

# Python GUI Programming – II

PyQt5 Events, Signals, and Slots

【110上】嵌入式系統技術實驗課程

TA: 陳翰群 [hanz1211.ee09@nycu.edu.tw](mailto:hanz1211.ee09@nycu.edu.tw)

# Events, Signals, and Slots

- GUIs are **event-driven**, meaning that they respond to events that are created by the user, from the **keyboard** or the **mouse**, or by events caused by the system, such as a **timer** or when connecting to Bluetooth
- When **exec\_()** is called, the application begins **listening for events** until it is closed.

# Events, Signals, and Slots

- In PyQt, event handling is performed with signals and slots.
- **Signals** are the events that **occur when a widget's state changes**.
  - Such as when a button is clicked or a checkbox is toggled
- **Slots** are the **methods** that are executed in response to the signal.
  - Slots can be simply Python functions or built-in PyQt functions

# Events, Signals, and Slots

- Take a look at the following code from the earlier QPushButton program:

```
button.clicked.connect(self.submit)
```

clicked() signal emitted





self.submit() is the slot get called

- Note that the slot method is a **callback function**, means that **will be called later** in the object lifetime
  - That's why we **don't use**

```
button.clicked.connect(self.submit())
```

# QMessageBox

- QMessageBox dialog box can be used to alert the user to a situation or to allow them to decide how to handle the issue
- There are four types of predefined QMessageBox widgets in PyQt

| QMessageBox Icons   | Types       | Details                                       |
|---|-------------|---|
|    | Question    | Ask the user a question.                      |
|    | Information | Display information during normal operations. |
|    | Warning     | Report noncritical errors.                    |
|  | Critical    | Report critical errors.                       |

# QMessageBox

- dialogs.py
- Create a QMessageBox object

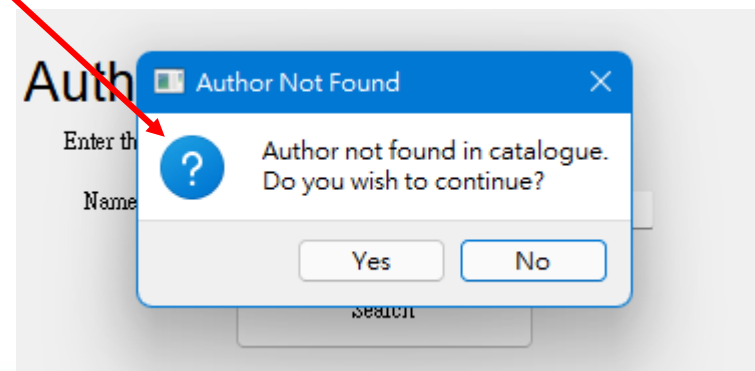
```
not_found_msg = QMessageBox()
```

- Edit message box detail, this line will show the widget

```
not_found_msg = QMessageBox.question(self, "Author Not Found",  
                                     "Author not found in catalogue.\nDo  
                                     you wish to continue?",  
                                     QMessageBox.Yes | QMessageBox.No,  
                                     QMessageBox.No)
```

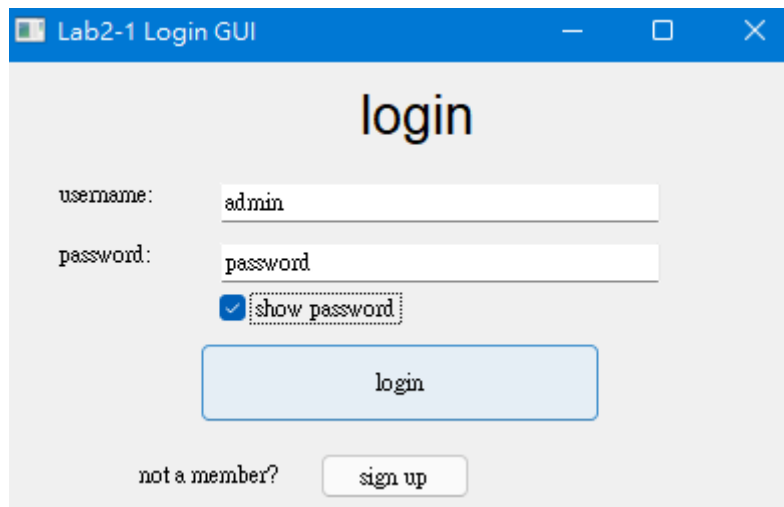
- Arguments

- Title
- Content
- Button separated by pipe key |
- Default highlight

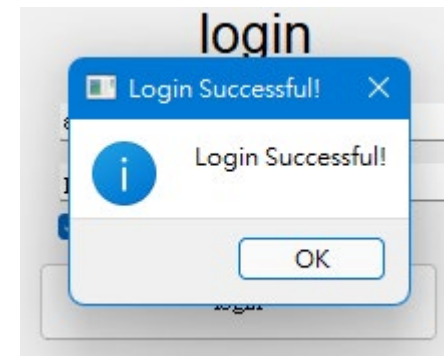


# Lab2-1 Login UI

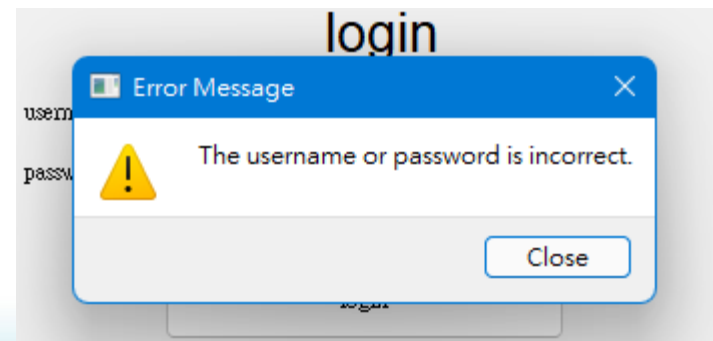
- lab2\_loginUI.py & Registration.py
- Recreate this GUI
  - Only the user stored in files/users.txt are allow to login
  - You should be able to toggle checkbox to hide/show password field
  - You can use sign up to add more user or just edit files/users.txt



user in  
files/users.txt



user not in  
files/users.txt





# Lab2-1 Login UI

- Hint:
  - After user click the button in QMessageBox, it will return the result, you can use if else to check the answer

```
answer = QMessageBox.question(...)
if answer == QMessageBox.Yes:
    ...
```

- The stateChanged signal will pass the state to slot as an argument

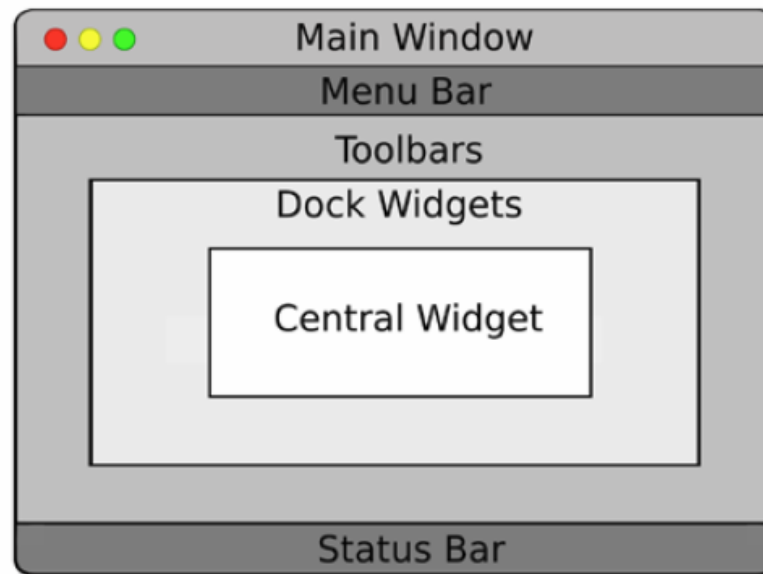
```
someCheckbox.stateChanged.connect(self.someSlot)
```

```
def someSlot(self, state):
    if state == Qt.Checked:
        ...
```



# QMainWindow vs. QWidget

- The **QMainWindow** class focuses on creating and managing the layout for the main window of an application.
- It allows you to set up a window with a **status bar**, a **toolbar**, **dock widgets**, or other menu options **in predefined places** all designed around functions that the main window should have.



# QMenuBar

- menu\_framework.py
- Note that we are using QMainWindow as base class, not QWidget

```
class BasicMenu(QMainWindow):
```

- In order to create a menubar, you must create an instance of the QMenuBar class, which we created by:

```
# Create menubar  
menu_bar = self.menuBar()  
menu_bar.setNativeMenuBar(False)
```

- Due to guidelines set by the MacOS system, you must set the property to use the platform's native settings to False. For those using Windows or Linux, you can comment this line out or delete it completely from your code.
- Adding menus to the menubar is also really simple in PyQt:

```
file_menu = menu_bar.addMenu('File')
```

# QAction

- menu\_framework.py
- A menu contains a list of action items such as Open, Close, and Find.
- In PyQt, actions are created from the **QAction** class, defining actions for menus and toolbars.

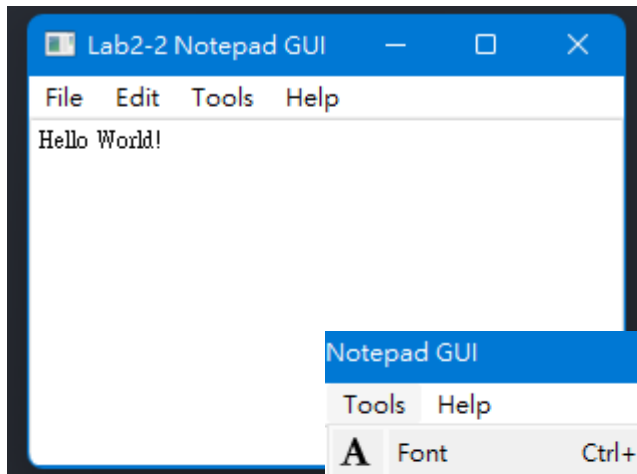
```
# Create actions for file menu  
exit_act = QAction('Exit', self)  
exit_act.setShortcut('Ctrl+Q')  
exit_act.triggered.connect(self.close)
```

- The Exit action, exit\_act, is an instance of the QAction class
- shortcut for the exit\_act is set explicitly using the setShortcut()
- Actions in the menu emit a **signal** and need to be connected to a slot in order to perform an action. This is done using **triggered.connect()**
- After defining QAction, use addAction() to add to menu

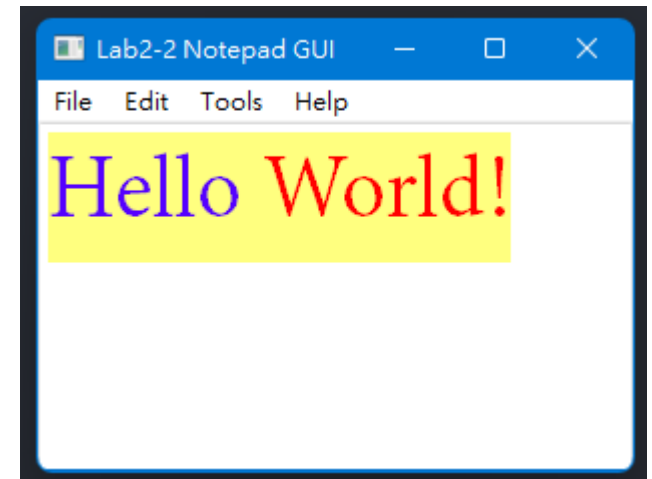
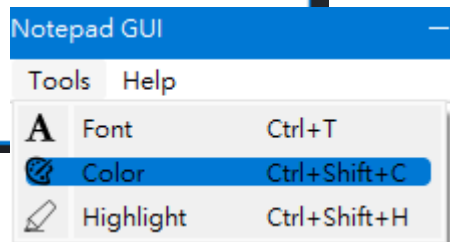
```
file_menu.addAction(exit_act)
```

# Lab2-2 Notepad

- lab2\_notepad.py
- Create a rich text notepad, allow different font and color input.
- We will check if you can open html file, edit & save it, and then open it again.



manual edit  
or open  
files/hello world.html



# Lab2-2 Notepad

- Hint:
- QInputDialog, QFileDialog, QFontDialog will return two variables: the user input/selection and a bool shows if user click yes

```
# Display input dialog to ask user for text to search for  
find_text, ok = QInputDialog.getText(self, "Search Text", "Find:")
```

- QColorDialog.getColor() only returns color, use color.isValid() to validate input

```
color = QColorDialog.getColor()  
if color.isValid():  
    self.text_field.setTextBackgroundColor(color)
```

# QTimer

- PyQt5 provides classes for dealing with dates, **QDate**, or time, **QTime**.
- **QDateTime** class supplies functions for working with both dates and time.
- clock.py

```
# Create timer object  
timer = QTimer(self)  
timer.timeout.connect(self.updateDateTime)  
timer.start(1000)
```

- The timer is set up in initializeUI(), and its timeout() signal is connected to the updateDateTime() slot. The timeout() signal is emitted every second.
- You can pause the timer by timer.stop()

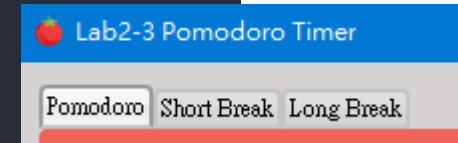
# QTabWidget

- lab2\_pomodoro.py
- A QTabWidget can have multiple QWidget added as child tab

```
self.tab_bar = QTabWidget(self)

self.pomodoro_tab = QWidget()
self.short_break_tab = QWidget()
self.long_break_tab = QWidget()

self.tab_bar.addTab(self.pomodoro_tab, "Pomodoro")
self.tab_bar.addTab(self.short_break_tab, "Short Break")
self.tab_bar.addTab(self.long_break_tab, "Long Break")
```



- The second argument of addTab() is the tab label
- When tab change, QTabWidget will emit **currentChanged()** signal, we need to connect to a slot to handle related functions

```
self.tab_bar.currentChanged.connect(self.tabsSwitched)
```

- The **slot will receive index** of the selected tab

```
def tabsSwitched(self, index):
```



# QLCDNumber

- lab2\_pomodoro.py
- This is just a timer with LCD style, the display() method can change the value
  - display() can simply accept string with proper format like "12:34"
  - You can check calculateDisplayTime() method for detail

```
self.pomodoro_lcd = QLCDNumber()  
self.pomodoro_lcd.setObjectName("PomodoroLCD")  
self.pomodoro_lcd.setSegmentStyle(QLCDNumber.Filled)  
self.pomodoro_lcd.display(start_time)
```

- Combined with QTimer to achieve a nice looking clock.

# setStyleSheet()

- lab2\_pomodoro.py & PomodoroStyleSheet.py
  - You don't need to edit PomodoroStyleSheet.py for this lab
- Qt support stylesheet with **CSS3** standard.
  - To apply stylesheet, use `setStyleSheet()` method, available for `QWidget` and `QApplication`
  - CSS is a standard GUI styling language, mostly **used in all modern websites**.
- You can save a stylesheet as a Python string and import from other files, or use a single line string on demand.
- Single line stylesheet in clock.py:

```
self.setStyleSheet("background-color: black")
```

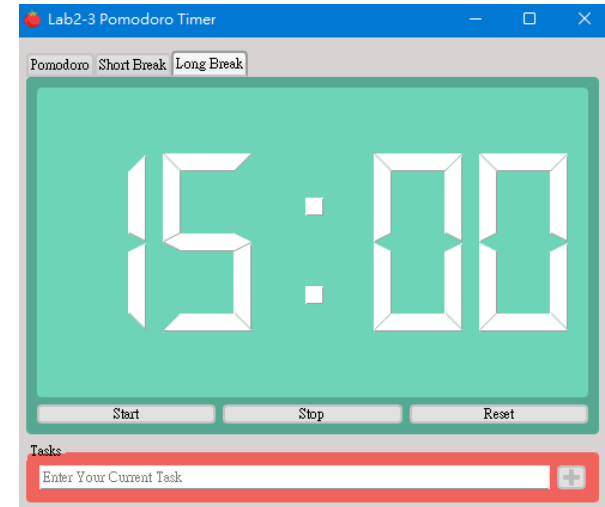
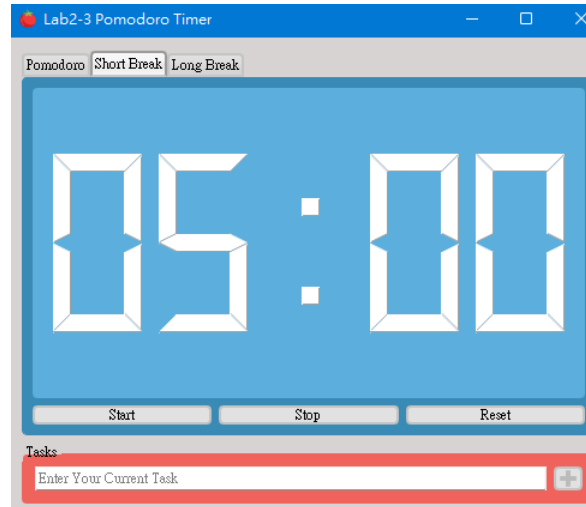
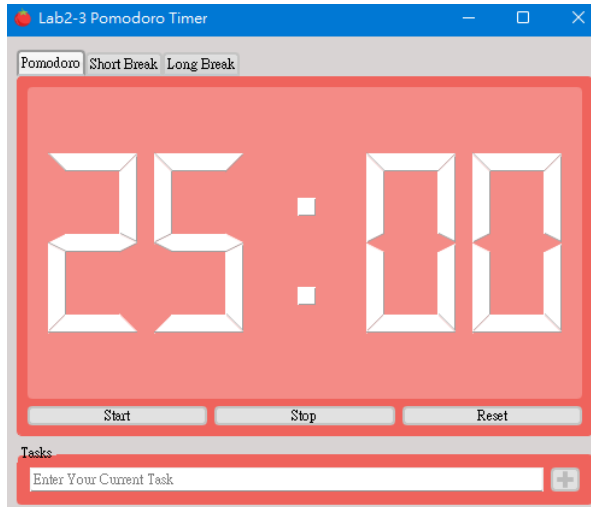
- Stand along stylesheet in lab2\_pomodoro.py:

```
from PomodoroStyleSheet import style_sheet
```

```
if __name__ == '__main__':  
    app = QApplication(sys.argv)  
    app.setStyleSheet(style_sheet)
```

# Lab2-3 Pomodoro Timer

- lab2\_pomodoro.py
- Make sure PomodoroStyleSheet.py is under the same directory.
- Every time you switch tab, the timer shall reset.
- When timer start, the start button need to be disable.



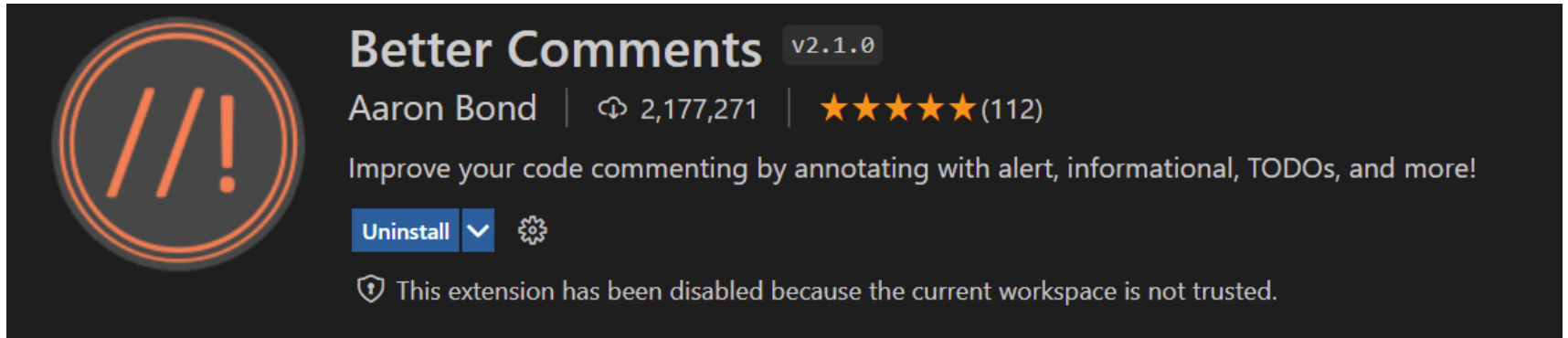
# Lab2-3 Pomodoro Timer

- Hint:
- You can change the global variable for debug, when demo, use 1 second for all time limit.

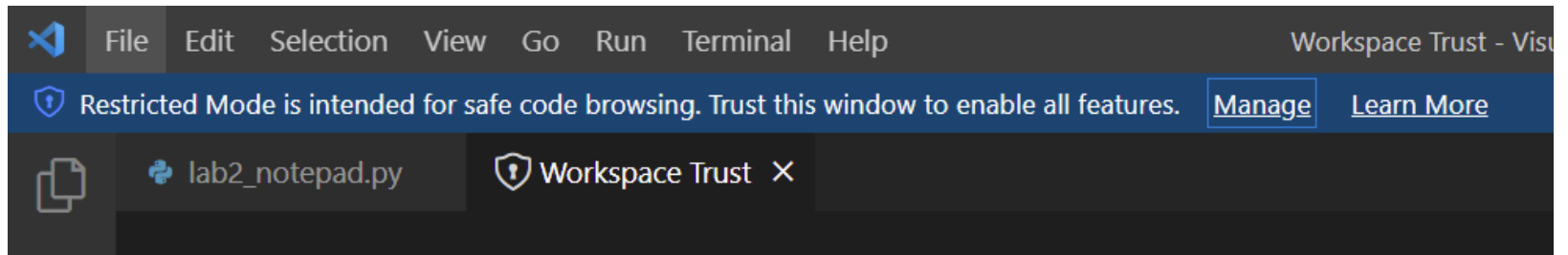
```
# Global variables for each timer  
POMODORO_TIME = 1500000 # 25 minutes in milliseconds  
SHORT_BREAK_TIME = 300000 # 5 minutes in milliseconds  
LONG_BREAK_TIME = 900000 # 15 minutes in milliseconds
```

# TODO Highlight

- You can install this extension in VSCode to highlight all TODOs in your editor.

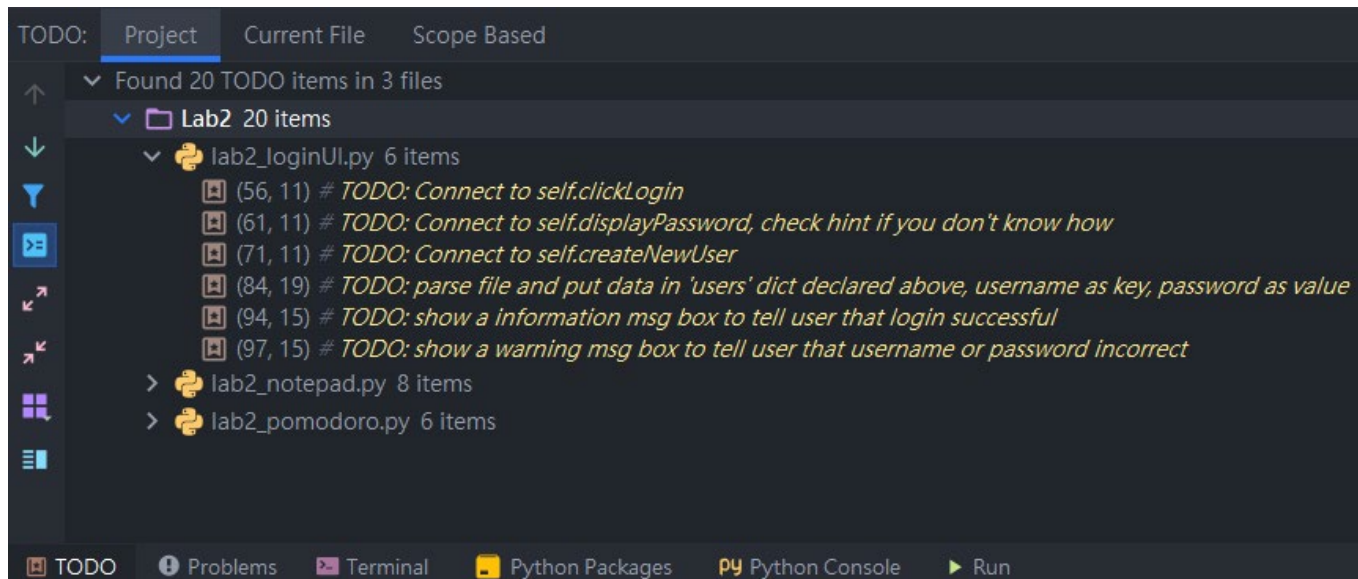


- You need to disable restriction mode to use extension features.



# TODO Highlight

- PyCharm should automatically highlight them for you. You can navigate all TODOs at TODO window.



# Portable venv with requirements.txt

- When inside your venv, you can install multiple packages with a file, note that with different python version, this may not work.

```
pip install -r requirements.txt
```

- Other package management tool like **Anaconda, Miniconda, Pipenv, Poetry...** will check dependency chain, for larger project is recommended.



# Demo

- 本次Lab以個人為單位
- 配分
  - Lab2-1 Login UI: 40%
  - Lab2-2 Notepad: 40%
  - Lab2-3 Pomodoro Timer: 20%
- Demo
  - 完成Lab後，請進視訊會議舉手呼叫助教們demo
  - 多個小題可以分次demo
  - 根據助教要求呈現程式執行結果
- 最後登記時間：21:20