Python GUI Programming – I

PyQt5 Widgets and Layout Management

【110上】嵌入式系統技術實驗課程

TA: 陳翰群 hanz1211.ee09@nycu.edu.tw

Python GUI Library

- PyQt
  - Qt GUI Framework is build with and for C++
  - PyQt API make it available for Python
  - https://doc.qt.io/qt-5.15/
  - https://www.riverbankcomputing.com/static/Docs/PyQt5/
- Kivy
  - https://kivy.org/
- Tkinter
  - https://tkdocs.com/

PyQt installation

- Create a venv is optional
  - The path of venv should only contains ASCII characters
- Getting PyQt for Windows
  - `pip install pyqt5`
- Getting PyQt for MacOS
  - If you are using system default python interpreter instead of venv
    - `pip3 install pyqt5`
  - Or if you are using Homebrew, then you can use the following line instead
    - `brew install pyqt5`
- Varify if PyQt downloaded properly by opening up the Python 3 interpreter and entering the following command:
  - `import PyQt5.QtWidgets`

PyQt5 Modules

Module Name	Description
QtWidgets	Provides the widgets and other classes for creating desktop-style UIs.
QtCore	Contains a variety of extra classes, including the essential non-GUI classes, such as ones for Qt’s signal and slot system.
QtGui	Contains classes for 2D graphics and imaging, event handling, and window system integration.
QtPrintSupport	Provides cross-platform support for configuring and connecting to printers.
QtNetwork	Provides classes for writing communications protocols using UDP or TCP.
QtMultimedia	Contains the classes for multimedia content, cameras, and radios.
QtMultimediaWidgets	Provides additional classes that increase the functionality of multimedia-related widgets.
QtWebEngineCore	Contains the core classes used by other WebEngine modules.
QtWebEngineWidgets	Classes that can be used to create a Chromium-based web browser.
QtSql	Provides classes for working with SQL databases.

PyQt Classes

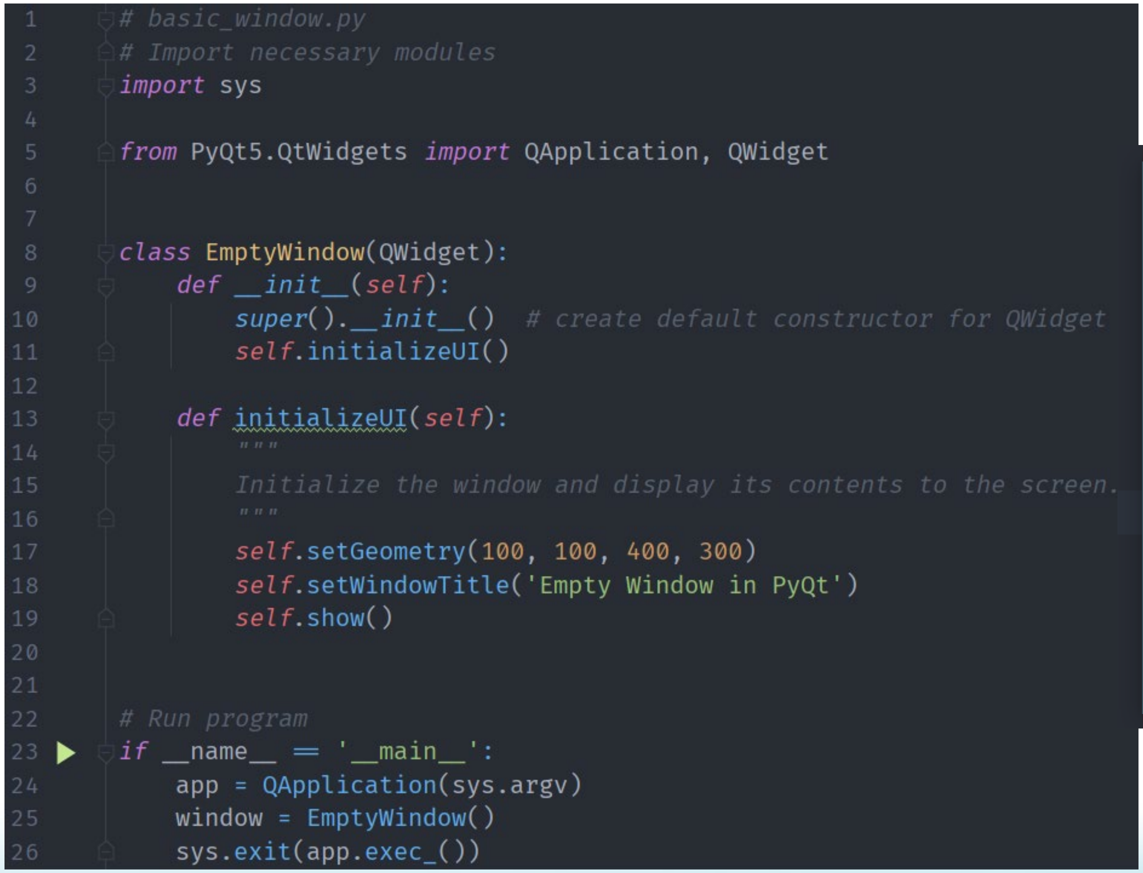
- For a list of all the PyQt classes, check out the following link:
  - https://riverbankcomputing.com/static/Docs/PyQt5/sip-classes.html
- Although it is written for C++, the Qt classes' documentation is generally more detailed. If you want more information about Qt classes, you can also check out
  - https://doc.qt.io/qt-5/classes.html

Classes for Building a GUI Window

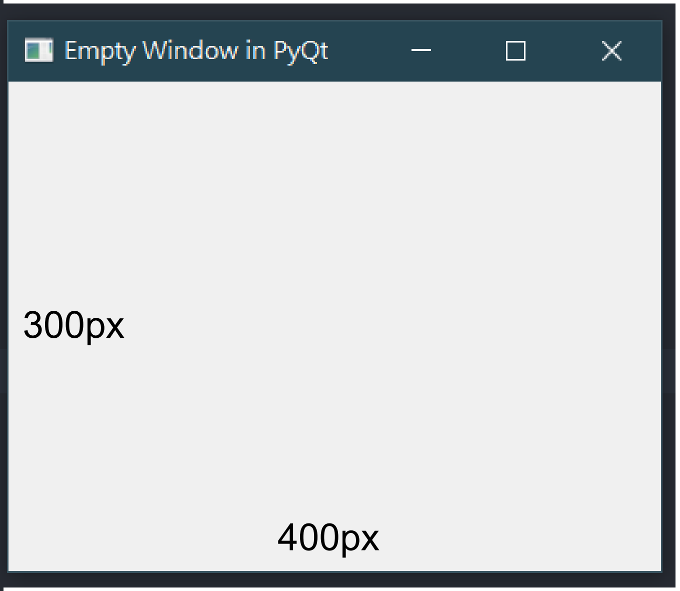
- QApplication
  - QApplication is responsible for <u>handling the initialization and finalization of widgets</u> in GUI
- QWidget
  - The QWidget class is the <u>base class for all of PyQt GUI objects</u>

Classes for Building a GUI Window

- Create an Empty Window (basic_window.py)



```python
# basic_window.py
# Import necessary modules
import sys

from PyQt5.QtWidgets import QApplication, QWidget


class EmptyWindow(QWidget):
    def __init__(self):
        super().__init__()  # create default constructor for QWidget
        self.initializeUI()

    def initializeUI(self):
        """
        Initialize the window and display its contents to the screen.
        """
        self.setGeometry(100, 100, 400, 300)
        self.setWindowTitle('Empty Window in PyQt')
        self.show()


# Run program
if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = EmptyWindow()
    sys.exit(app.exec_())
```

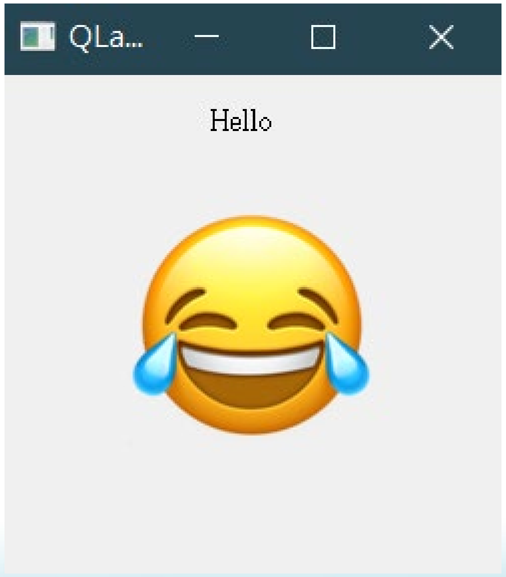


Classes for Building a GUI Window

- QApplication takes as an argument sys.argv
  - You can also pass in an empty list if you know that your program will not be taking any command-line arguments using
    - app = QApplication([])
- EmptyWindow Class inherits from QWidget, which is the base class for which all other user interface objects are derived.
- Call the show() method on the window object to display it to the screen.
- app.exec_() starts the event loop and will remain here until you quit the application.
- sys.exit() ensures a clean exit.

QLabel Widget

- labels.py
- QLabel object acts as a noneditable placeholder to display text, images, or movies.
- QtGui handles numerous graphical elements. QPixmap is a Qt class that is optimized for showing images on the screen.
- setText / setPixmap can determine QLabel context
- move(x, y) will move the widget to absolute position



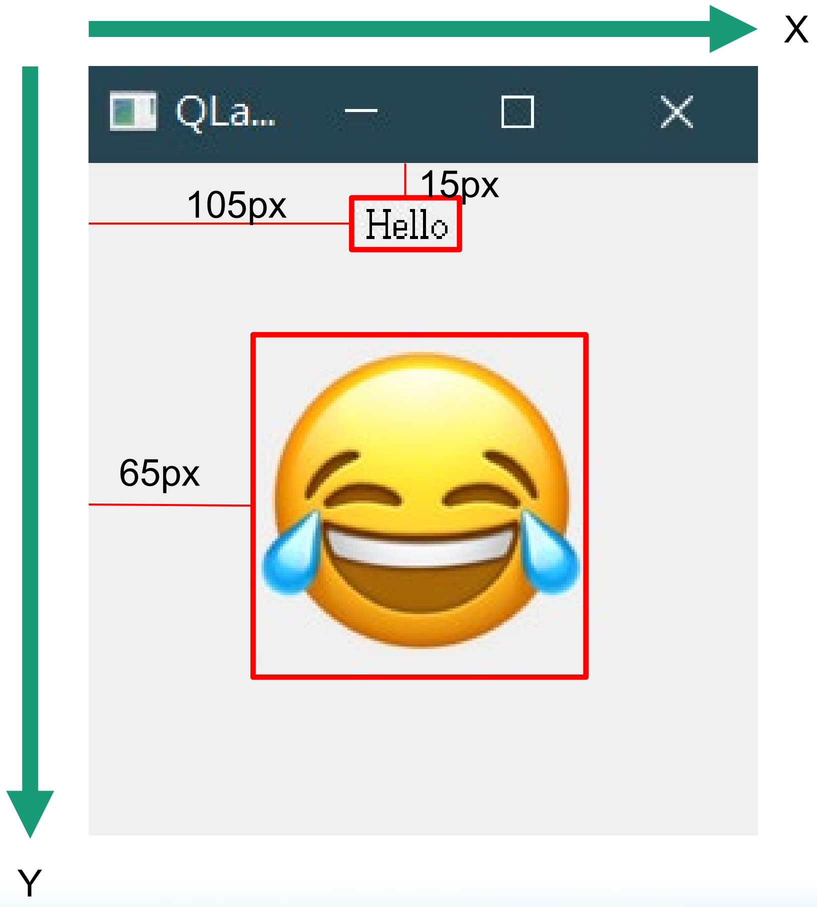
QLabel Widget

- Move the text “Hello” start at x=105, y=15

```python
text = QLabel(self)
text.setText("Hello")
text.move(105, 15)
```

- Move emoji to x=65, y=65

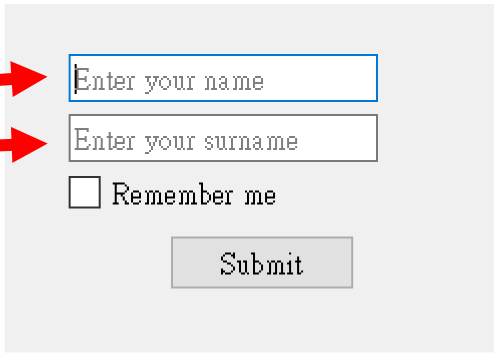
```python
emoji = QLabel(self)
pixmap = QPixmap(image)
emoji.setPixmap(pixmap)
emoji.move(65, 65)
```



QLineEdit Widget

- login_form.py
- QLineEdit create an areas where the user can input the text for their data on a single line.
- If you need multiple lines to enter text in, use QTextEdit
- setPlaceholderText() context will be clear after user starts input

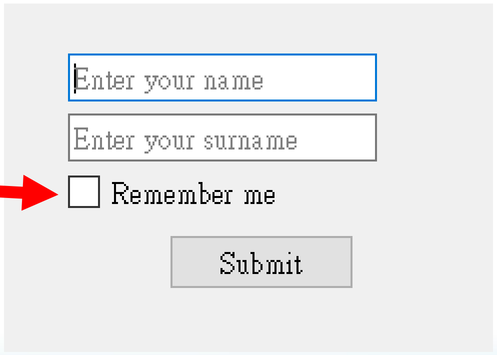
```python
self.name = QLineEdit(self)
self.surname = QLineEdit(self)
self.name.setPlaceholderText("Enter your name")
self.name.move(150, 50)
self.surname.setPlaceholderText("Enter your surname")
self.surname.move(150, 80)
```



QCheckBox Widget

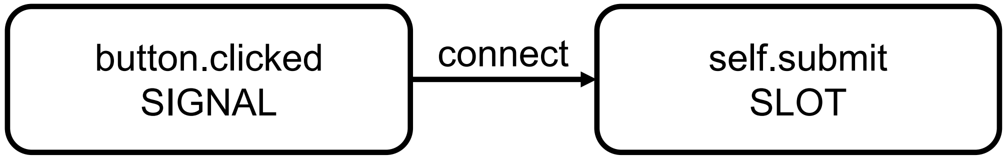
- login_form.py
- QCheckBox widget is a selectable button that generally has two states, on or off.
- The checkboxes in QCheckBox are not mutually exclusive, meaning you can select more than one checkbox at a time.
  - To make them mutually exclusive, add the checkboxes to a QButtonGroup object.
- isChecked() will return True if the checkbox is checked

```
self.remember = QCheckBox("Remember me", self)
self.remember.move(150, 110)
```



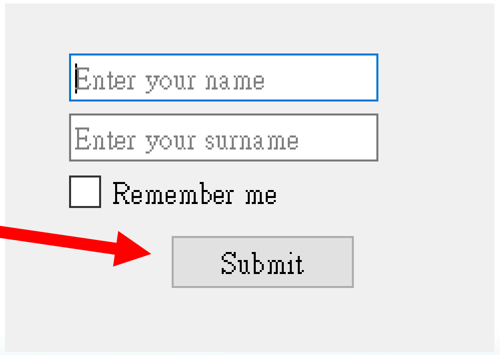
QPushButton Widget

- login_form.py
- QPushButton can be used to command the computer to perform some kind of operation or answer a question.
- When clicked, the QPushButton widget will send out a signal that can be connected to a function.



```
button = QPushButton("Submit", self)
button.move(200, 140)
button.clicked.connect(self.submit)
```

self.submit is a function pointer
because we do not call it right now



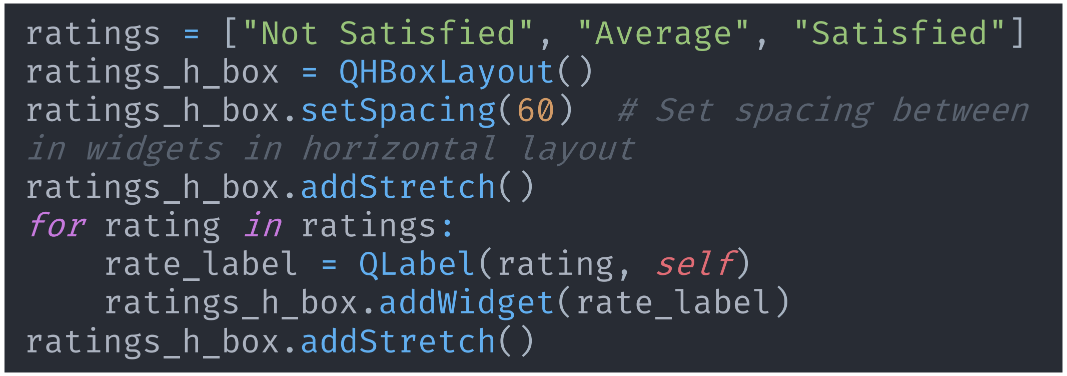
BoxLayout Widget

- QHBoxLayout – Used to arrange widgets horizontally from left to right in the window
- QVBoxLayout – Used to arrange widgets vertically from top to bottom in the window

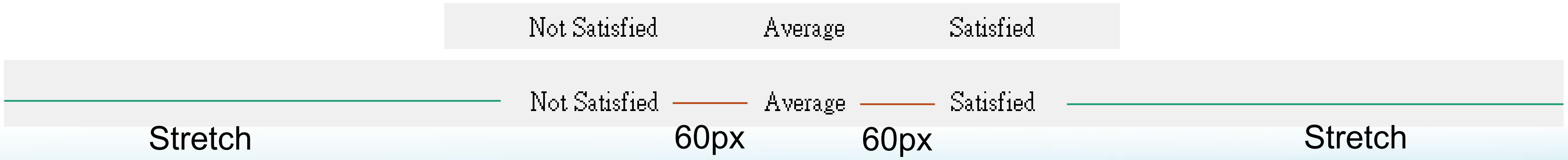
- To add widget inside BoxLayout
  - Create widget instance
  - Pass the child widget to addWidget() method of parent BoxLayout

BoxLayout Widget

- survey.py
- You can use for loop to push widget into BoxLayout
- setSpacing will create fixed space between child widgets
- addStretch will create dynamic space



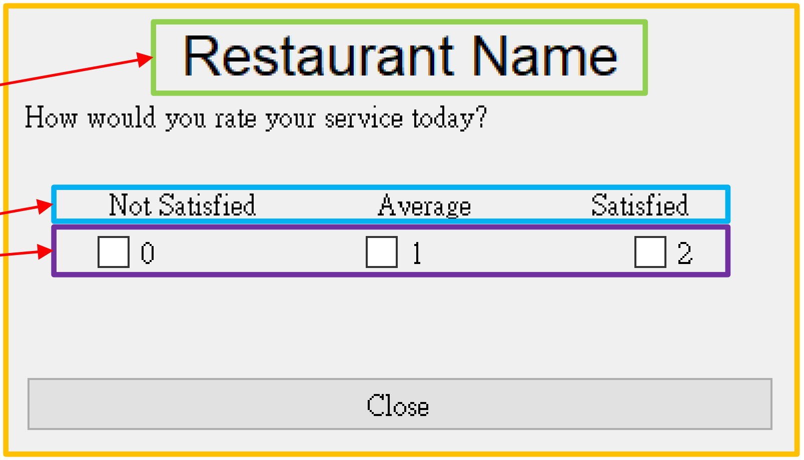
```
ratings = ["Not Satisfied", "Average", "Satisfied"]
ratings_h_box = QHBoxLayout()
ratings_h_box.setSpacing(60)  # Set spacing between
in widgets in horizontal layout
ratings_h_box.addStretch()
for rating in ratings:
    rate_label = QLabel(rating, self)
    ratings_h_box.addWidget(rate_label)
ratings_h_box.addStretch()
```



BoxLayout Widget

- survey.py
- The application consists of three separate QHBoxLayout objects – title_h_box, ratings_h_box, and cb_h_box – and a single QVBoxLayout layout, v_box.
- For this GUI, v_box will act as the container for all of the other widgets and layouts, arranged vertically from top to bottom.

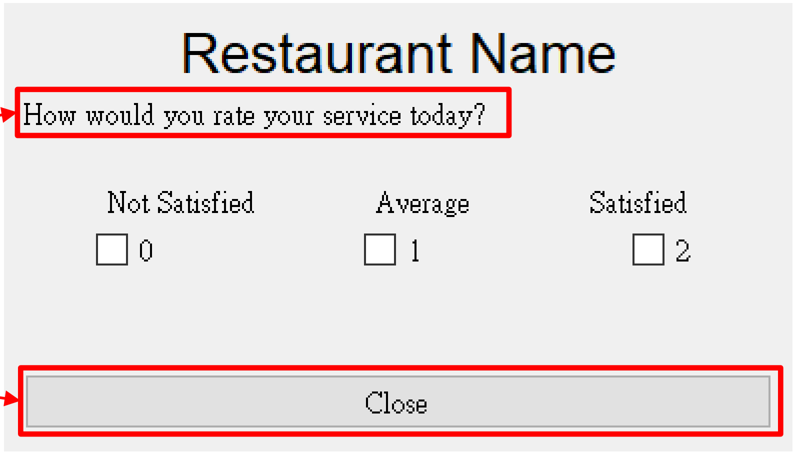
```
v_box = QVBoxLayout()
v_box.addLayout(title_h_box)
v_box.addWidget(question)
v_box.addStretch(1)
v_box.addLayout(ratings_h_box)
v_box.addLayout(cb_h_box)
v_box.addStretch(2)
v_box.addWidget(close_button)
```



BoxLayout Widget

- survey.py
- Not only nesting BoxLayout, you can just add widget inside boxlayout

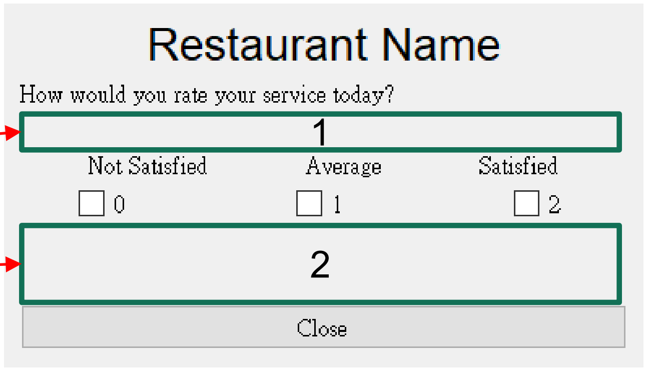
```
v_box = QVBoxLayout()
v_box.addLayout(title_h_box)
v_box.addWidget(question)
v_box.addStretch(1)
v_box.addLayout(ratings_h_box)
v_box.addLayout(cb_h_box)
v_box.addStretch(2)
v_box.addWidget(close_button)
```

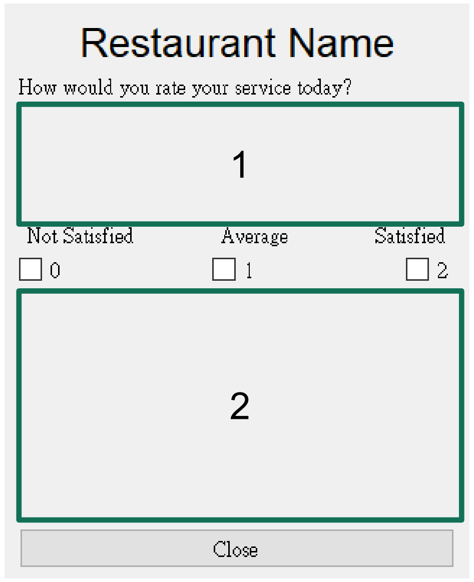


BoxLayout Widget

- survey.py
- addStretch will add space based on given proportion
  - Remaining space will be adjusted dynamically
  - If not specified, is addStretch(1)

```
v_box = QVBoxLayout()
v_box.addLayout(title_h_box)
v_box.addWidget(question)
v_box.addStretch(1)
v_box.addLayout(ratings_h_box)
v_box.addLayout(cb_h_box)
v_box.addStretch(2)
v_box.addWidget(close_button)
```



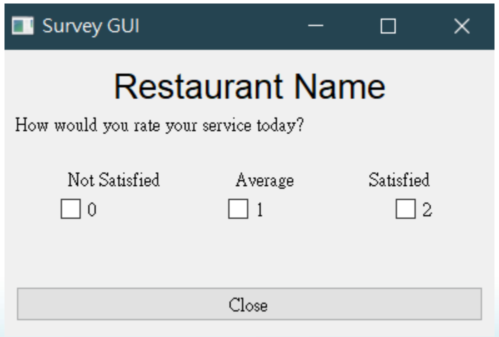


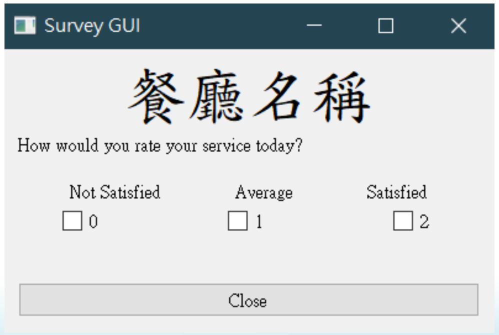
QFont Widget

- survey.py
- Import from QtGui library
- QLabel has a addFont method, accept QFont object
  - QFont(<font name>, <font size>)
- UTF-8 supported

```
from PyQt5.QtGui import Qfont
...

title = QLabel("Restaurant Name")
title.setFont(QFont('Arial', 17)
```

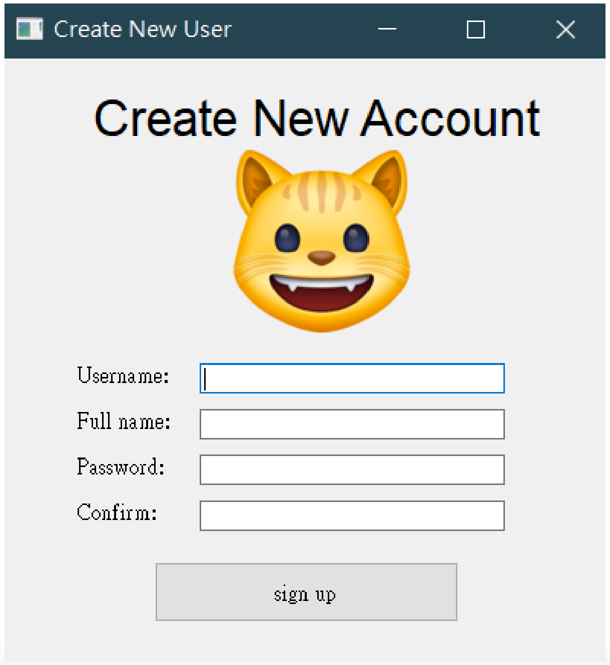
```
from PyQt5.QtGui import Qfont
...

title = QLabel("餐廳名稱")
title.setFont(QFont('標楷體', 30))
```





Lab1-1 Registration GUI

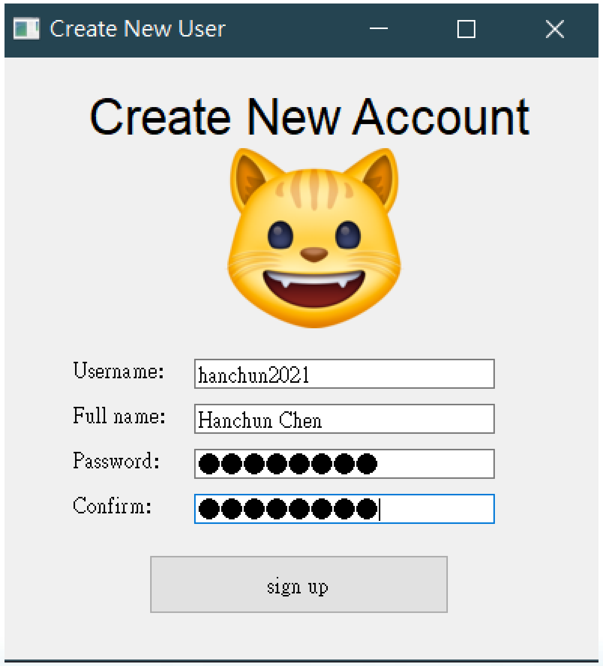
- lab1_registration.py
- Recreate this GUI
  - You should be able to edit all input field
  - Put whatever user image you like on top of inputs
  - The password field should be censored



Create New User

Create New Account

Username:	
Full name:	
Password:	
Confirm:	



Create New User

Create New Account

Username:	hanchun2021
Full name:	Hanchun Chen
Password:	●●●●●●●●●
Confirm:	●●●●●●●●●

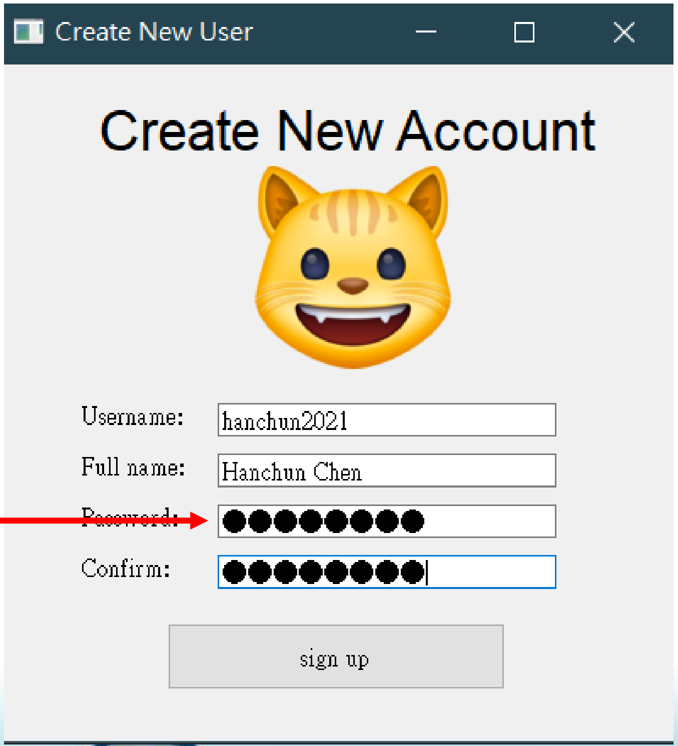
Lab1-1 Registration GUI

- Hint:
  - No restriction using absolute layout or BoxLayout
  - You can resize QLineEdit using resize() method

```
self.name_entry = QLineEdit(self)
self.name_entry.resize(200, 20)
```

  - To make QLineEdit as password field, do this:

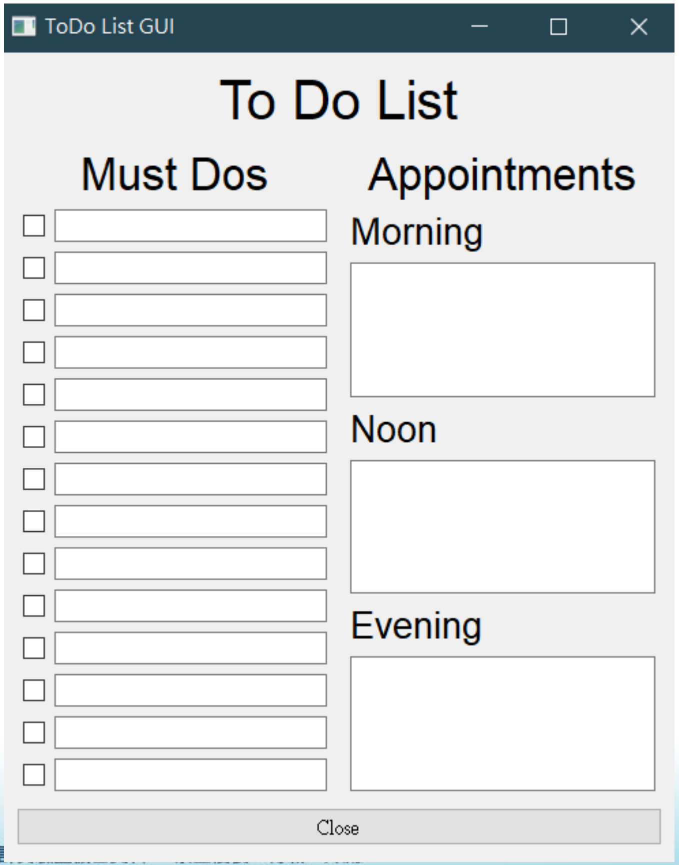
```
self.pswd_entry = QLineEdit(self)
self.pswd_entry.setEchoMode(QLineEdit.Password)
```

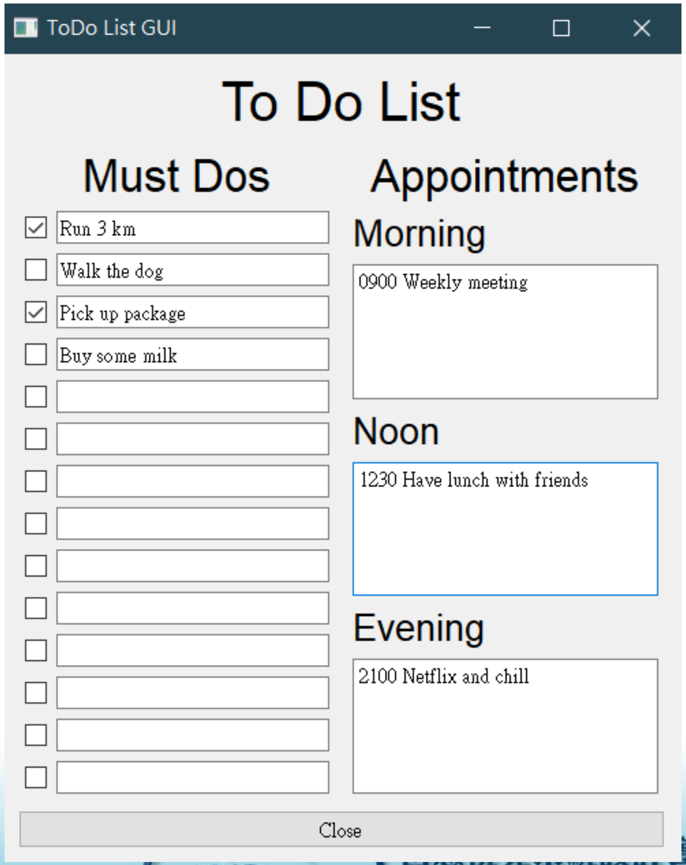


Username: hanchun2021
Full name: Hanchun Chen
Password: ●●●●●●●●●
Confirm: ●●●●●●●●●

Lab1-2 ToDo List

- lab1_todolist.py
- Recreate this GUI
  - You should be able to edit all input field and checkbox
  - Be careful not to use the same object multiple times





- [x] Run 3 km
- [ ] Walk the dog
- [x] Pick up package
- [ ] Buy some milk

Morning: 0900 Weekly meeting

Noon: 1230 Have lunch with friends

Evening: 2100 Netflix and chill

Lab1-2 ToDo List

- Hint:
  - Use setAlignment and Qt.AlignCenter to quickly align title at center

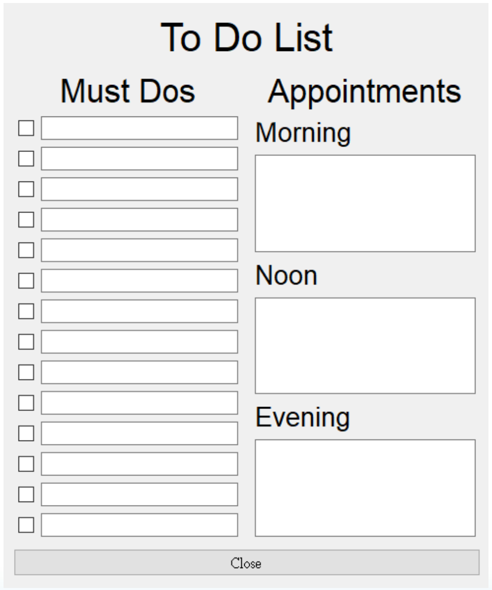
```python
from PyQt5.QtCore import Qt
...

todo_title = QLabel("To Do List")
todo_title.setFont(QFont('Arial', 24))
todo_title.setAlignment(Qt.AlignCenter)
```

  - Use QTextEdit() for multiline input box

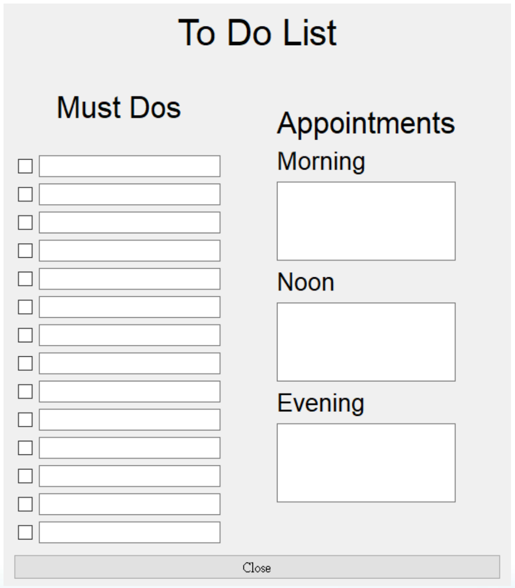
Lab1-2 ToDo List

- Hint:
  - You can use setContentsMargins(<left>, <top>, <right>, <bottom>), to create buffer zone around the BoxLayout

```
appt_v_box = QVBoxLayout()
appt_v_box.setContentsMargins(5, 5, 5, 5)
```

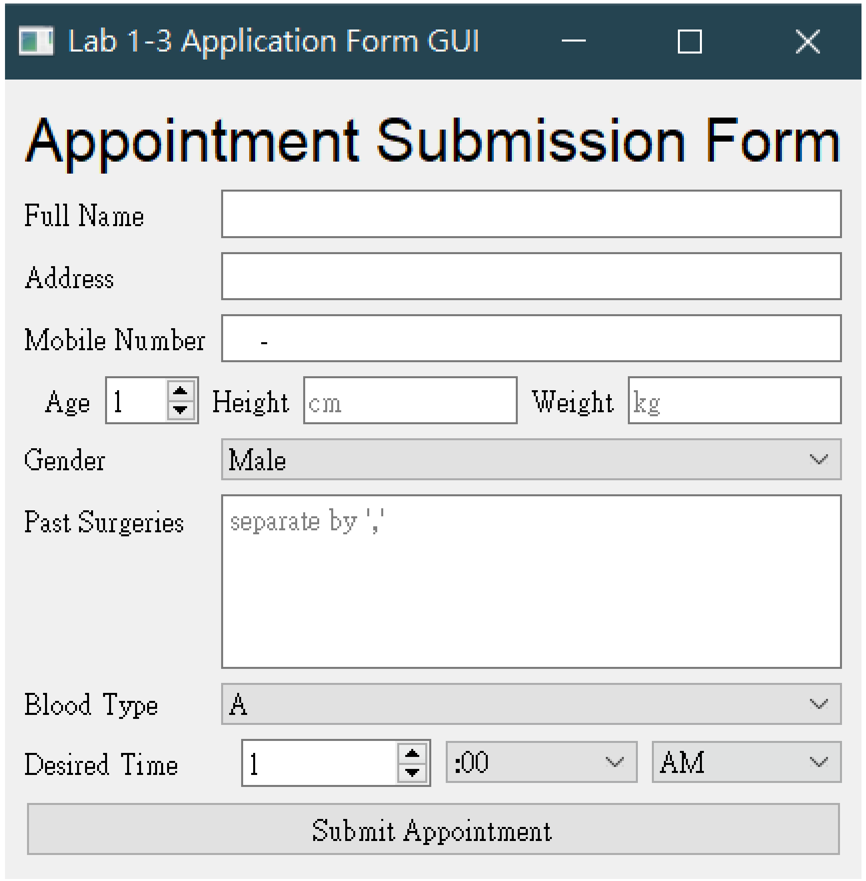


```
appt_v_box = QVBoxLayout()
appt_v_box.setContentsMargins(50, 50, 50, 50)
```



Lab1-3 Application Form (Challenge)

- lab1_application.py
- Recreate this GUI
  - No need to reach pixel perfect.
  - But all widgets and their relative location should be the same.


Lab 1-3 Application Form GUI

Appointment Submission Form

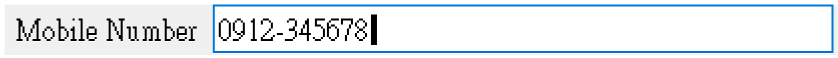
Full Name	
Address	
Mobile Number	-
Age 1	Height cm Weight kg
Gender	Male
Past Surgeries	separate by ','
Blood Type	A
Desired Time	1 :00 AM

Submit Appointment

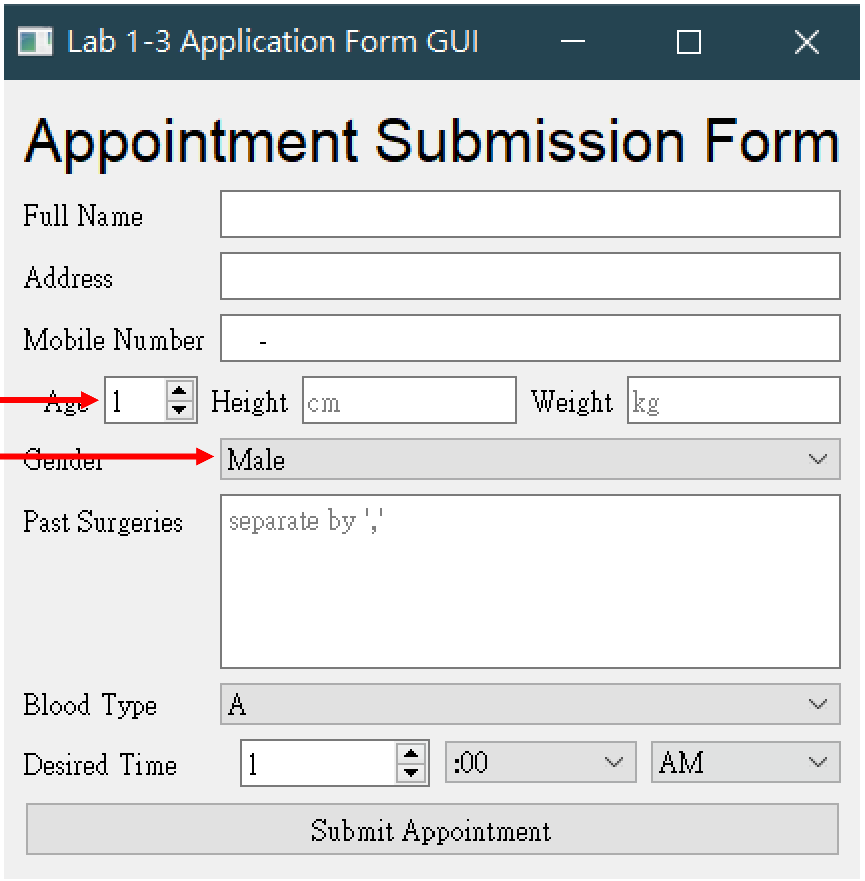
Lab1-3 Application Form (Challenge)

- Hint:
- This is QSpinBox
  - Try to figure it out by yourself
- This is QComboBox
  - Just Google it
- You can preformat QLineEdit

```
mobile_num = QLineEdit()
mobile_num.setInputMask("0000-000000;")
```



Mobile Number 0912-345678



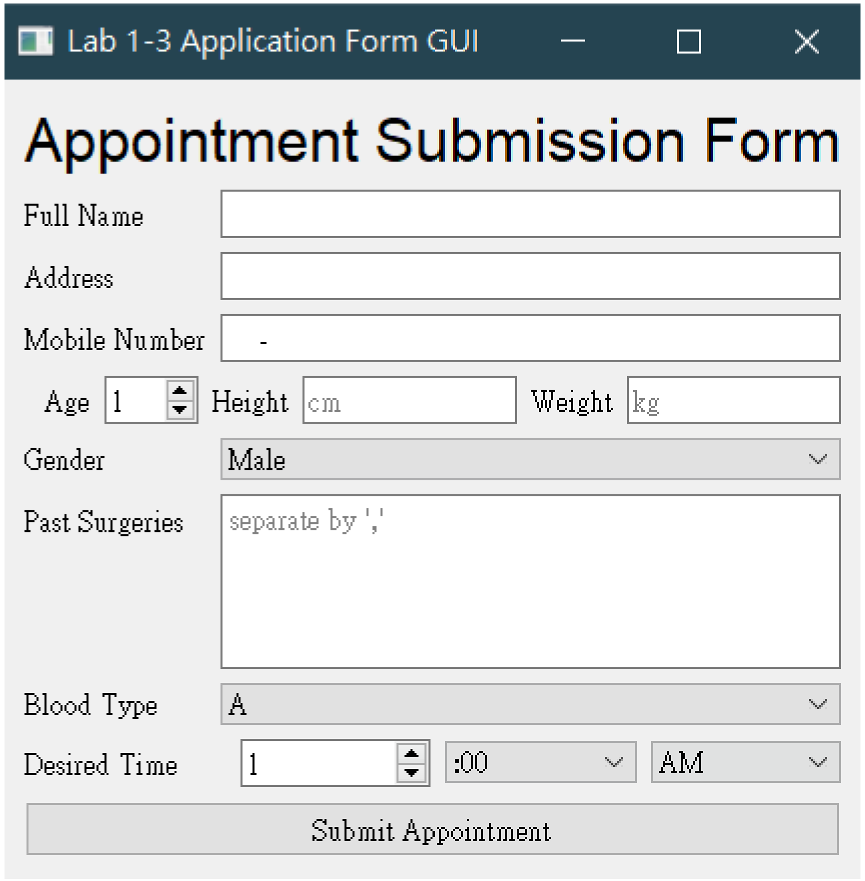
Lab 1-3 Application Form GUI

Appointment Submission Form

Full Name	
Address	
Mobile Number	-
Age	1
Height	cm
Weight	kg
Gender	Male
Past Surgeries	separate by ','
Blood Type	A
Desired Time	1 :00 AM

Lab1-3 Application Form (Challenge)

- Hint:
- You may want to checkout QFormLayout
    - It can save you a lot of time to build this


Lab 1-3 Application Form GUI

Appointment Submission Form

Full Name

Address

Mobile Number -

Age 1 Height cm Weight kg

Gender Male

Past Surgeries separate by ','

Blood Type A

Desired Time 1 :00 AM

Submit Appointment

Demo

- 本次Lab以個人為單位
- 配分
  - Lab1-1: 40%
  - Lab1-2: 40%
  - Lab1-3: 20%
- Demo
  - 完成Lab後，舉手呼叫助教們demo
  - 多個小題可以分次demo
  - 呈現程式執行結果
- 最後登記時間：21:20