Design and Create a Scoring and Grading UI Interface

Objective:

Design and develop a graphical user interface (UI) to perform scoring and grading functions, and display the results. Your submission should include a demonstration video showcasing the functionality, an executable file for deployment, and the complete source code.

Required Functions:

- Data collection or import data from files.
- Scoring and grading based on specified criteria.

Bonus Features (UI Enhancements):

- Clear and intuitive display of intermediate items.
- Enhanced user-friendly and aesthetically pleasing interface design.

Notes:

- Implement the UI using PyQt, as it offers powerful tools like Qt Designer for rapid UI development.
- Refer to the PyQt UI demo available at GitHub - UIDemo4SummerSchool for inspiration and guidance.

Appendix 1: PyQt Setup Quick Tutorial

Environment Setup Checklist

- PyCharm
- PyQt5
  - Includes Qt Creator or Qt Designer
- PYUIC

Environment Setup Instructions

1. Download and install PyCharm following instructions available on Google.

2. Install Qt Designer and PYUIC plugin in PyCharm:
    - For Windows: How to Install QT Designer + PYUIC in PyCharm
    - For Ubuntu: Setting up PyQt5 + QtDesigner in PyCharm on Ubuntu 18

Development Pipline

1. **Development of Backend and Frontend Code:**
    - Utilize PyCharm for writing both backend and frontend functional code.
2. **Visual UI Design:**
    - Use Qt Designer for visually designing the user interface.
3. **Conversion of .ui to .py:**
    - Utilize `pyuic` to convert .ui files created in Qt Designer into Python script files.

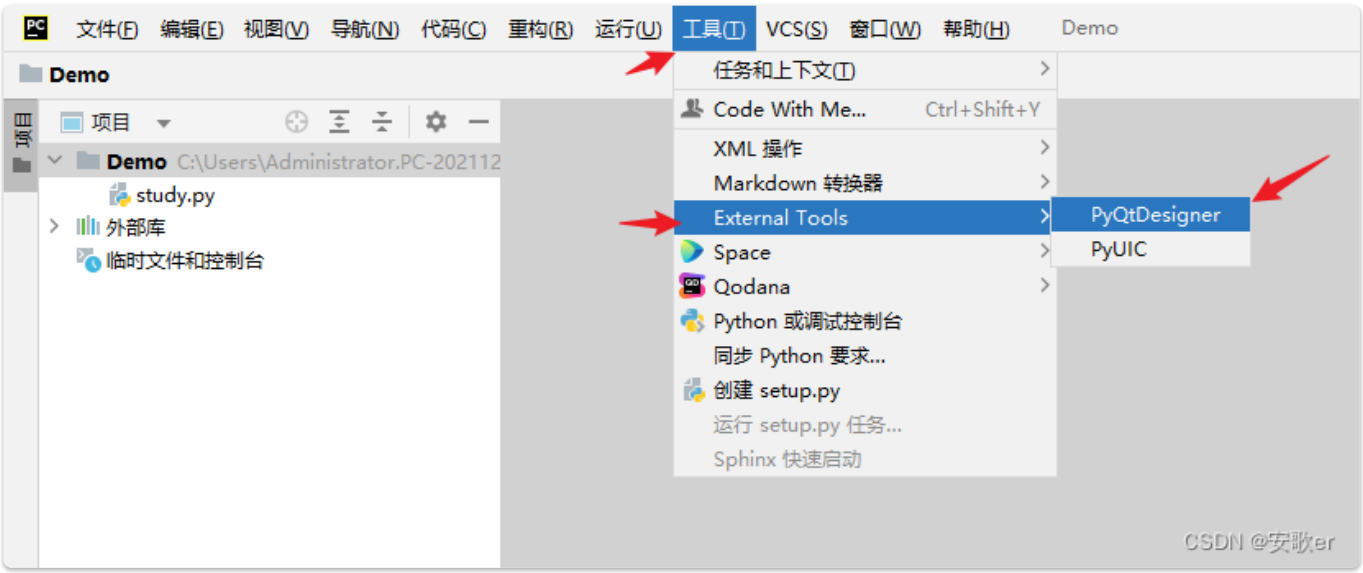
Appendix 2: Instructions for the first demo in PyQt

Step 1: Visualization Interface Design

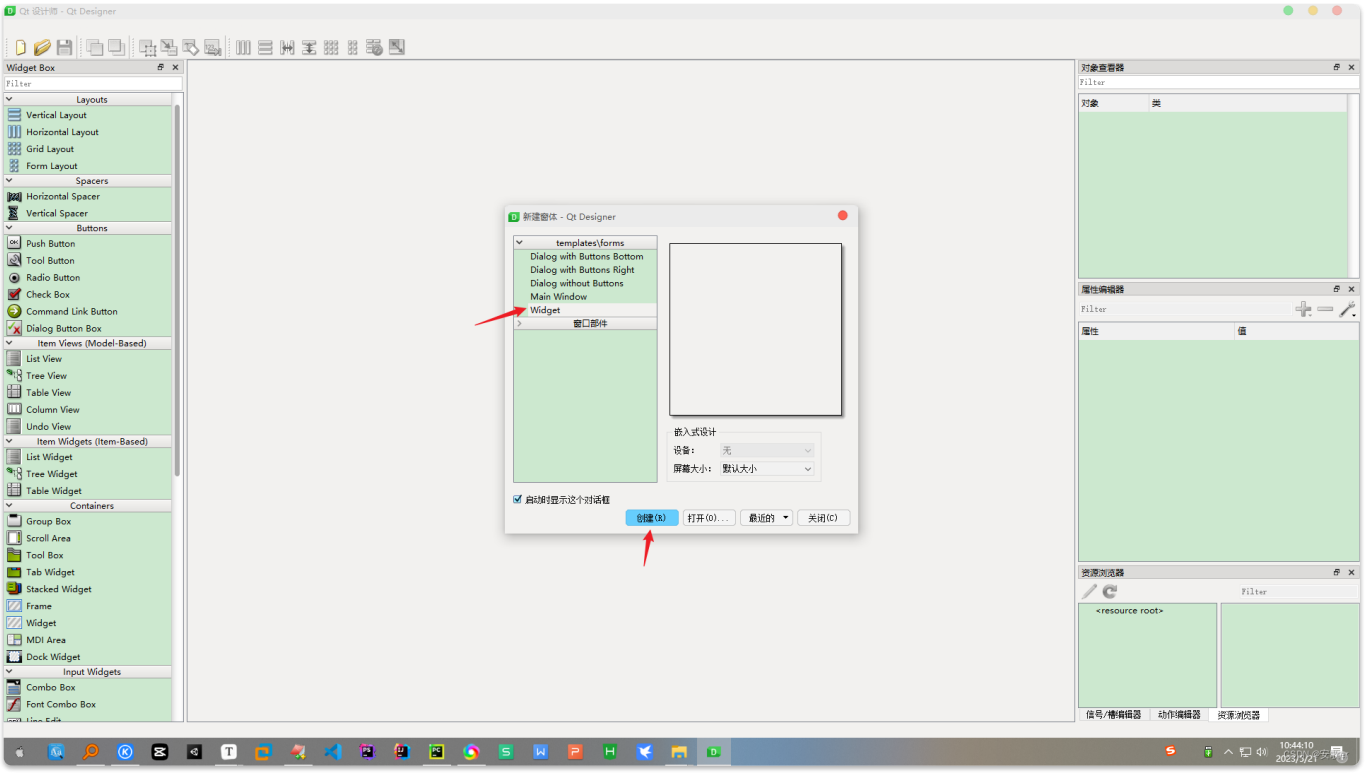
I. Visualization Interface Initialization

First, open QT Designer. Based on the opening method in Pycharm, the specific steps are as follows: [Tools] -- [External Tools] -- [PyQtDesigner].

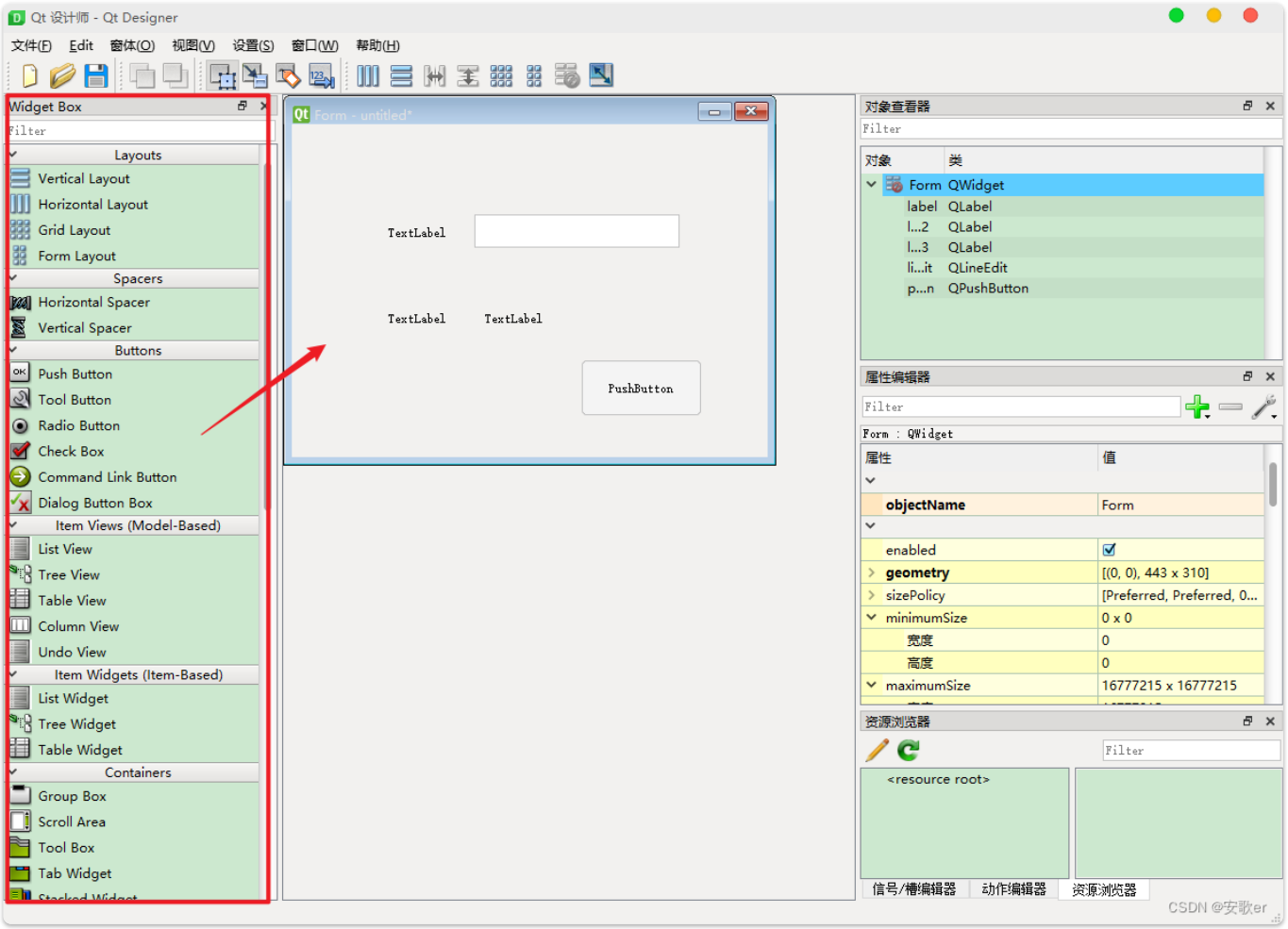
Please note, if there is no [External Tools] option, please set up the environment according to the previous text and configure the relevant path.



In the opened interface, select [Widget] in the display window and then click [Create] to generate the editing interface for the new form.



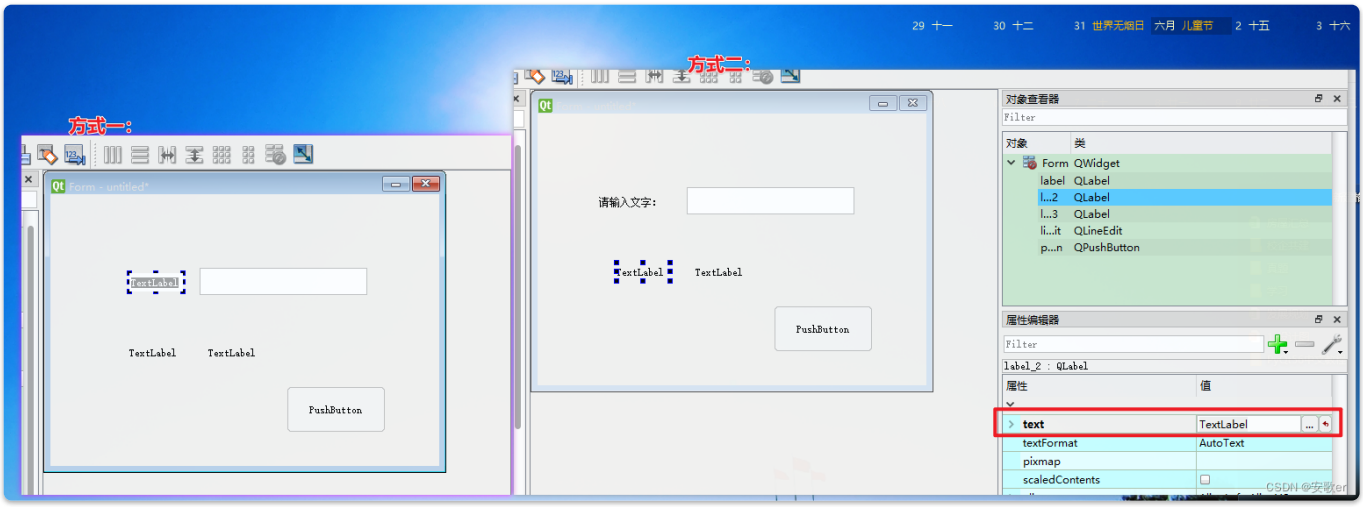
Then, according to your UI interface design, select the appropriate widgets, hold down the left mouse button, and drag them into the new form, arranging them in the desired positions. The following image shows a simple example of widget placement.



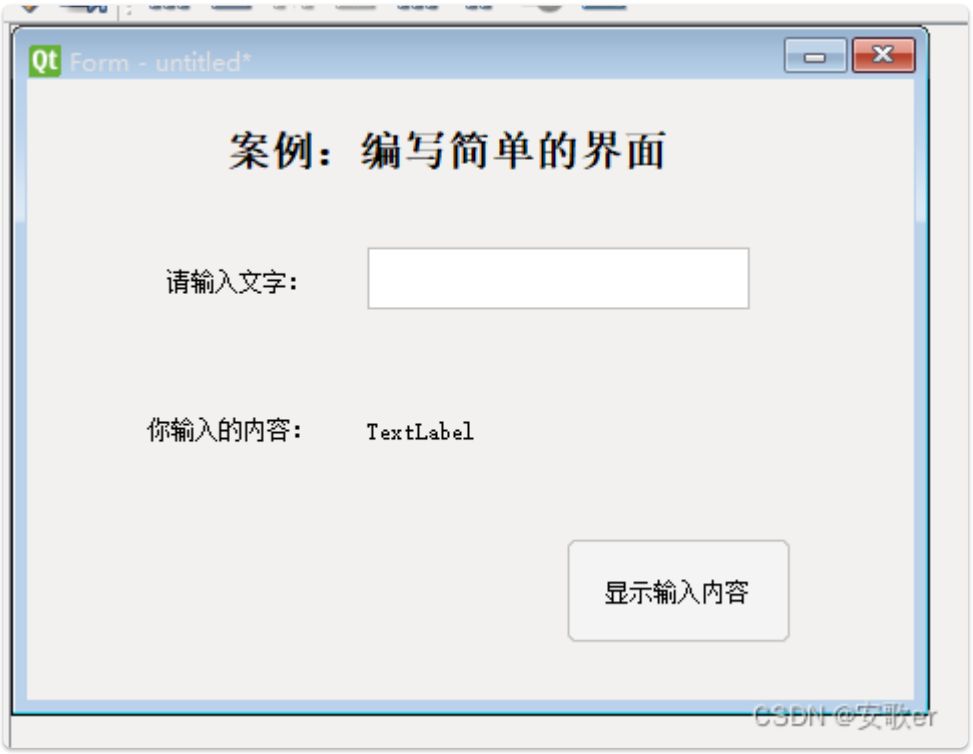
II. Editing Widget Properties

Next, do two things for each widget:

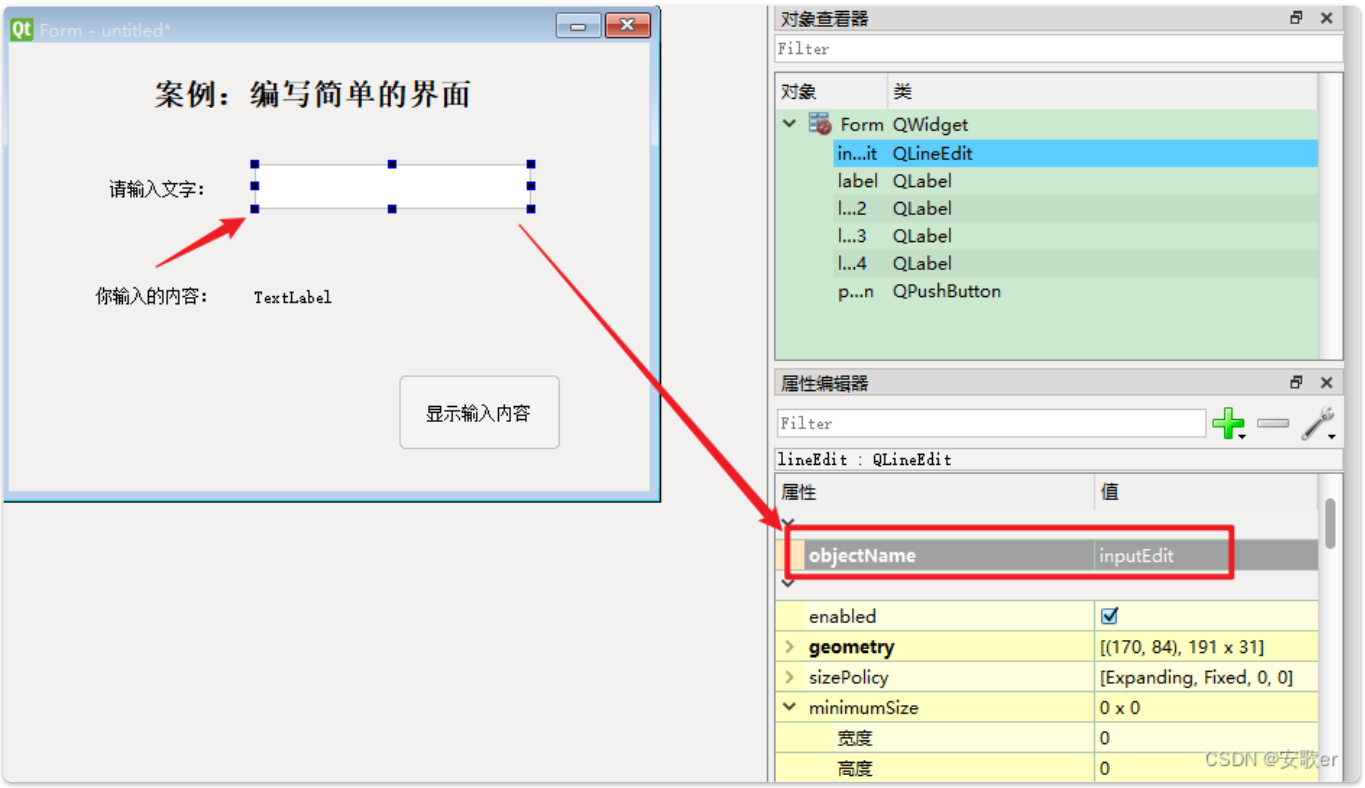
1. Set the display content on the widget:
    - Method 1: Double-click the widget to enter its editing state, allowing you to directly edit the content of the widget.
    - Method 2: Select the widget you want to edit, then find the [text] property in the [Property Editor] at the bottom right corner and change it. As shown in the figure:



The final result of editing the widget content on the modified interface (displayed by pressing Ctrl+R) is shown in the figure:

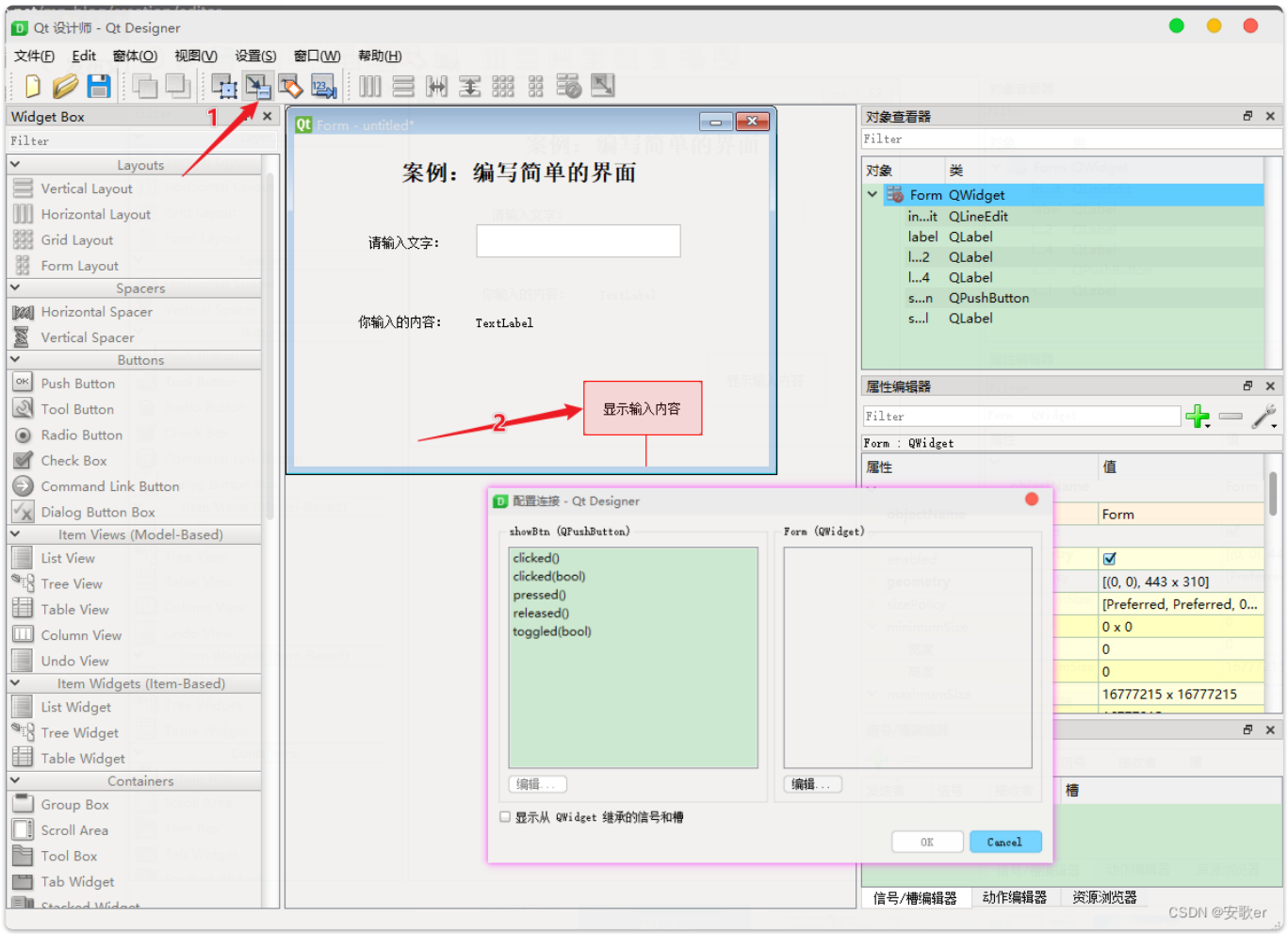


2. Change the name of the widget. The new widget name will be used for calling in the subsequent code. It is generally required to name it according to its function. For example, a jump button can be named `gotoBtn` so that its purpose is clear from its name. The specific way to change the name is: select the widget you want to rename, then find the `objectName` property in the [Property Editor] and change it. (For other widgets, please refer to this step.)



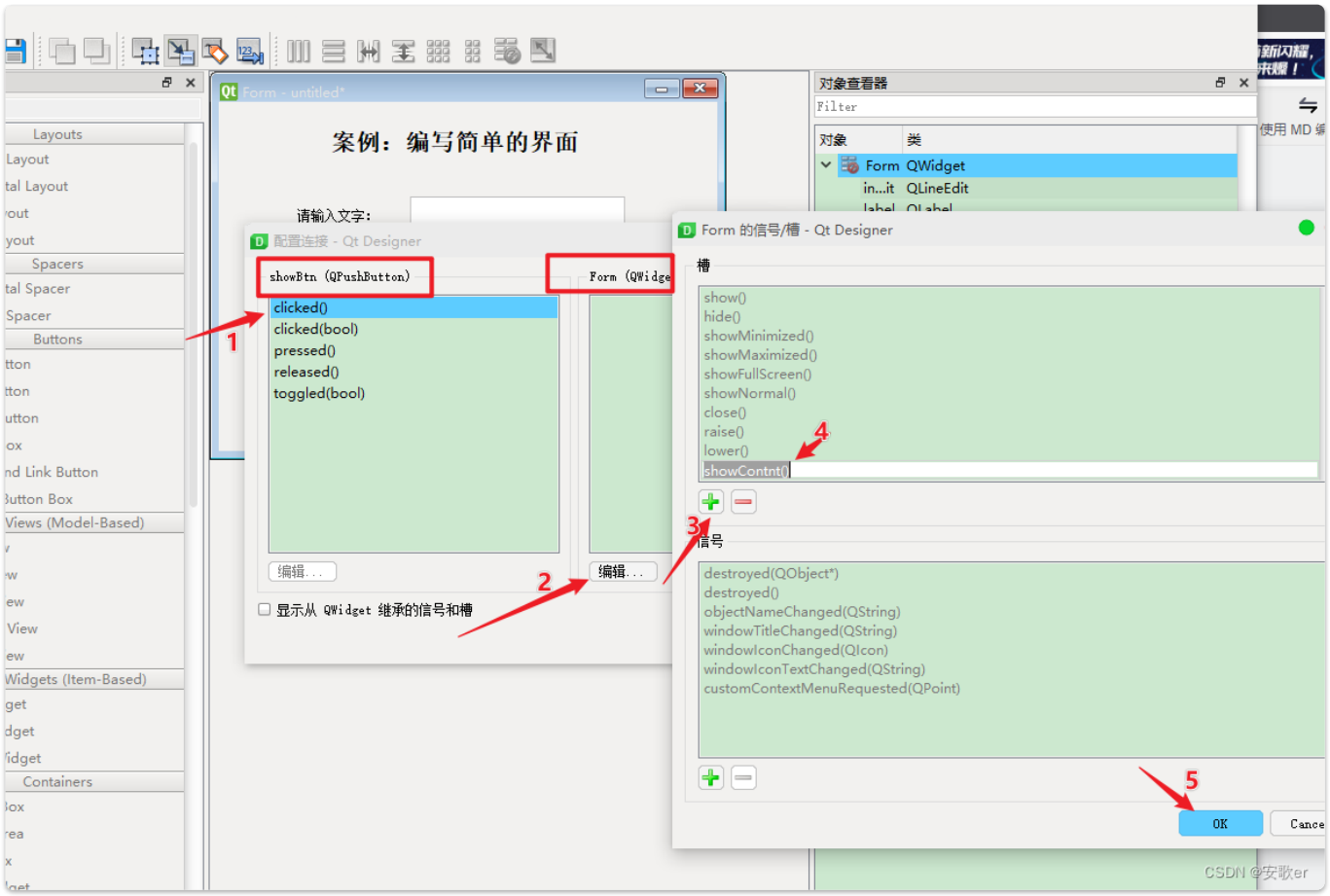
III. Event Response: Signal and Slot Functions

After completing the above steps, we will bind an event response to the "Display Input Content" button. The specific operation is as follows: select the [Signal/Slot] icon in the menu bar (as shown in the figure), then click the "Display" button, hold it down, and drag it. When a red line appears, move the mouse outside the form. At this point, the "Configure Connection" dialog box will pop up.

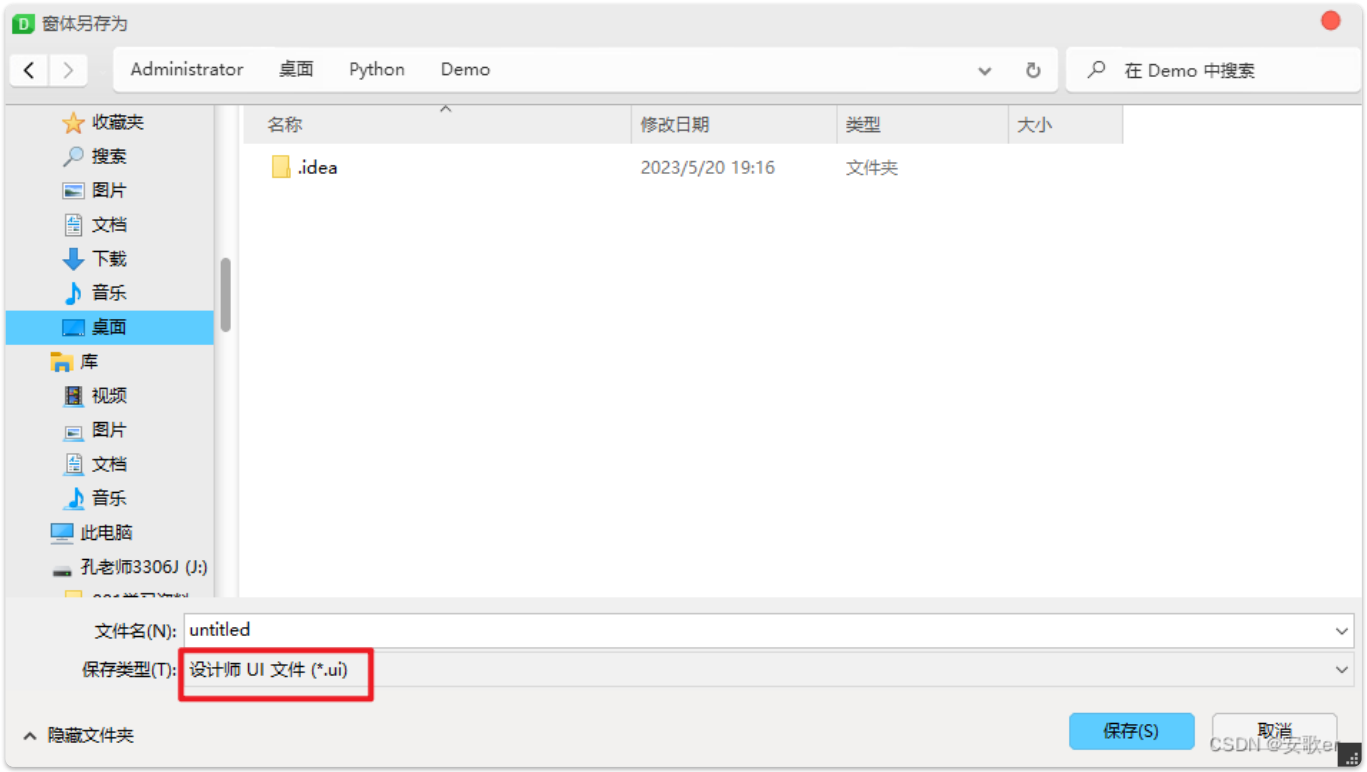


In the "Configure Connection" dialog box, the left sidebar corresponds to the event for `showBtn`. Select the `clicked` event. Then, the right sidebar will become editable. Click the "Edit" button below, and the "Signal/Slot" dialog box will appear. In this dialog box, click the "+" button and name a slot function, for example, `showContent()`. After completing this, click the "OK" button.

Return to the "Configure Connection" dialog box. On the right sidebar, you will see the slot function you named. Select it to bind it to the `clicked` event of the button. This way, in your program, clicking the button will execute the logic defined in your `showContent()` function.

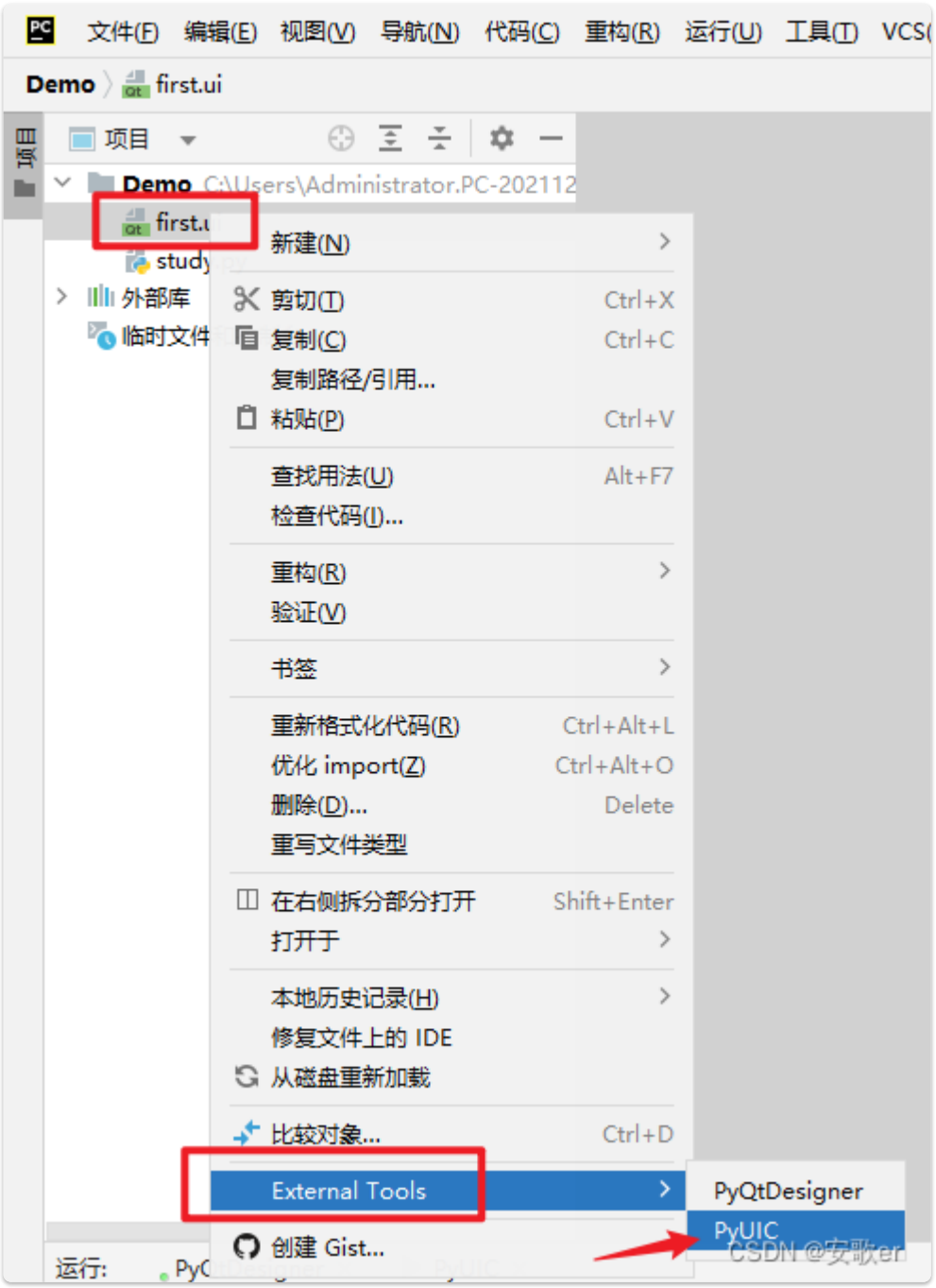


At this point, the design of the visual interface is complete. Click [File] -> [Save], which will prompt you to save the file as a `*.ui` type. Name it `first.ui`, choose a suitable location, and then save it.

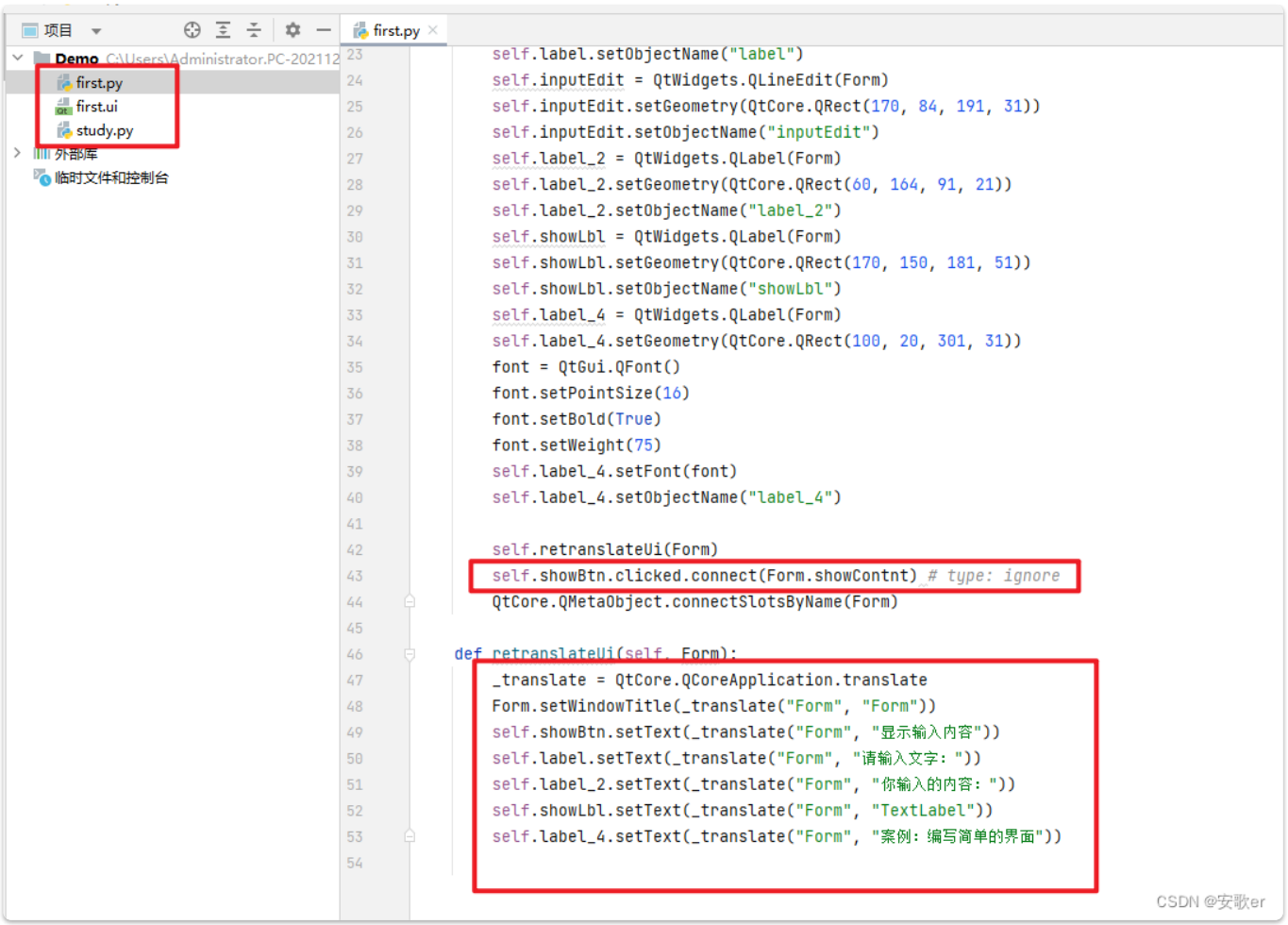


Step 2: Converting UI File to Python File

In the PyCharm development environment, navigate to "Project", locate the recently saved `first.ui` file, right-click on it, then select [External Tools] -> [PyUIC]. This action will generate a `first.py` file.



Opening the `first.ui` file allows you to view its contents.



Step 3: Writing Code to Implement Specific Logic

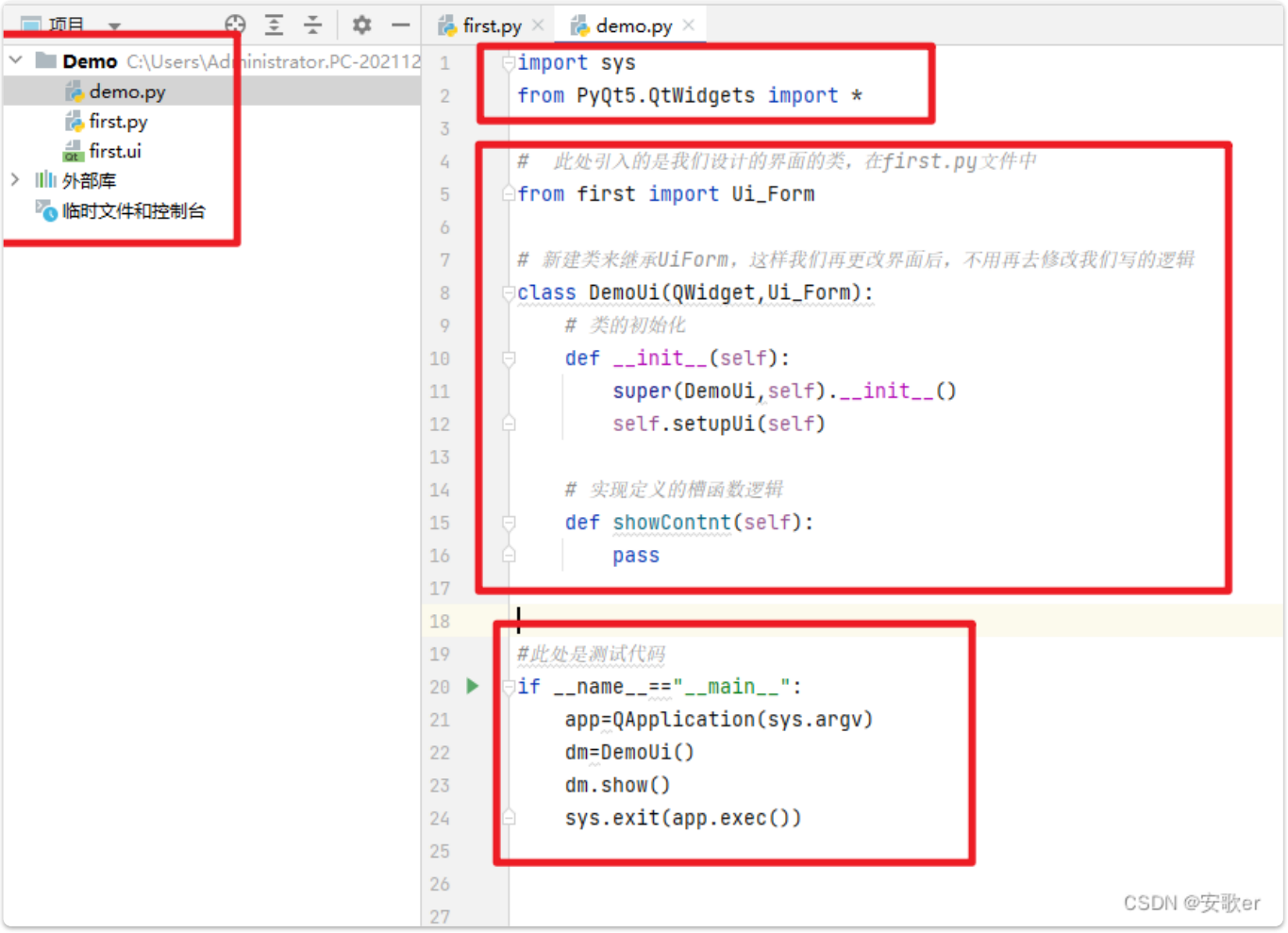
In your project, create a new Python file named `demo.py`. Open this file to begin writing your code.

I. Implementing Project Framework Imports

In the `demo.py` file, write the following code snippet:

```
import sys
from PyQt5.QtWidgetsimport *

#  此处引入的是我们设计的界面的类，在first.py文件中
from first import Ui_Form

# 新建类来继承UiForm，这样我们再更改界面后，不用再去修改我们写的逻辑
classDemoUi(QWidget,Ui_Form):
# 类的初始化def__init__(self):
  super(DemoUi,self).__init__()
  self.setupUi(self)

# 实现定义的槽函数逻辑
```

```
defshowContnt(self):
        pass

#此处是测试代码
if __name__=="__main__":
        app=QApplication(sys.argv)
        dm=DemoUi()
        dm.show()
        sys.exit(app.exec())
```



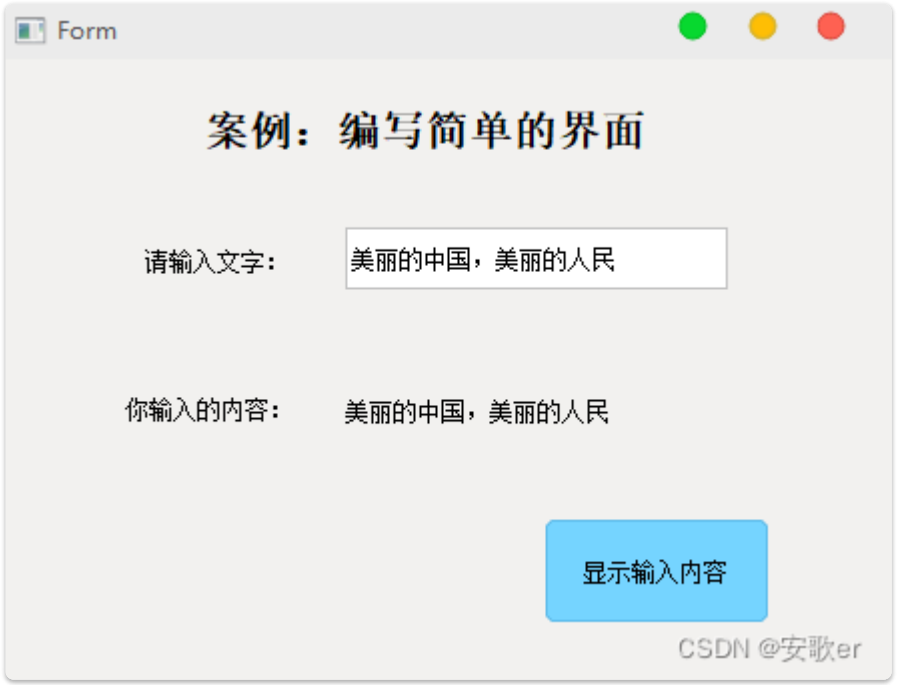
After clicking the run button, the interface window will appear as shown in the image.

II. Implementing the Slot Function Logic

Within the slot function, implement the response to events. The specific code is as follows:

```
# 实现定义的槽函数逻辑
def showContnt(self):
        str =self.inputEdit.text()
        self.showLbl.setText(str)
```

Once completed, clicking run will enable the button event response as depicted in the image.



That completes the development of a complete visual program.