**Software Design Laboratory**

Laboratory Report #6: Event Driven and Concurrent Programming

Content:

1. PreLab

**A**. Readings, Observation and Conclusion

**Threads:**

Fundamentals of Python: First Programs, Lambert (9781337671019): Chapter 10

**Observation**

The thread, as with any object in Python, may hold data, be stored in data structures, and be supplied as arguments to methods. Some code declared on a thread, however, can also be run as a process. A thread's class must implement a run method in order for this code to be executed. A thread can be in several states at the same time. Figure 10-1 depicts some of the states that a Python thread can be in during its lifespan. The box labeled "The ready queue" in this diagram represents a data structure, whereas the box labeled "The CPU" is a hardware resource. The labeled ovals represent the thread states. A thread remains dormant when it is formed until its start function is called. Running this method also "readys" the thread and adds a reference to it to the ready queue. A queue is a data structure that governs access to a single resource on a first-come, first-served basis. In this situation, the resource is the CPU, which can only execute the instructions of one thread at a time. The run method of a freshly launched thread is likewise active. However, before it can execute its first instruction, the thread must wait its turn in the ready queue.

* Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
* Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.

* It can be pre-empted (interrupted)
* It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

**Conclusion**

A thread is a fundamental of any application that the operating system manages. The operating system accomplishes parallelism or multitasking by partitioning the process into threads. It is a simple procedure that provides a distinct execution path. Because these threads may be operated really concurrently, multithreaded applications can run quicker on computer systems with several CPUs. A software can continue to respond to input. This is applicable for both single and many CPUs. Threads in a process can share global variable memory. If a global variable is modified in one thread, the change affects all threads. Local variables can exist in a thread. A thread of execution is the shortest series of programmed instructions that may be controlled independently by a scheduler, which is usually a component of the operating system. A thread is an execution route in a computer program. Threading adds new independent execution routes to your software. Every program begins with one or more execution paths/threads. Depending on your needs, you can generate more threads to do concurrent activities. It is a concept of resource usage that is efficient. Having several threads in an application has two significant potential benefits:

* Enhance the perceived responsiveness of an application.
* Improve the real-time performance of an application on multicore computers.

There are libraries that can help. It should be noted that Threading necessitates precise synchronization to avoid race circumstances.

Graphical User Interface:Fundamentals of Python: First Programs, Lambert (9781337671019): Chapter 8

Websites:  
• https://doc.qt.io/qtforpython/  
• <https://docs.python.org/3/library/threading.html>

Observation

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

* Import the Tkinter module.
* Create the GUI application main window.
* Add one or more of the above-mentioned widgets to the GUI application.
* Enter the main event loop to take action against each event triggered by the user.

**Conclusion**

Modern computer applications are user-friendly7. User interaction is not restricted to console-based I/O. They have a more ergonomic graphical user interface (GUI) thanks to high speed processors and powerful graphics hardware. These applications can receive inputs through mouse clicks and can enable the user to choose from alternatives with the help of radio buttons, dropdown lists, and other GUI elements (or widgets).

Such applications are developed using one of various graphics libraries available. A graphics library is a software toolkit having a collection of classes that define a functionality of various GUI elements. These graphics libraries are generally written in C/C++.

**B. Answer to Questions**

Short Answer  
1. Explain what happens when a user clicks a command button in a fully functioning GUIprogram.

Answer:

When we click a button, the command will be executed and then it calls the function in command.

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

Answer:

1. Windows components are often laid and tested before adding to the computation in perform of events so that its feel and look can be checked individually and the corresponding event handler’s performance can also be reviewed.
2. If the text field is used for input and output of numbers, the data needs to be converted to string after the input is stored and also before output is produced. To simplify this task of the compiler, the integratedField and floatField are used which provides a value attribute in advance to the data.
3. The correct code for displaying this is

self.add FloatField(initial value, row grid value, column grid value, width, precision, type)

here initial value =(0,0) row grid value = 1, column grid value = 1

width = 15, precision= 2, Therefore, the codes becomes

self.addFloatField(value=0,0, row=1,column =1, width = 15, precision = 2);

1. The only difference between the integerField and floatField is the amount of characters it can take. Field can take width of 10 whereas FloatField has an additional precision element. The data is processed equally by both of them.

Multiple Choice  
Lambert, Chapter 10, Review Questions 1 to 10

1. B. multiprocessing

2. C. its start method is called

3. A. the start method

4. B. condition class

5. B. synchronization problems

6. C. IP address

7. C. IP Address

8. B. socket

9. B. strings

10. B. create a separate client-handler thread for each client

2. InLab

**A. Objectives** (You can state your own objectives based on the readings in Prelab)

**B. Steps Performed with screenshots of tools used:**

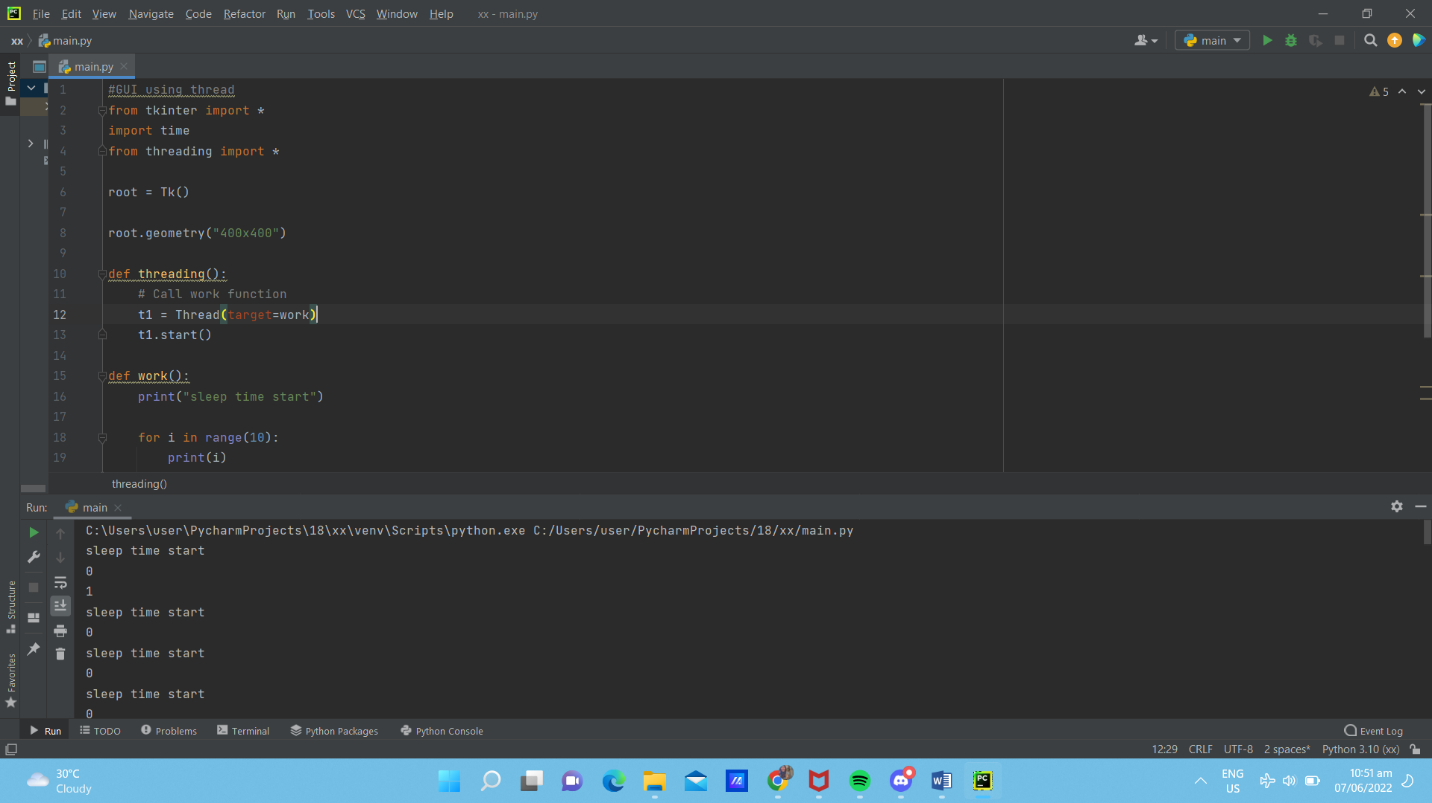
• Visual Studio Code, PyCharm, Atom or Spyder

• Git,

• QtDesigner

C.Sample run with DISCUSSIONS (DON’T copy and paste fromthe ebook). Use the Source Code of Lambert.

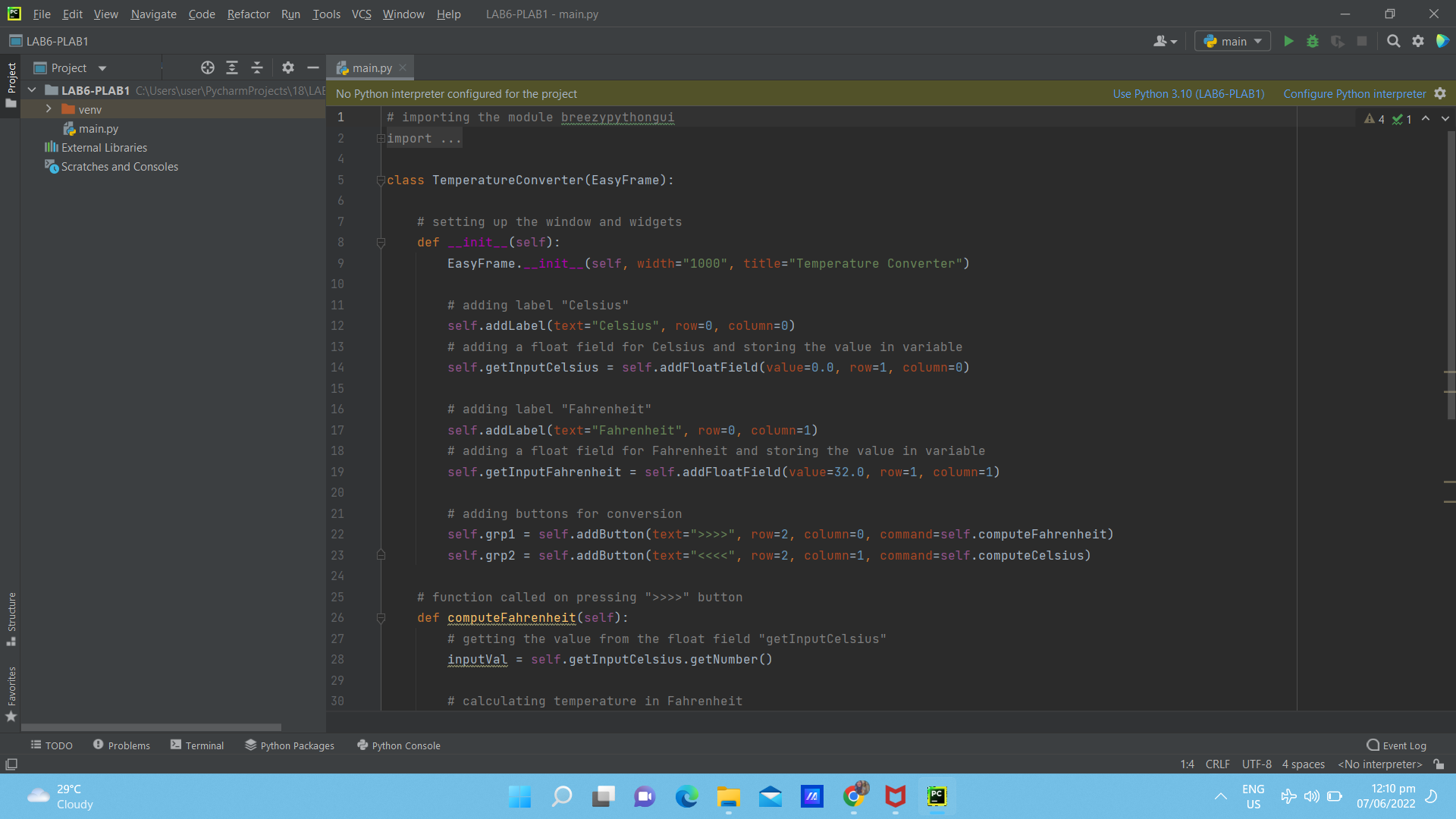
IMPORTANT: Include figure numbers and labels. Edit your screengrabs

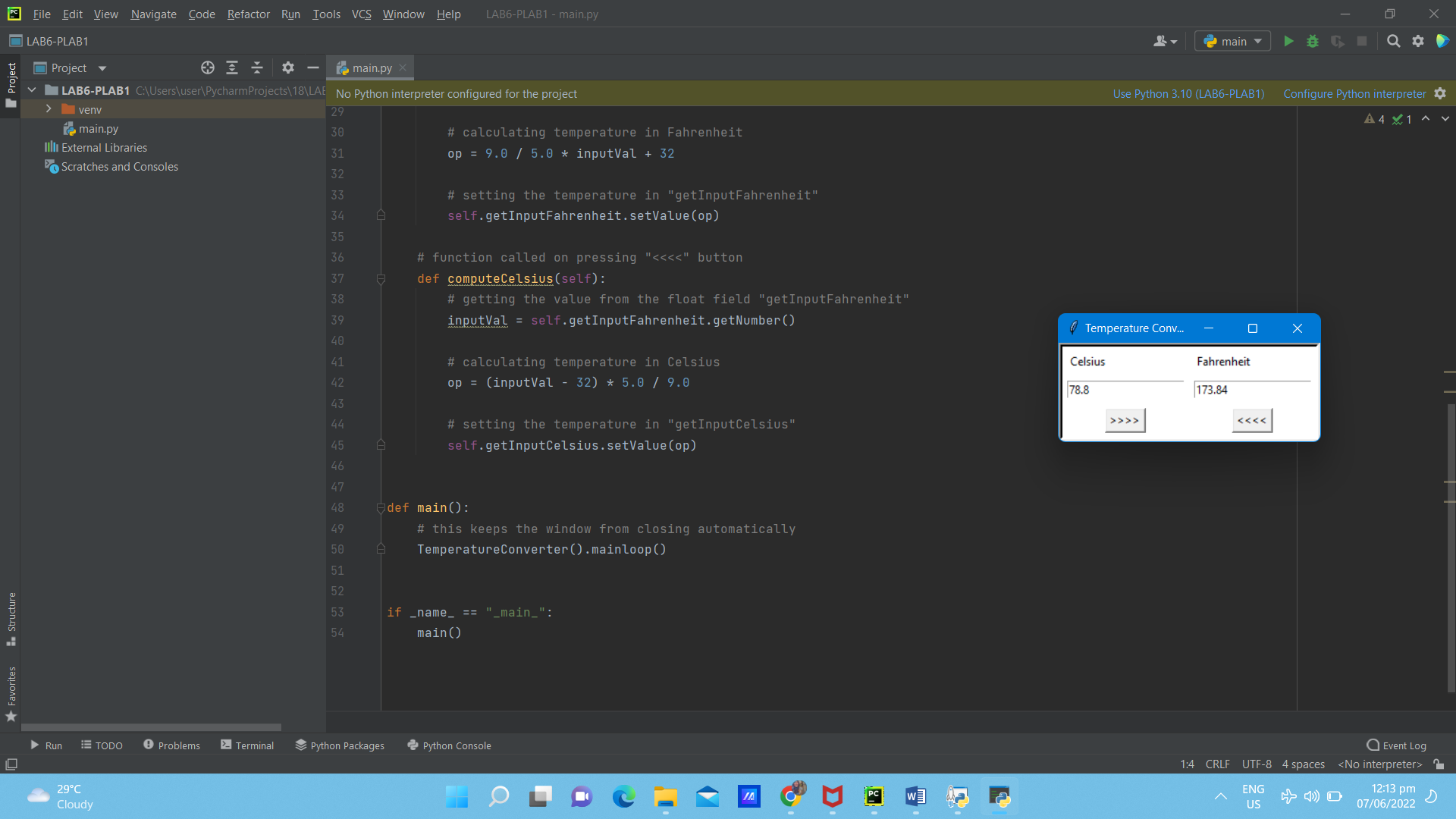


3. PostLab

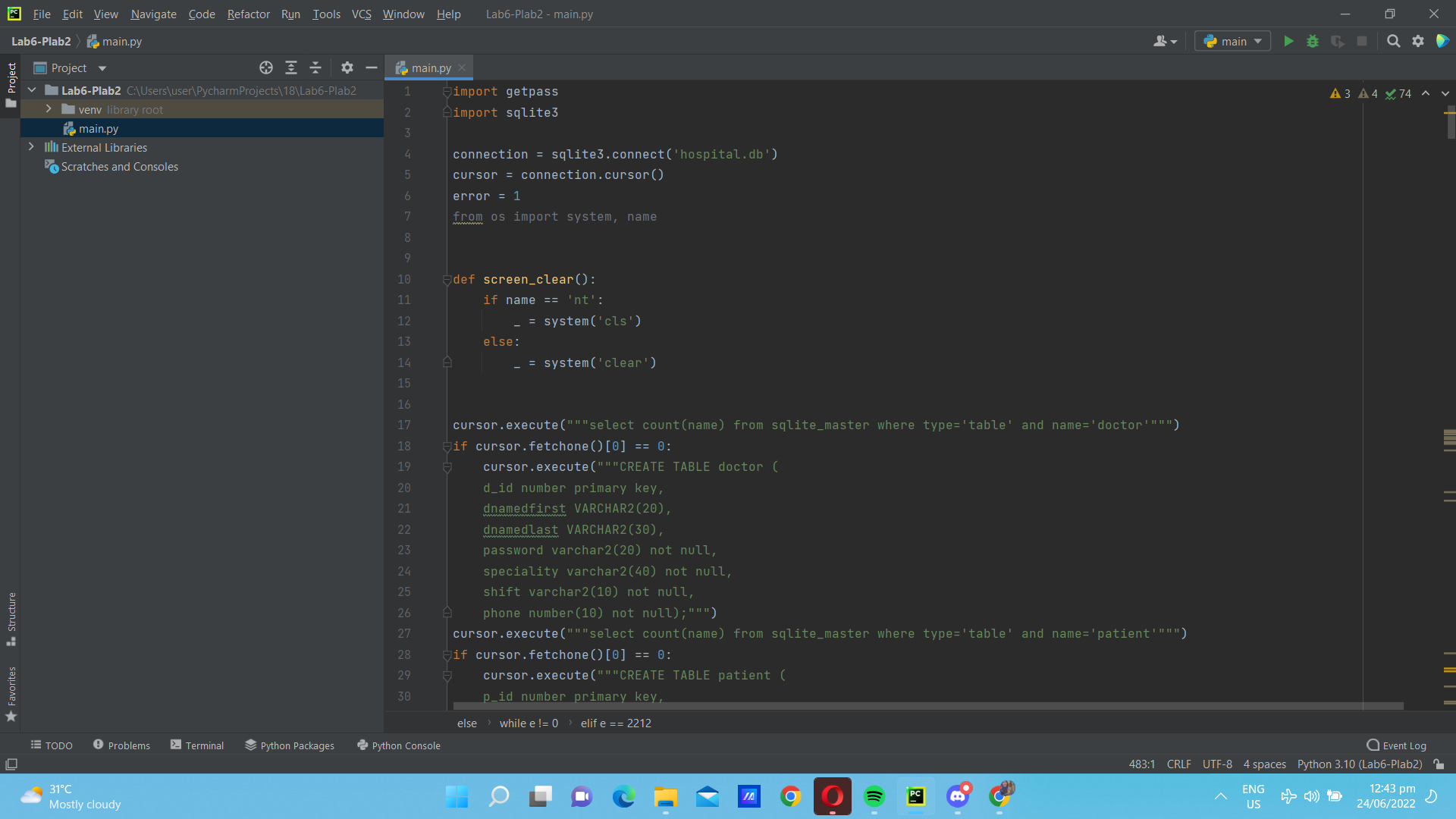
A. Machine Problems

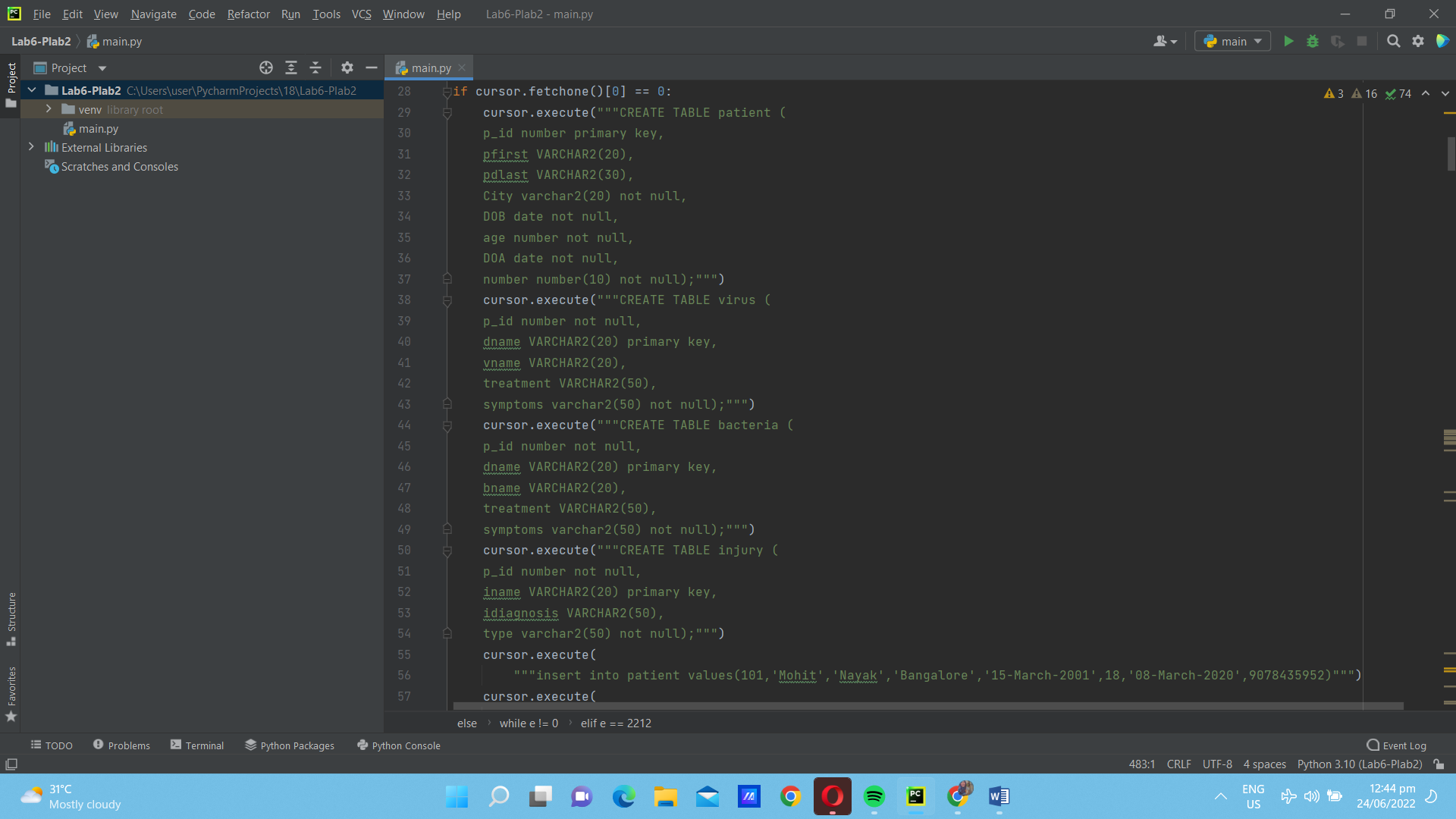
1. Write a GUI-based program that allows the user to convert temperature values between degrees Fahrenheit and degrees Celsius. The interface should have labeled entry fields for these two values. These components should be arranged in a grid where the labels occupy the first row and the corresponding fields occupy the second row. At start-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, labeled >>>> and <<<<. When 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 second button should perform the inverse function.

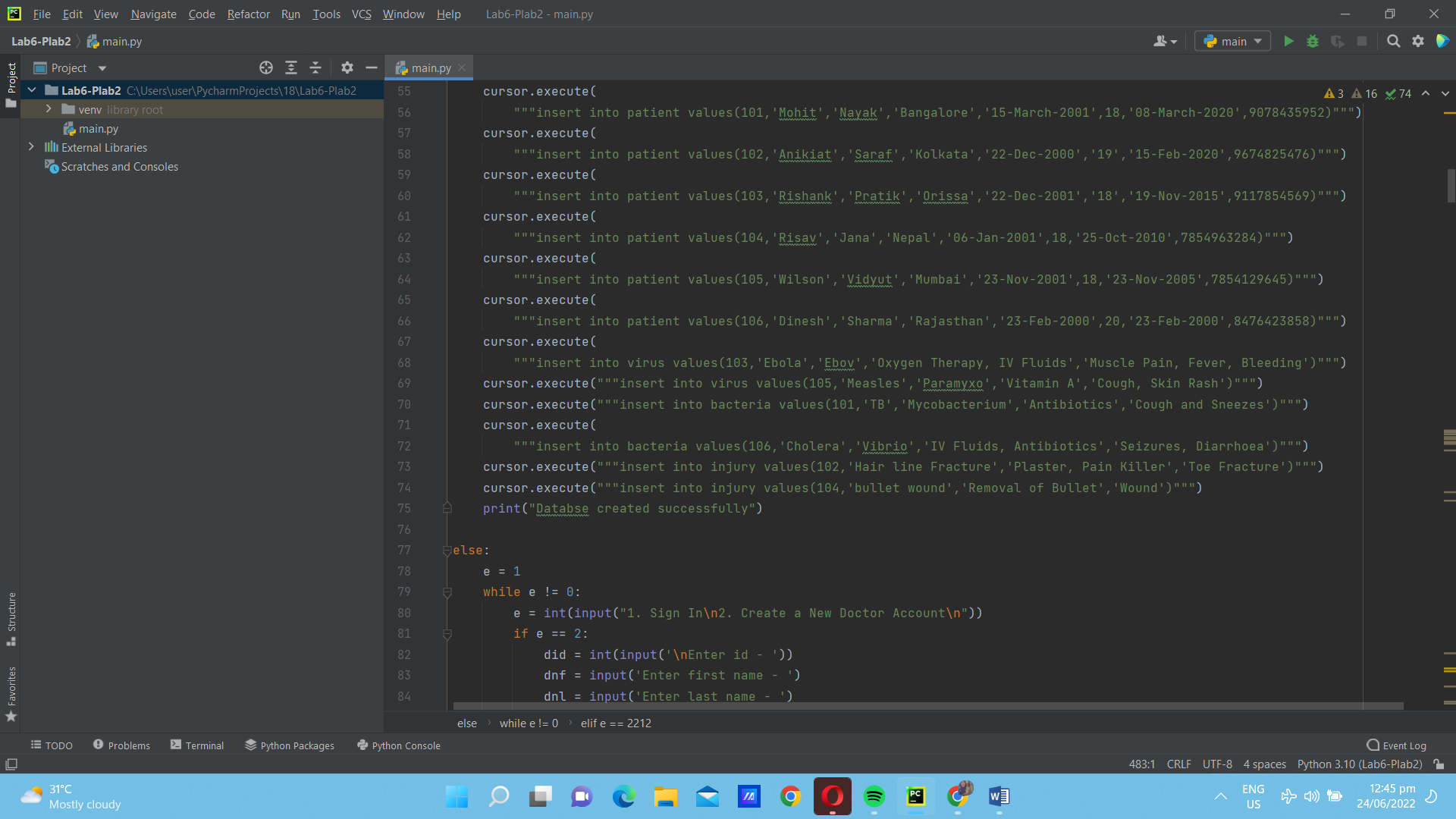


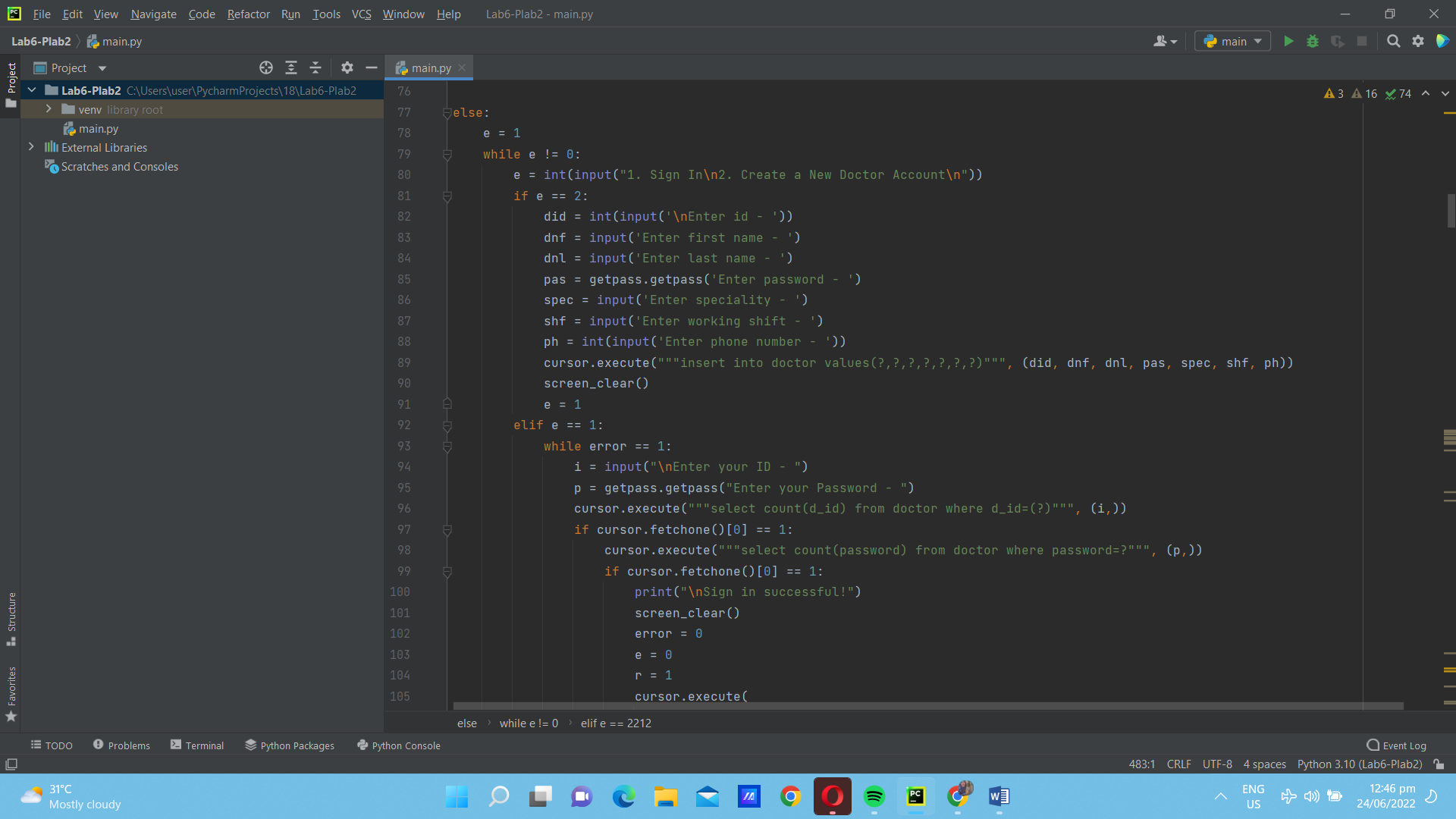


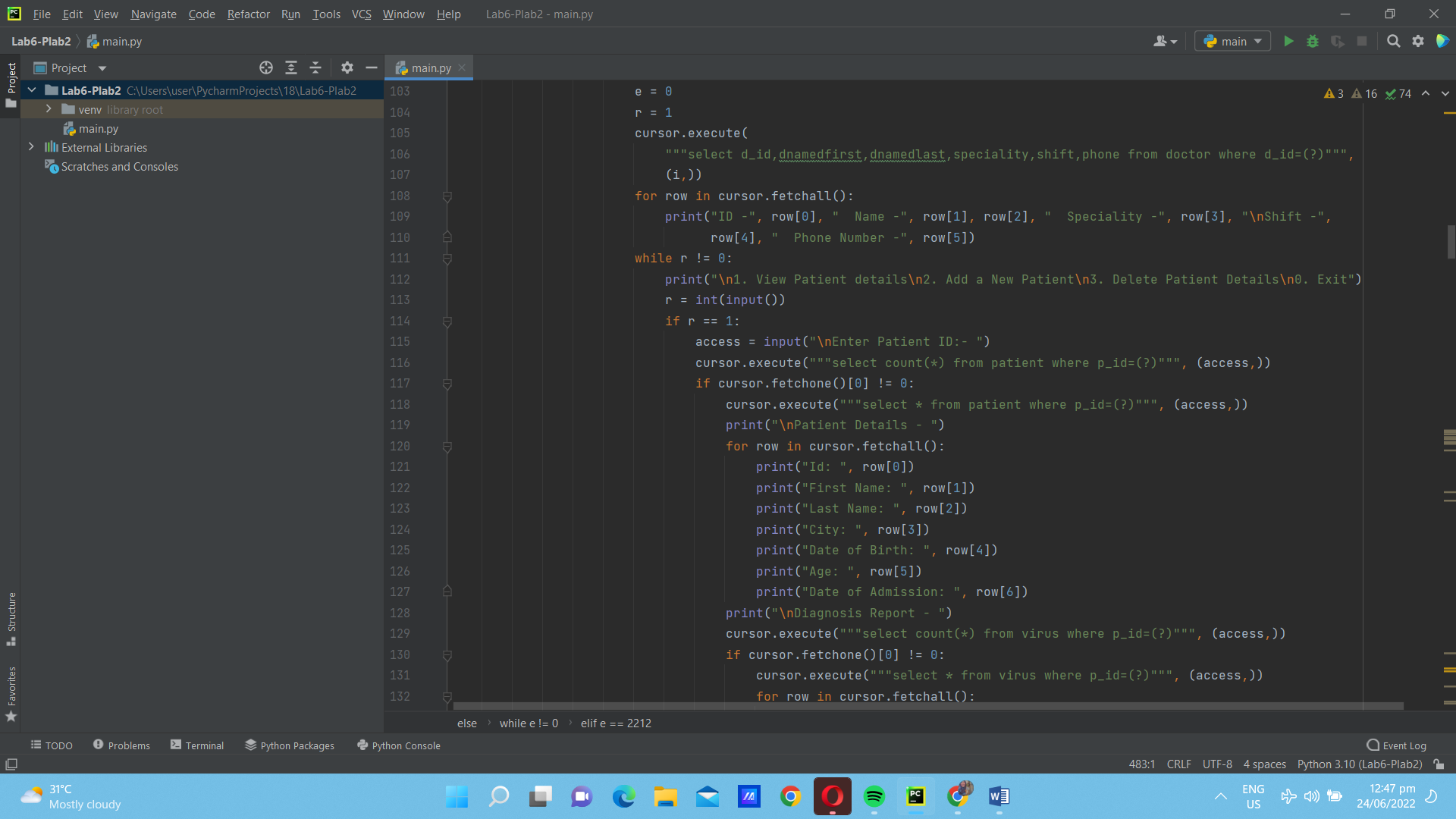
2. Modify the doctor application discussed in this chapter so that it tracks clients by name and history. A Doctor object has its own history list of a patient’s inputs for generating replies that refer to earlier conversations, as discussed in Chapter 5. A Doctor object is now associated with a patient’s name. The client application takes this name as input and sends it to the client handler when the patient connects. The client handler checks for a pickled file with the patient’s name as its filename (“.dat”). If that file exists, it will contain the patient’s his- tory, and the client handler loads the file to create the Doctor object. Otherwise, the patient is visiting the doctor for the first time, so the client handler creates a brand-new Doctor object. When the client disconnects, the client handler pickles the Doctor object in a file with the patient’s name.



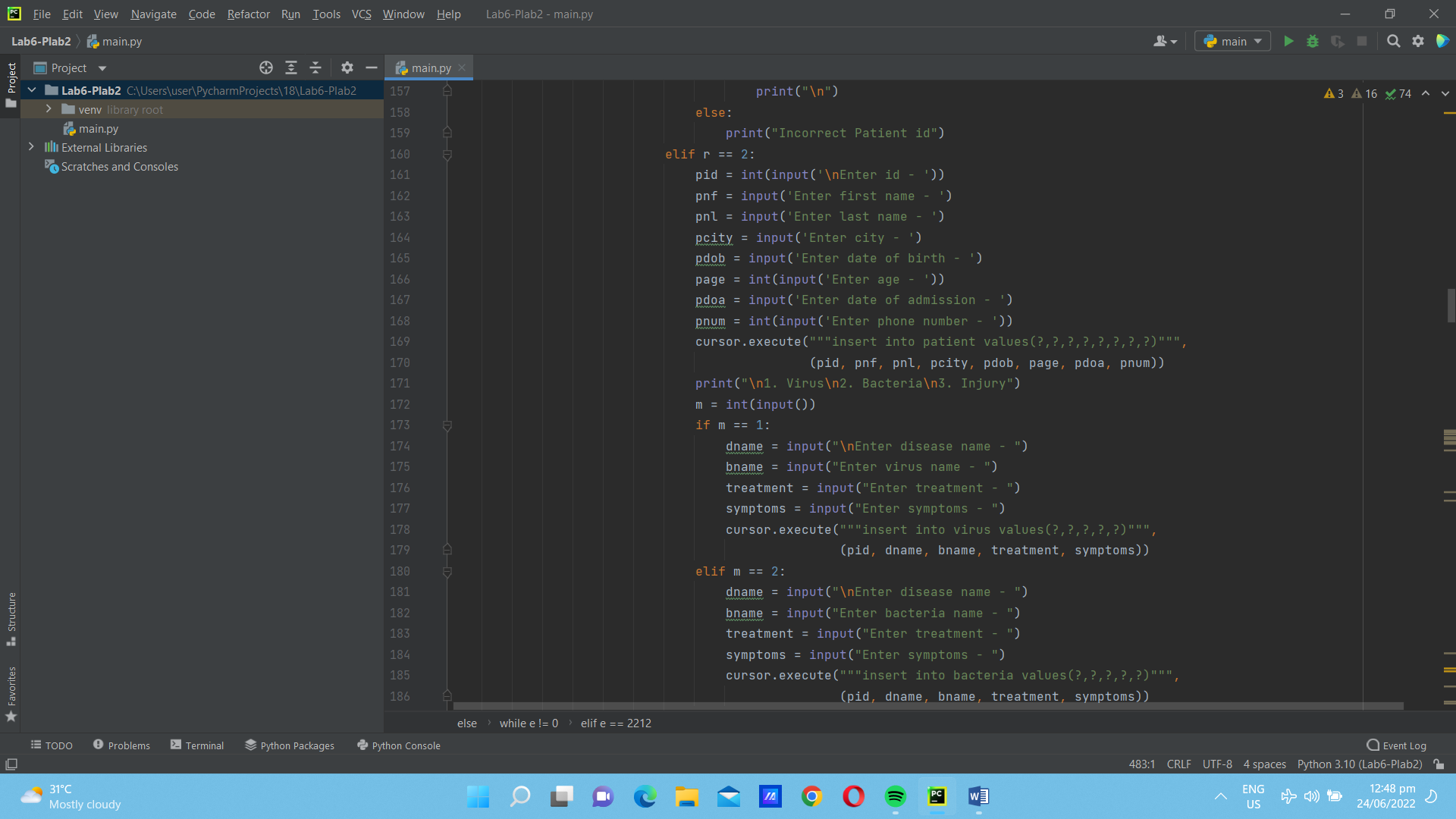


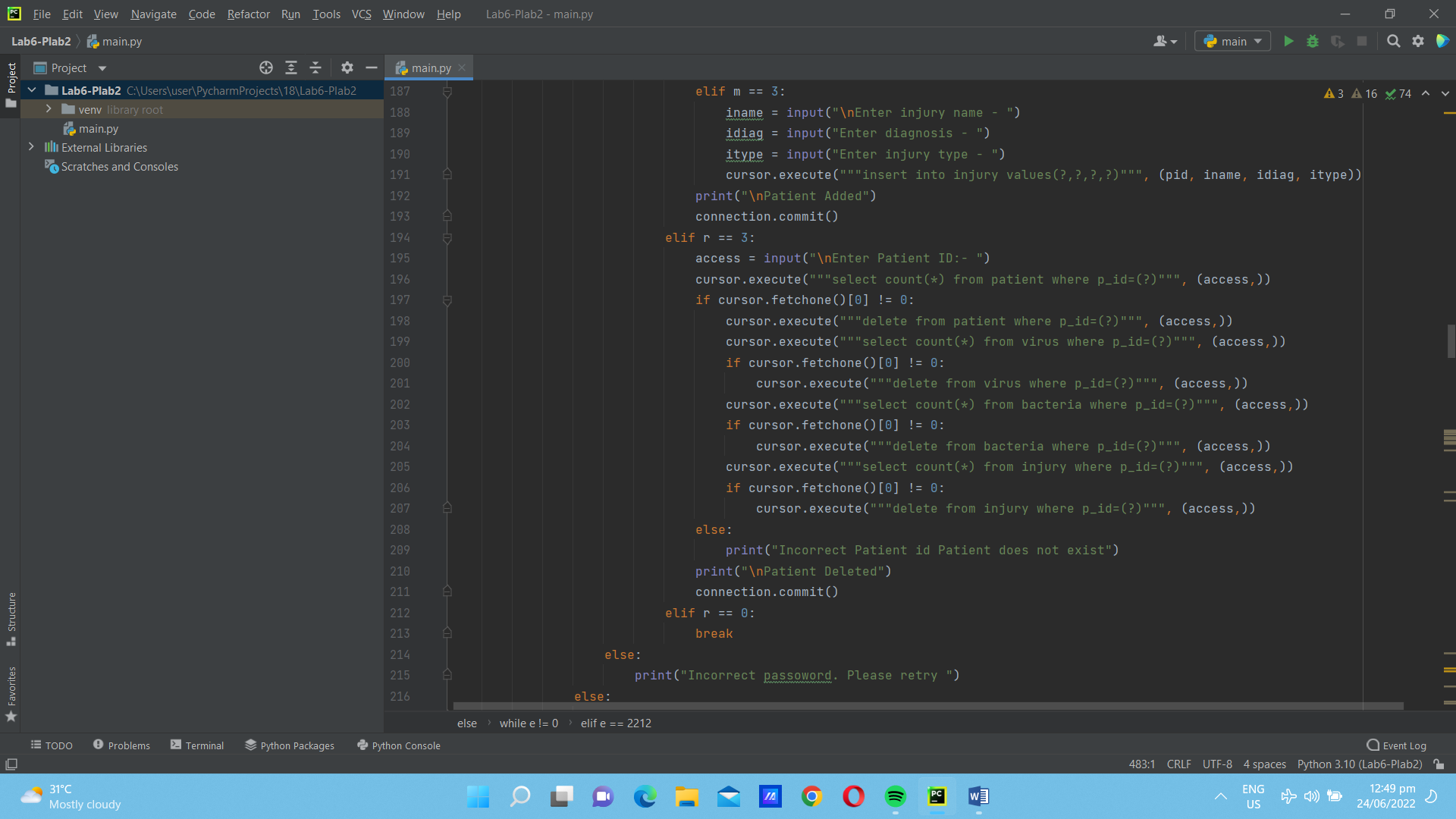


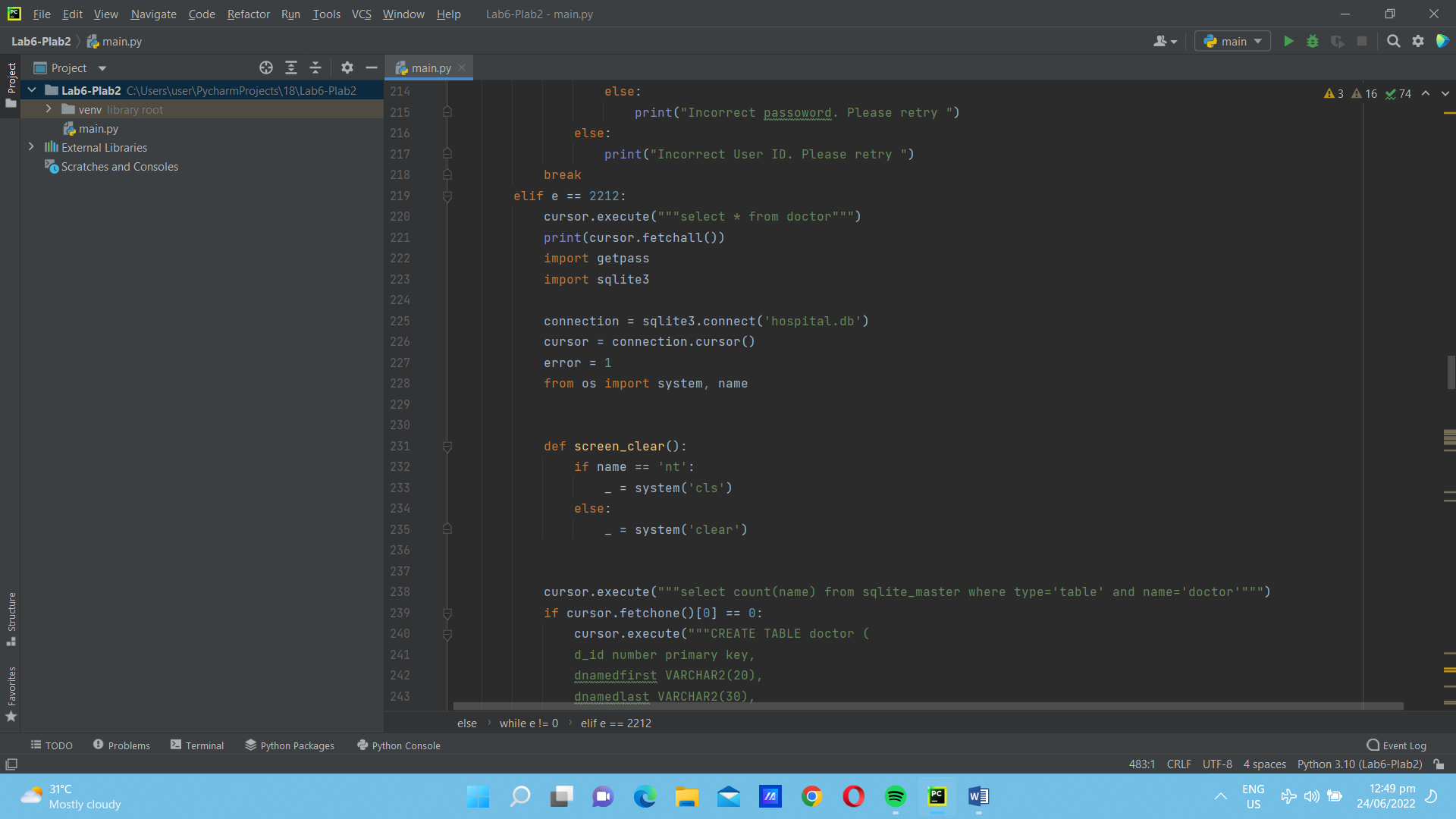


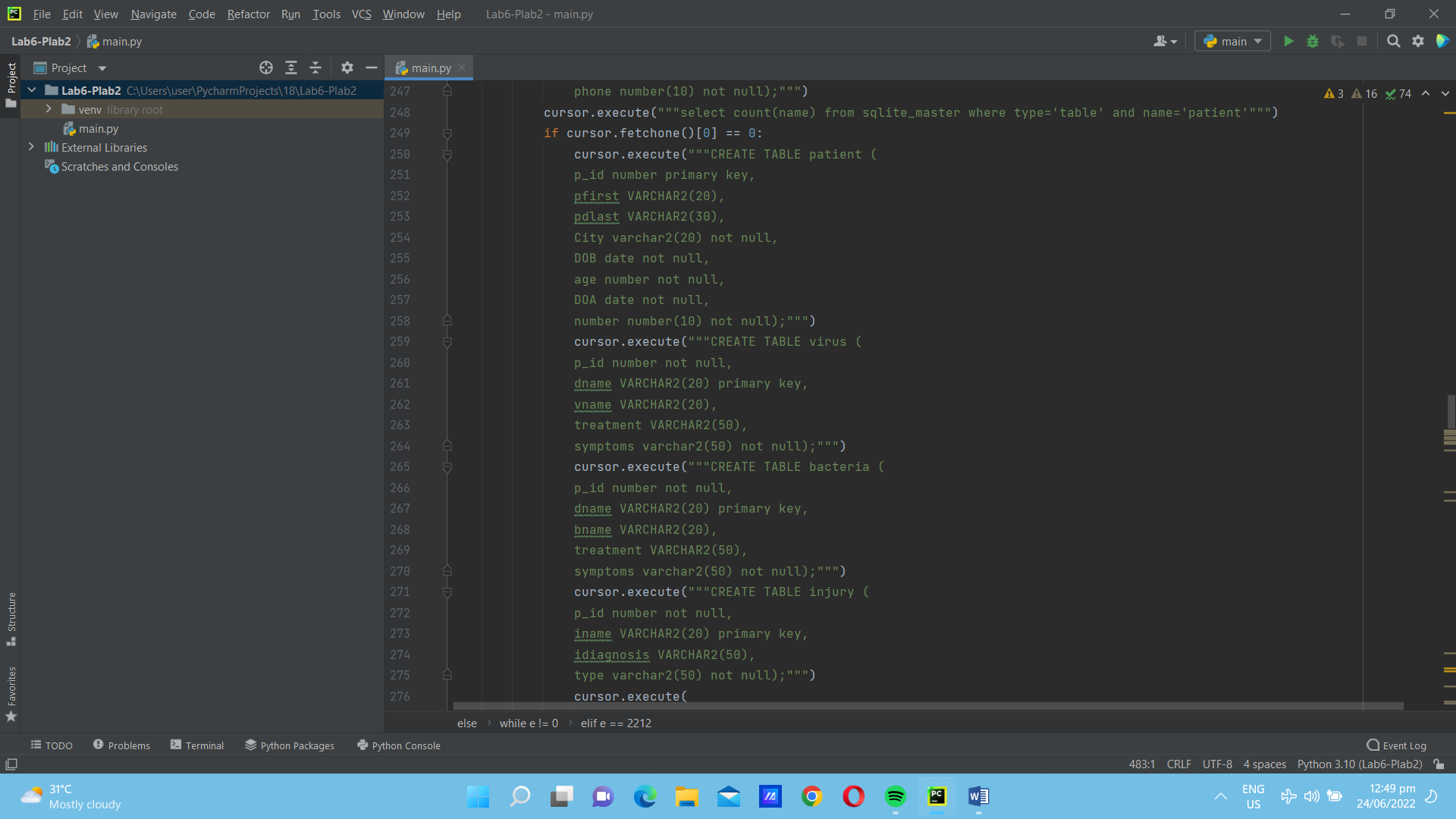


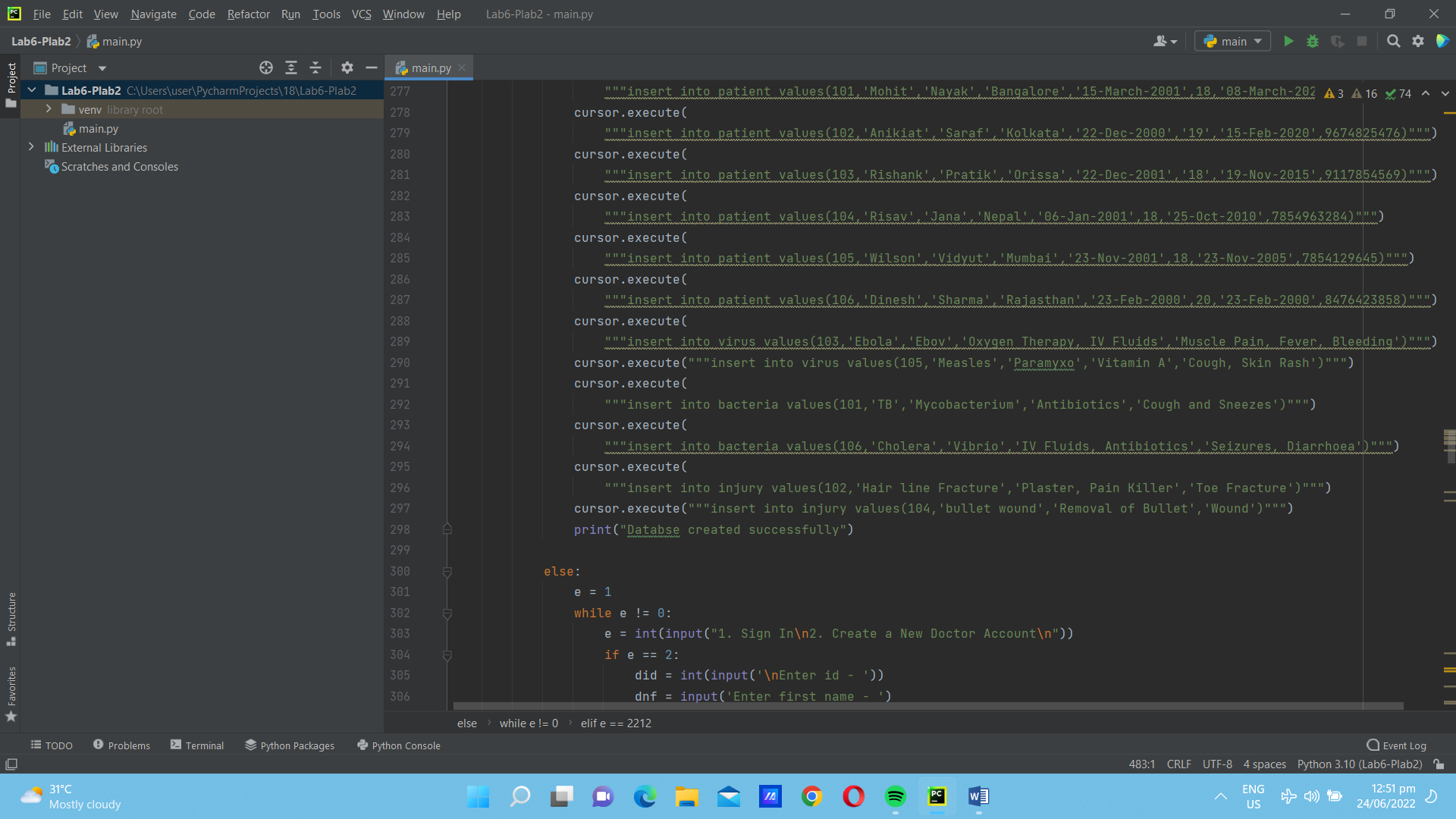


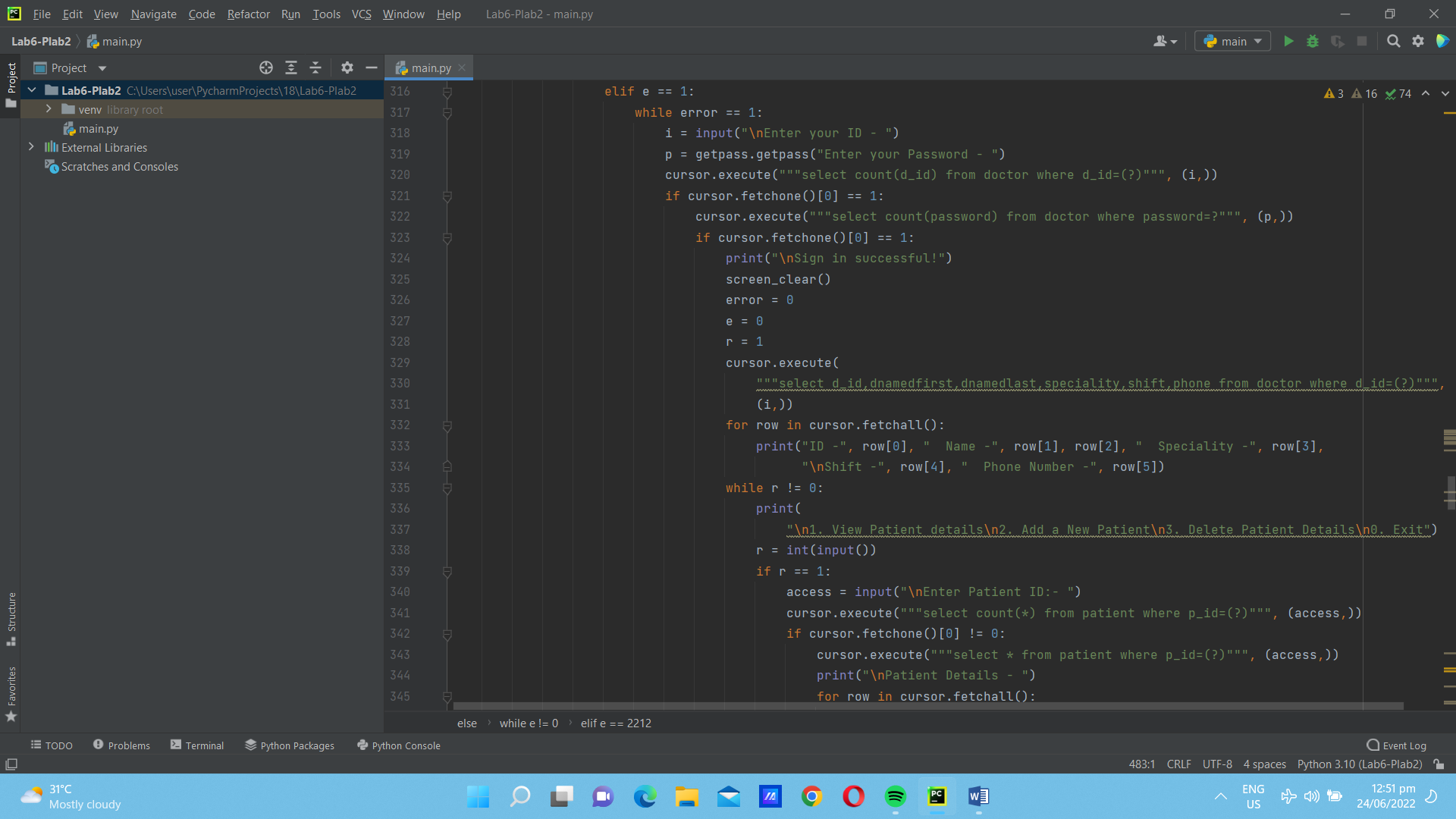


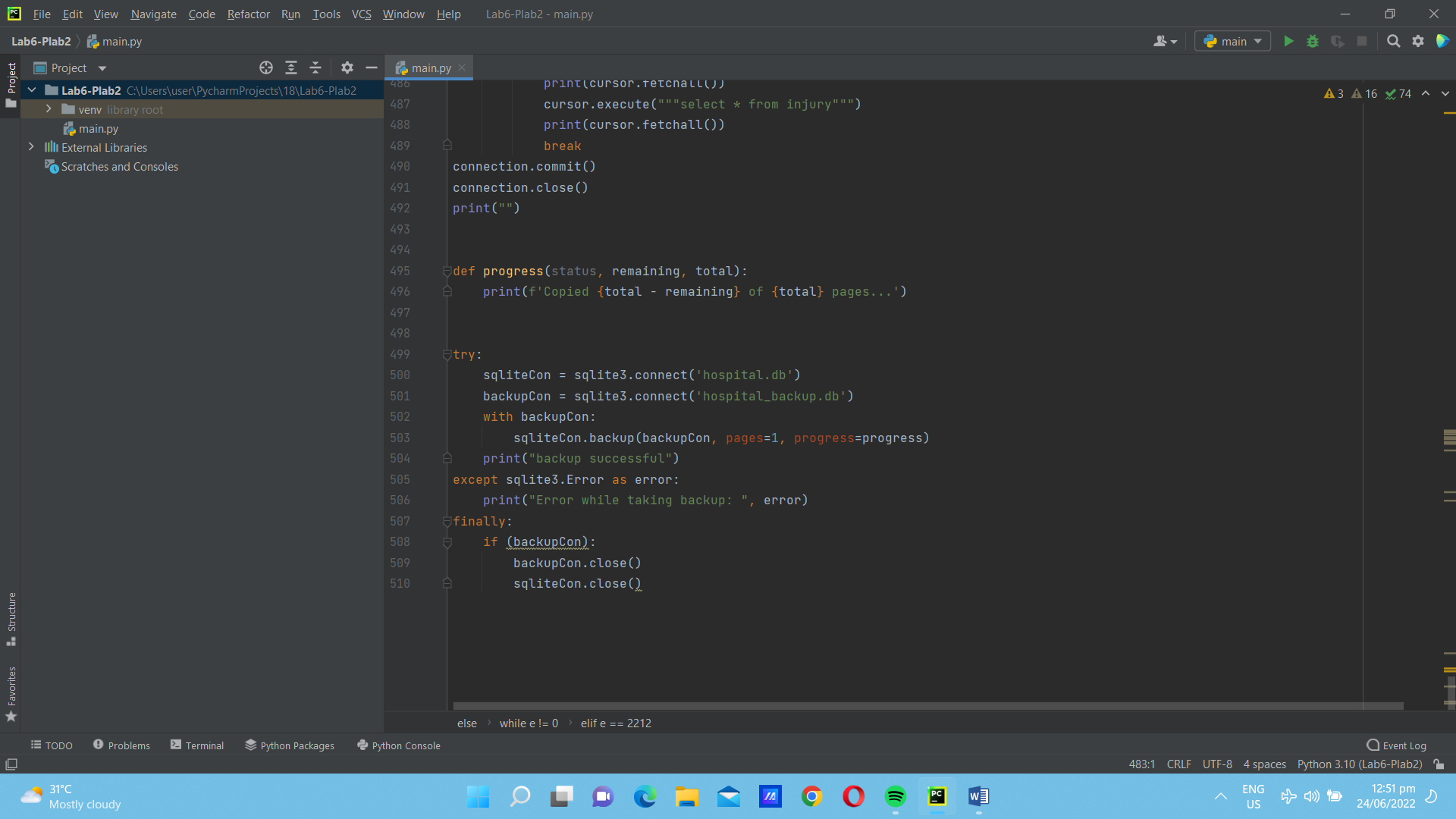




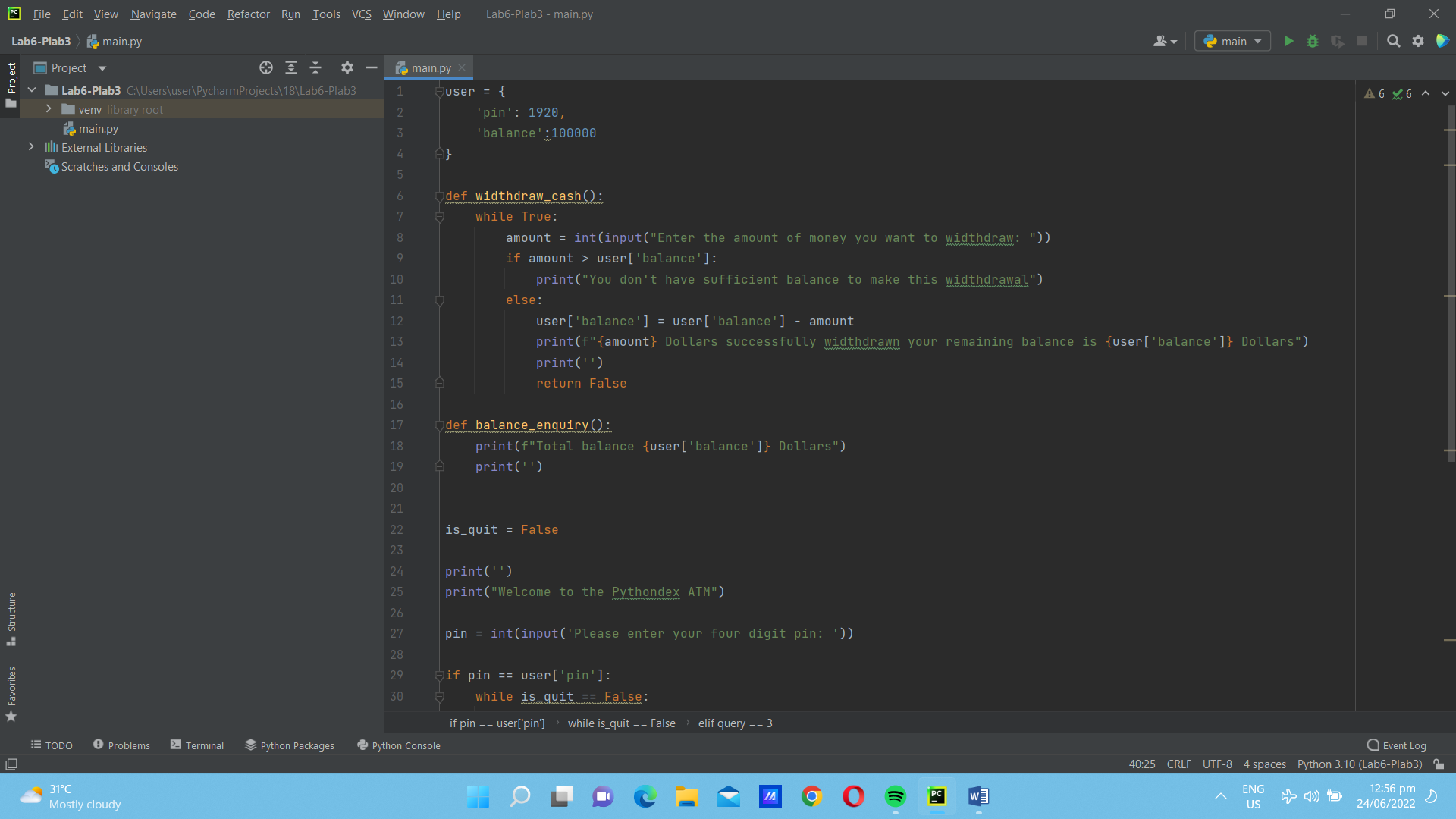


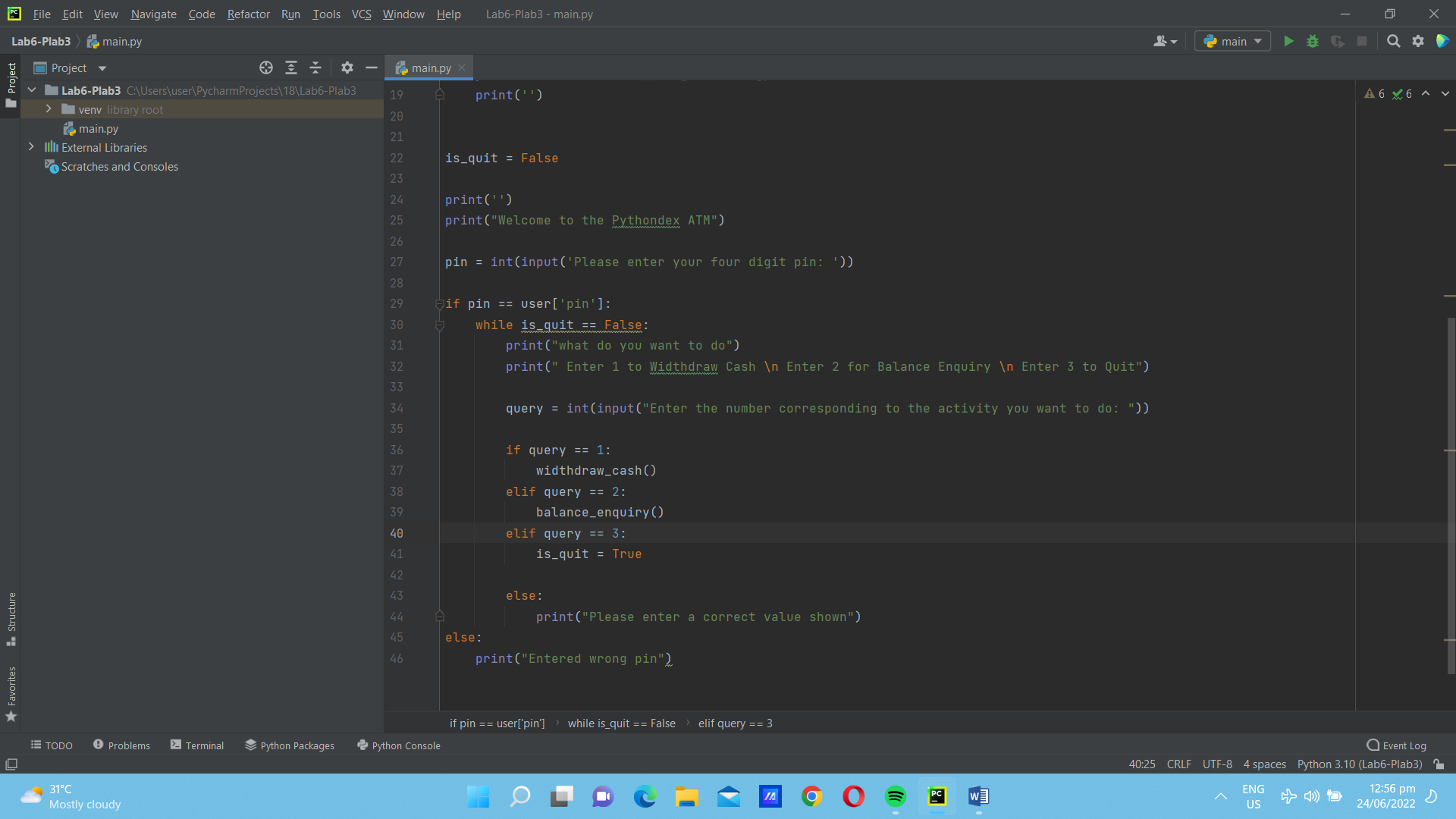






3. A crude multi-client chat room allows two or more users to converse by sending and receiving messages. On the client side, a user connects to the chat room as in the ATM application, by clicking a Connect button. At that point, a transcript of the conversation thus far appears in a text area. At any 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 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 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 a list of strings, while \_\_str\_\_ returns the join of this list, separated by newlines.





B. Debugging and Sample Run (with edited screengrabs and discussion)

IMPORTANT: Include figure numbers and labels. Edit your screengrabs

Note:1. Save all files in one folder (.py and .ui).

Name the folder: <Course and Section>\_LastName,FirstName\_LAB REPORT#X.  
Example: SOFTWARE DESIGN-2A\_PENTECOSTES,JAY-AR\_LAB REPORT#6.

2.Commit all source codes to Github **(individual Github account**)  
Github Repository Name: Software Design Lab Exercises and put a URL of your Github repository