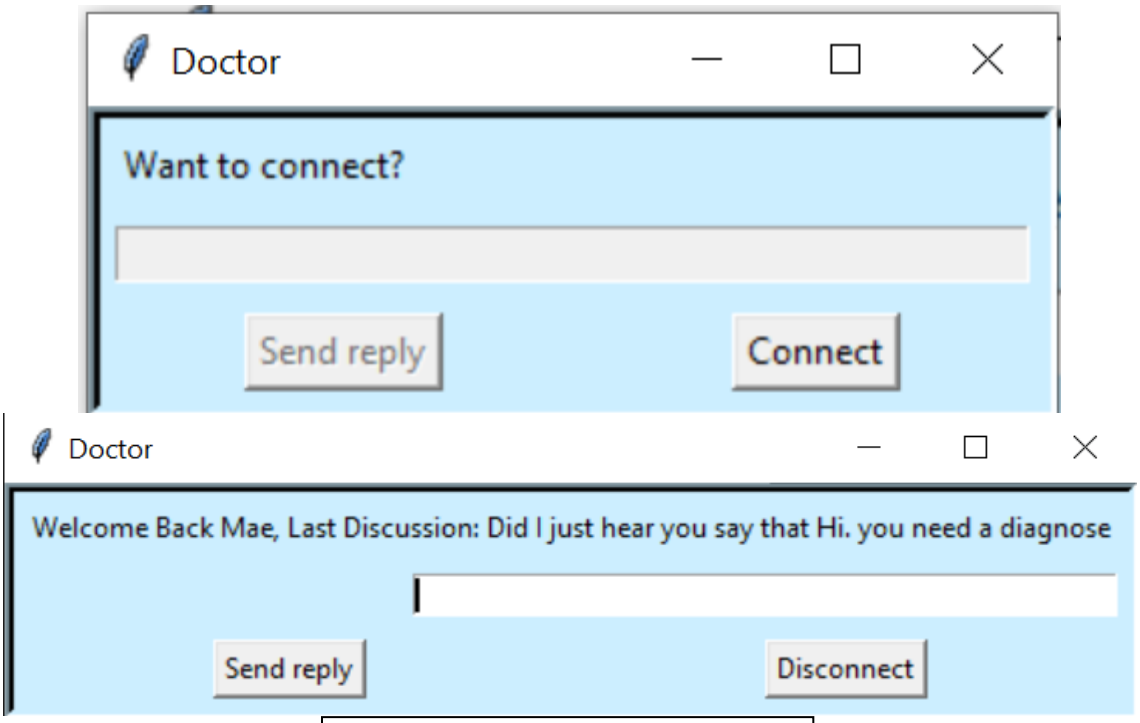


Figure 2.12 Third run of the Doctor App.

In the third run of the Doctor App where the second patient has disconnected, the first patient which was Mae, has connected to the app again with the program knowing that Mae already held a conversation before. Here, the Doctor has brought up the last conversation they've had before Mae disconnected in the first run.

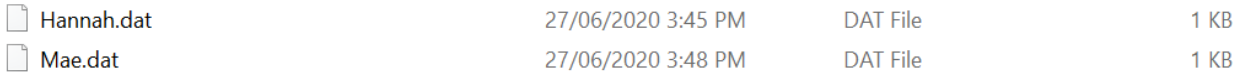


Figure 2.13 Conversation Files.

This figure shows the held conversations of the two patients that have connected to the Doctor App. The program has recognized that Mae was a previous patient because it held to its program a history of their conversation.

Machine Problem 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 conversation 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 display 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 the ATM application of Project 8. The server creates a thread-safe transcript at start-up, listens form 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 the 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 client'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 list of strings, while __str__ returns the join of the list, separated by newlines.

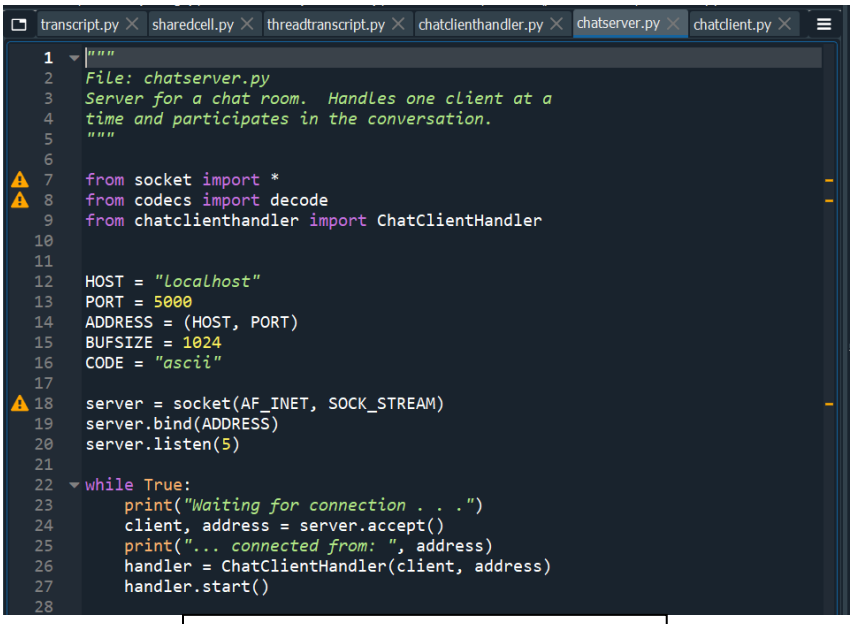
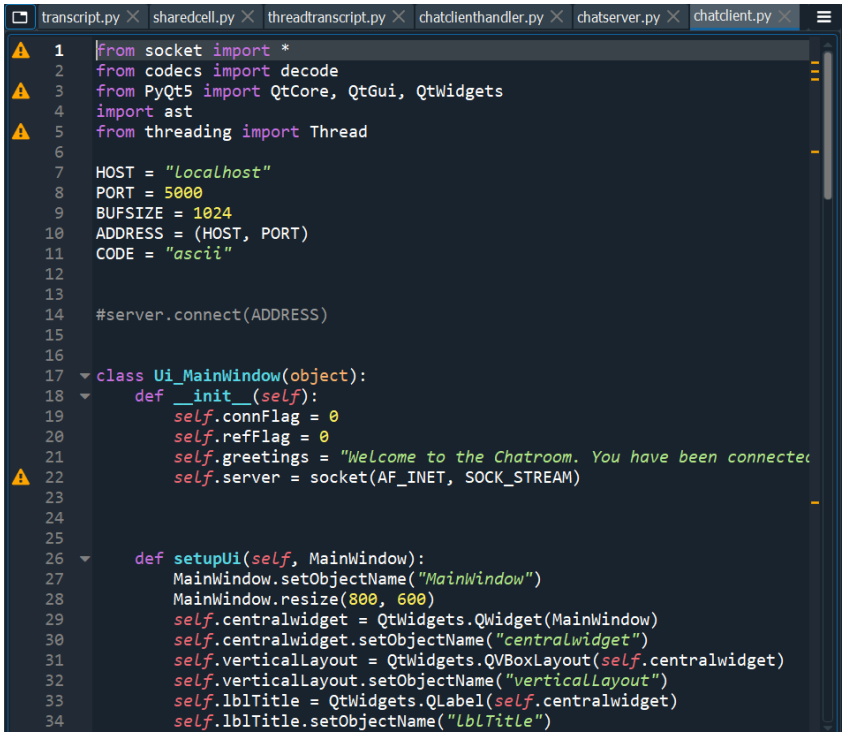


Figure 2.14 Source code of chatserver.py.

The server side script will attempt to establish a socket and bind it to an IP address and port specified by the user. The script will then stay open and receive connection requests and will append respective socket objects to a list to keep track of active connections. Every time a user connects, a separate thread will be created for that user. In each thread, the server awaits a message, and sends that message to other users currently on the chat. If the server encounters an error while trying to receive a message from a particular thread, it will exit that thread.



```
        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.E
        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)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Multi-user Cha
        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),
                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),
                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.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) )
            #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_())
```

Figure 2.15 Source code of chatclient.py.

The client side script will simply attempt to access the server socket created at the specified IP address and port. Once it connects, it will continuously check as to whether the input comes from the server or from the client, and accordingly redirects output. If the input is from the server, it displays the message on the terminal. But if the message is from the user, it sends the message that the user enters to the server for it to be broadcasted to the other users.

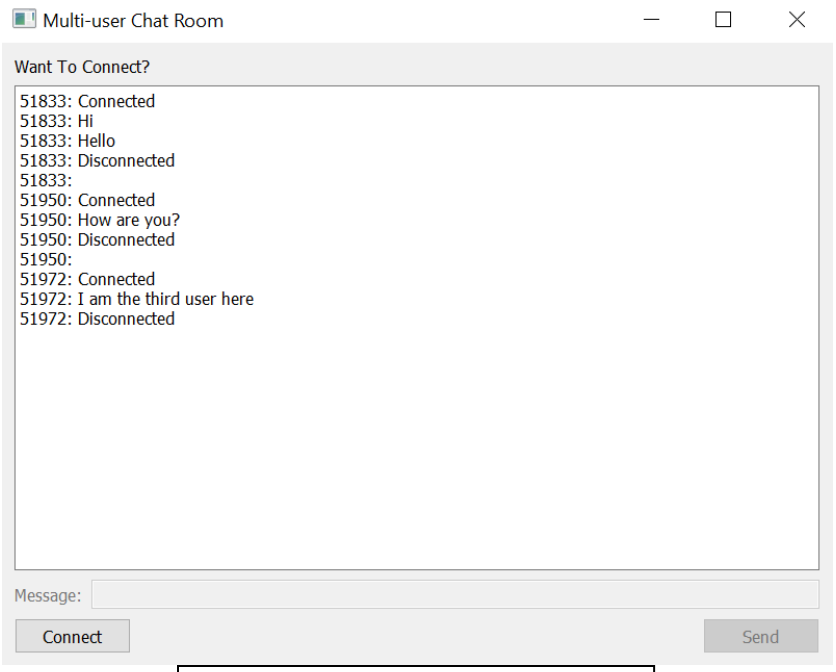


Figure 2.16 Sample run of the program.

This figure shows the sample run to test the program. In here, basic conversation between three users on the same server is observed which confirms the functionality of the program as it allows multiple users to connect to the server and chat with all other online users. Furthermore, it works in a broadcast fashion which means that messages from a user are broadcasted to other users.

OneDrive:
Group 1: https://bit.ly/3ijdvbA

Github:
Darrel Virtusio: https://bit.ly/2YBaFXm
Hannah Antaran: https://bit.ly/2Vpk7ey
Kathleen Tupas: https://bit.ly/2BJxWxp