LẬP TRÌNH PYTHON
PyQt6

NGUYỄN HẢI TRIỀU[1]

[1]Bộ môn Kỹ thuật phần mềm,

Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, February 2022

Nội dung

Concepts for Creating Good Interface Design [1]

The following is a list of concepts to consider when designing your own UI:

1. Clarity: Using clear language, hierarchy, and flow with visual elements to avoid ambiguity.
2. Conciseness: Simplifying the layout to include only what the user needs to see or interact with at a given time in order to be brief, but also comprehensive
3. Consistency: Design the UI so that there is consistency across the application.
4. Efficiency: Utilizing good design and shortcuts to help the user improve productivity
5. Familiarity: Consider elements that users normally see in other UIs and how they would expect them to perform in your applications.
6. Responsive

PyQt6

Description

Qt is set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems.

Installation

pip install PyQt6

Qt Designer

Description

Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. You can compose and customize your windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, and test them using different styles and resolutions.

Installation

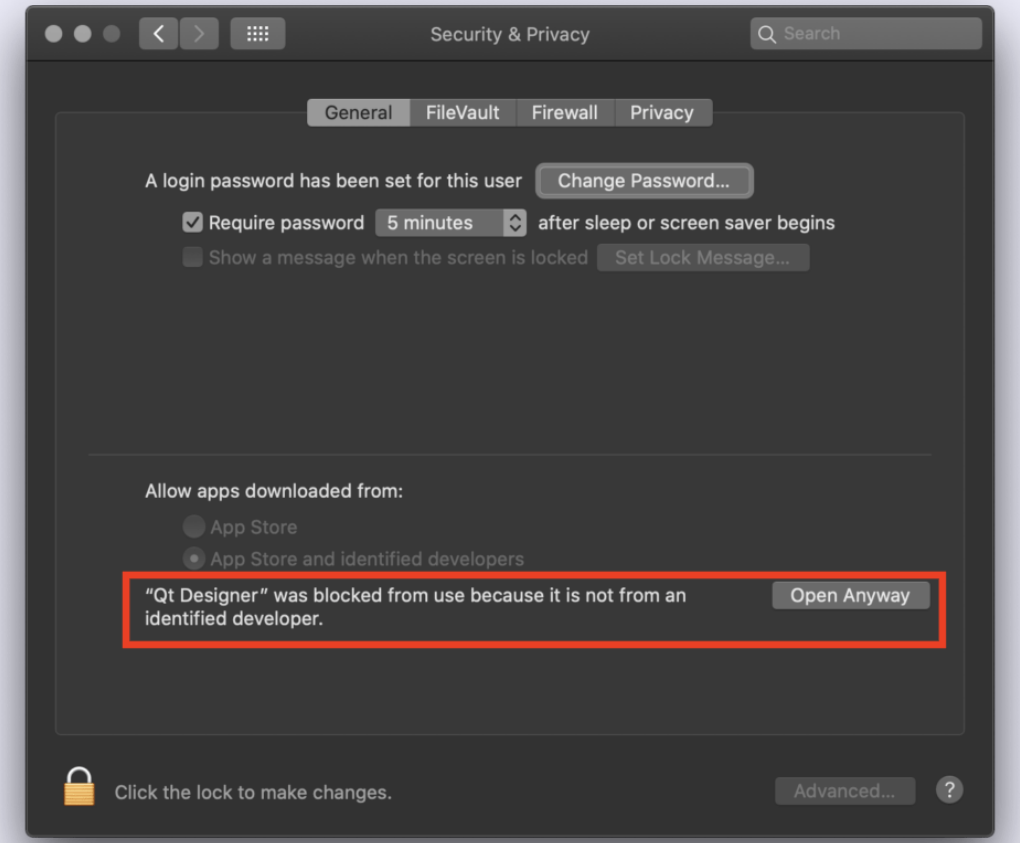
Here are small, standalone installers of Qt Designer for Windows and Mac:
`https://build-system.fman.io/qt-designer-download`
or

```
pip install PySide6
```

Qt Designer

Note for Mac OS

To run in MacOS, selecting: System Preferences > Security & Privacy > General > Open Anyway for Qt Designer



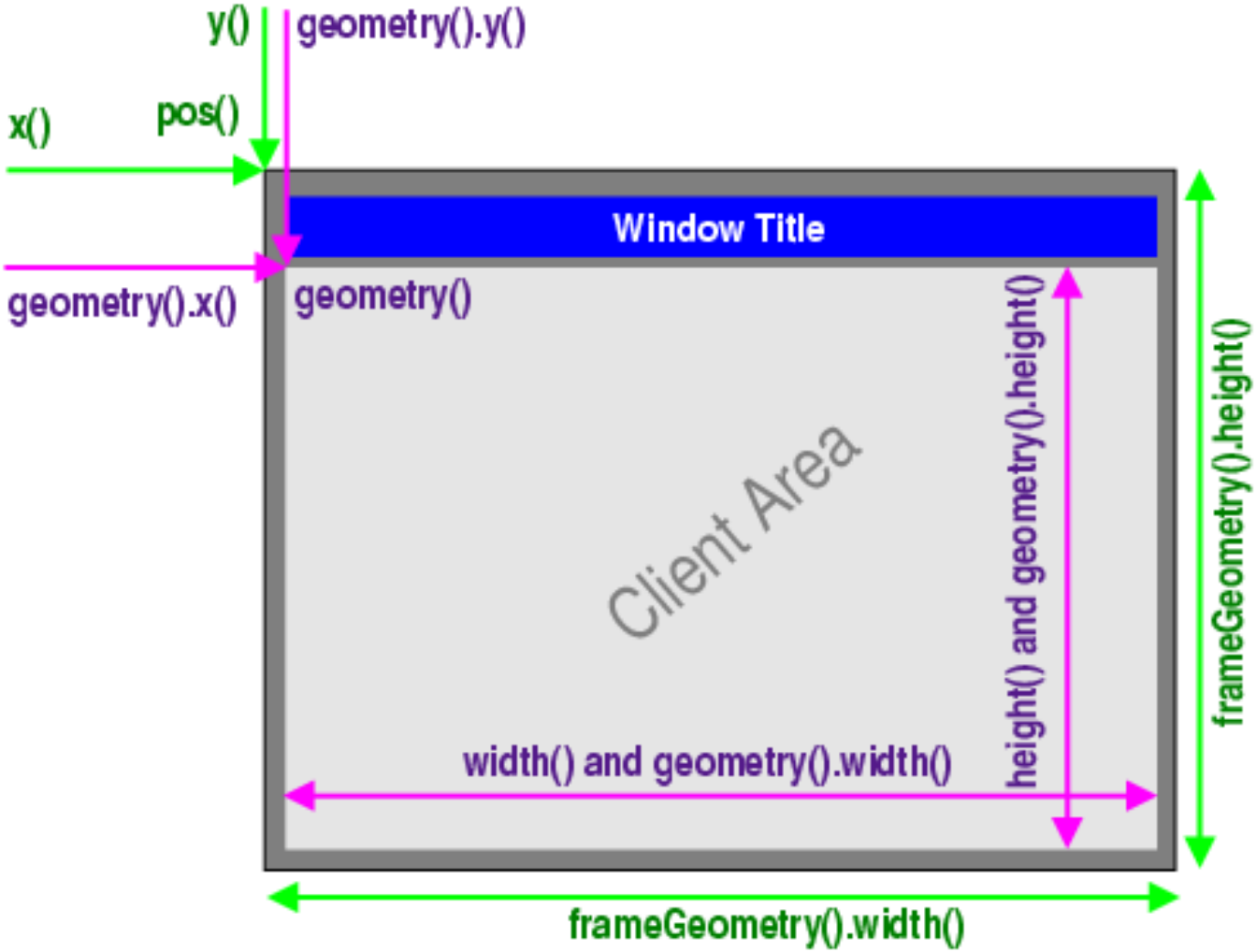
A GUI application generally consists of *a main window* and possibly one or more *dialog boxes*.

1. The main window is where the user will spend most of their time when using your application and can **consist of a menu bar, a status bar, and other widgets**.
2. Dialog boxes typically are made up of **text, maybe one or more widgets for collecting information, and buttons**.

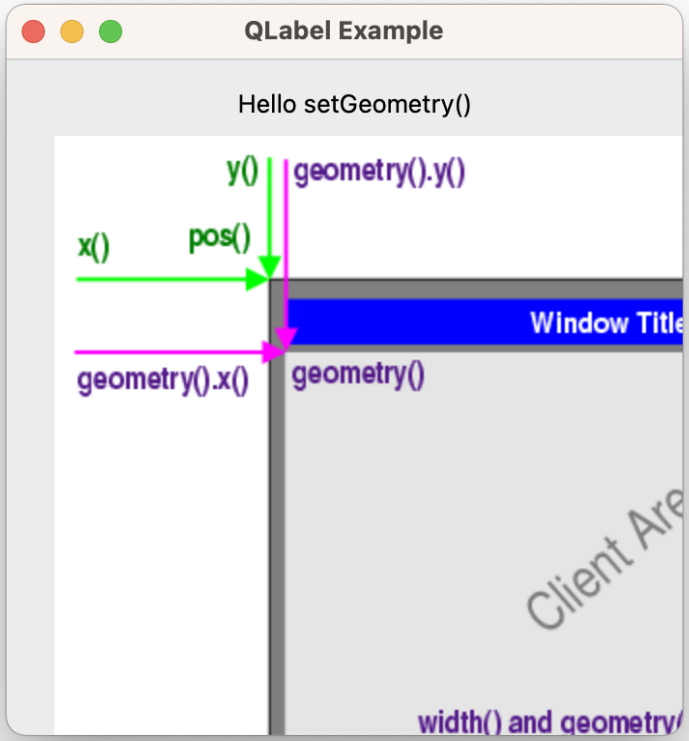
```python
#The sys module can be used in PyQt to pass command line arguments to our applications and to close them
import sys
#The QtWidgets module provides the widget classes that you will need to create desktop-style GUIs
from PyQt6.QtWidgets import QApplication, QWidget
#QApplication is responsible for managing the application's main event loop and widget initialization and finalization
class EmptyWindow(QWidget):
    '''Constructor for Empty Window Class'''
    def __init__(self) -> None:
        super().__init__()
        self.initializeUI()
    def initializeUI(self):
        '''Set up the application'''
        #The method setGeometry() defines the location of the window on your computer screen and its dimensions, width and height
        self.setGeometry(500, 200,400,400)
        # The setWindowTitle() method is used to set the title of the window.
        self.setWindowTitle("My First App")
        self.show() # Display the window on the screen
if __name__=='__main__':
    #The main event loop is where user interactions in the GUI window, such as clicking on a button, are managed
    app=QApplication(sys.argv) # or app=QApplication([])
    #a window object which  inherits from the class we created, EmptyWindow.
    window=EmptyWindow()
    sys.exit(app.exec())
    #The method exec() starts the application's event loop and will remain here until you quit the application. The function sys.exit() ensures a clean exit.
```

Explanation for Creating an Empty Window:

Hình 1: Window Geometry



A QLabel object acts as a non-editable placeholder to display plain or rich text, hyperlinks, images, or GIFs. It is also useful for creating labels around other widgets to specify their roles or give them titles.



Hình 2: Example of using QLabel widgets to place images and text in a window

```python
import sys
from PyQt6.QtWidgets import QApplication, QWidget, QLabel
#The QtGui module handles numerous graphic elements used in GUIs
from PyQt6.QtGui import QPixmap
class MainWindow(QWidget): #MainWindow class that inherits from the QWidget
    def __init__(self):
        super().__init__()
        self.initializeUI()
    def initializeUI(self):
        """Set up the application's GUI."""
        self.setGeometry(500, 100, 350, 350)
        self.setWindowTitle("QLabel Example")
        self.setUpMainWindow()
        self.show()
    def setUpMainWindow(self):
        hello_label=QLabel(self)
        hello_label.setText("Hello setGeometry()")
        hello_label.move(120,15)
        image="fig/geometry.png"
        try:
            with open(image):
                geo_label=QLabel(self)
                pixmap=QPixmap(image)
                geo_label.setPixmap(pixmap)
                geo_label.move(25,40)
        except FileExistsError as error:
            print(f"Image not found.\n Error:{error}")
if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    sys.exit(app.exec())
```

Example code for Using QLabel

Explanation for Using QLabel

1. **QPixmap** is a Qt class that is optimized for showing images on the screen and is useful for displaying an image on a QLabel object.
2. By passing self as a parameter to QLabel, you set the MainWindow class as the parent of the label
3. what the label will say using **setText()**
4. **move()** method to arrange the label in the window (absolute positioning)

A slightly more complex GUI: User Profile GUI

Hình 3: The User Profile GUI that displays a user's information

A slightly more complex GUI: User Profile GUI

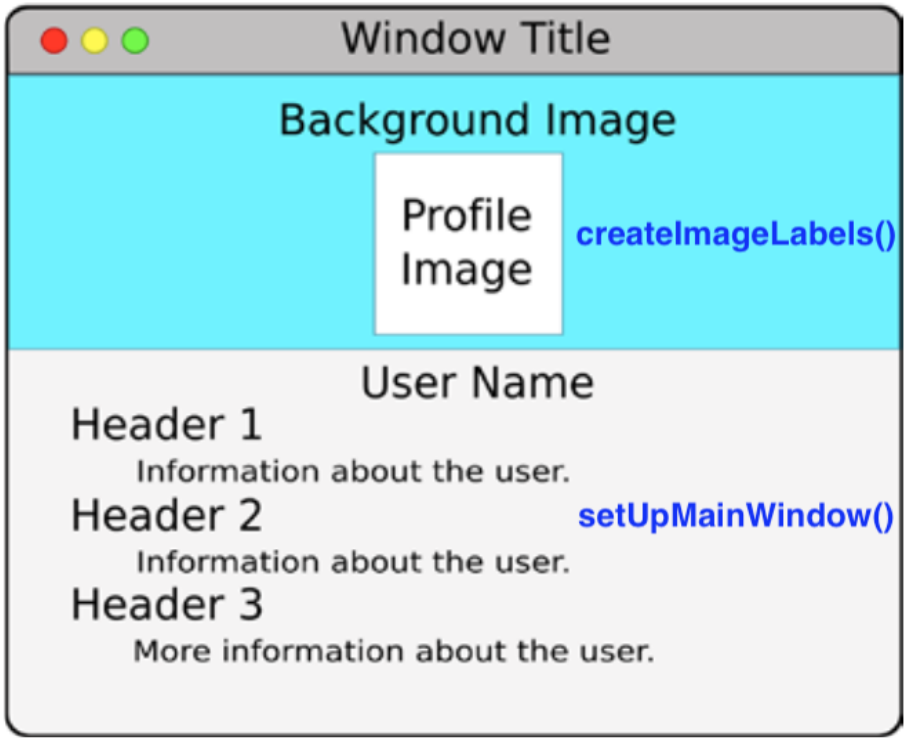
Designing the User Profile GUI

1. The user interface can be divided into two parts. The upper portion uses QLabel objects that display a profile image that lies on top of a background image.
2. The bottom portion shows the user's information with multiple QLabel widgets, with the textual information arranged vertically and broken down into smaller sections, delineated by the use of different font sizes.

A slightly more complex GUI: User Profile GUI

Schematic for the User Profile GUI

MainWindow(QWidget)



A slightly more complex GUI: User Profile GUI

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget, QLabel
3  from PyQt6.QtGui import QFont, QPixmap
4  class MainWindow(QWidget):
5      def __init__(self):
6          super().__init__()
7          self.initializeUI()
8
9      def initializeUI(self):
10         """Set up the application's GUI."""
11         self.setGeometry(50, 50, 250, 400)
12         self.setWindowTitle("User Profile GUI")
13
14         self.setUpMainWindow()
15         self.show()
16
17 if __name__ == '__main__':
18     app = QApplication(sys.argv)
19     window = MainWindow()
20     sys.exit(app.exec())
```

- QFont class from the QtGui module, which allows for us to modify the size and types of fonts in our application

```
1       def createImageLabels(self):
2           """Open image files and create image labels."""
3           images = ["images/skyblue.png",
4                     "images/profile_image.png"]
5
6           for image in images:
7               try:
8                   with open(image):
9                       label = QLabel(self)
10                      pixmap = QPixmap(image)
11                      label.setPixmap(pixmap)
12                      if image == "images/profile_image.png":
13                          label.move(80, 20)
14              except FileNotFoundError as error:
15                  print(f"Image not found.\nError: {error}")
```

Before creating setUpMainWindow(), let's create a separate method in MainWindow, that will handle loading the different images and creating QLabel objects to display them.

```
def setUpMainWindow(self):
        """Create the labels to be displayed in the window."""
        self.createImageLabels()
        #user_label
        user_label = QLabel(self)
        user_label.setText("John Doe")
        user_label.setFont(QFont("Arial", 20))
        user_label.move(85, 140)
        #bio_label
        bio_label = QLabel(self)
        bio_label.setText("Biography")
        bio_label.setFont(QFont("Arial", 17))
        bio_label.move(15, 170)
        #about_label
        about_label = QLabel(self)
        about_label.setText("I'm a Software Engineer with 10 years\
            experience creating awesome code.")
        about_label.setWordWrap(True)
        about_label.move(15, 190)
        #skills_label
        skills_label = QLabel(self)
        skills_label.setText("Skills")
        skills_label.setFont(QFont('Arial', 17))
        skills_label.move(15, 240)
        #languages_label
        languages_label = QLabel(self)
        languages_label.setText("Python  |  PHP  |  SQL  |  JavaScript")
        languages_label.move(15, 260)
        #experience_label
        experience_label = QLabel(self)
        experience_label.setText("Experience")
```

```
32          experience_label.setFont(QFont("Arial", 17))
33          experience_label.move(15, 290)
34          #developer_label
35          developer_label = QLabel(self)
36          developer_label.setText("Python Developer")
37          developer_label.move(15, 310)
38          #dev_dates_label
39          dev_dates_label = QLabel(self)
40          dev_dates_label.setText("Mar 2011 - Present")
41          dev_dates_label.setFont(QFont("Arial", 10))
42          dev_dates_label.move(15, 330)
43          #driver_label
44          driver_label = QLabel(self)
45          driver_label.setText("Pizza Delivery Driver")
46          driver_label.move(15, 350)
47          #driver_dates_label
48          driver_dates_label = QLabel(self)
49          driver_dates_label.setText("Aug 2015 - Dec 2017")
50          driver_dates_label.setFont(QFont("Arial", 10))
51          driver_dates_label.move(15, 370)
```

Several QLabel objects for showing text are instantiated. For example, the user_label displays the user's name using setText() in the window. You can set a QLabel widget's font with the method setFont(). The user_label is then centered in the window using move(). Other labels are created in a similar manner.

A slightly more complex GUI: User Profile GUI

Bài tập

Sinh viên sử dụng QLabel, thực hiện xây dựng ứng dụng hiển thị Profile cá nhân.

Event Handlers and Signals and Slots

Definition of Event

GUIs are event driven, meaning that they respond to events that are created by the user, from a keyboard or a mouse, or by events caused by the system, such as a timer or when connecting to Bluetooth.

Event handling

In Qt, special kinds of events are even generated to handle communication between widgets.

- the application needs to listen for those events and respond to them appropriately.
- event handling is handled in one of two ways – either through event handlers or with **signals and slots**.

Event Handlers and Signals and Slots I

Event handling in Qt

The communication between objects in Qt, such as widgets, is handled by signals and slots

- **Signals** are generated whenever an event occurs, such as when a button is clicked or a checkbox is toggled on or off.
- **Slots** are the methods that are connected to an event and executed in response to the signal. Slots can either be *built-in PyQt functions* or *Python functions* that you *create yourself*.

Event Handlers and Signals and Slots II

Example

Whenever a user clicks a button in the window, that button click will send out, or emit, a signal:

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

- Here, button is a widget, and **clicked is the signal**.
- In order to make use of that signal, we must use connect() to call some function, which in this case is **buttonClicked(), which is the slot**.
- The buttonClicked() method could then perform some action, such as opening a new window.
- Many signals also pass along additional information to the slot, such as a Boolean value that tells whether or not the button was pressed.

The QPushButton Widget

QPushButton

The QPushButton widget can be used to perform actions and make choices. When you **click on the QPushButton widget, it sends out a signal that can be connected to a function.**

Example

Set up a QPushButton that, when clicked, uses signals and slots to change the text of a QLabel widget and shows how to handle closing an application's main window.

QPushButton: Example I

```
1  import sys
2  from PyQt6.QtWidgets import (QApplication, QWidget, QLabel,
3      QPushButton)
4  from PyQt6.QtCore import Qt
5
6  class MainWindow(QWidget):
7
8      def __init__(self):
9          super().__init__()
10         self.initializeUI()
11
12     def initializeUI(self):
13         """Set up the application's GUI."""
14         self.setGeometry(100, 100, 250, 150)
15         self.setWindowTitle("QPushButton Example")
16
17         self.setUpMainWindow()
18         self.show()
19
20     def setUpMainWindow(self):
21         """Create and arrange widgets in the main window."""
22         self.times_pressed = 0
23
24         self.name_label = QLabel(self)
25         self.name_label.setText("Don't push the button.")
26         self.name_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
27         self.name_label.move(60, 30)
28
```

QPushButton: Example II

```
29          self.button = QPushButton("Push Me", self)
30          self.button.move(80, 70)
31          self.button.clicked.connect(self.buttonClicked)
32
33      def buttonClicked(self):
34          """Handle when the button is clicked.
35          Demonstrates how to change text for widgets,
36          update their sizes and locations, and how to
37          close the window due to events."""
38          self.times_pressed += 1
39
40          if self.times_pressed == 1:
41              self.name_label.setText("Why'd you press me?")
42          if self.times_pressed == 2:
43              self.name_label.setText("I'm warning you.")
44              self.button.setText("Feelin' Lucky?")
45              self.button.adjustSize()
46              self.button.move(70, 70)
47          if self.times_pressed == 3:
48              print("The window has been closed.")
49              self.close()
50
51  if __name__ == '__main__':
52      app = QApplication(sys.argv)
53      window = MainWindow()
54      sys.exit(app.exec())
```

QPushButton: Example III

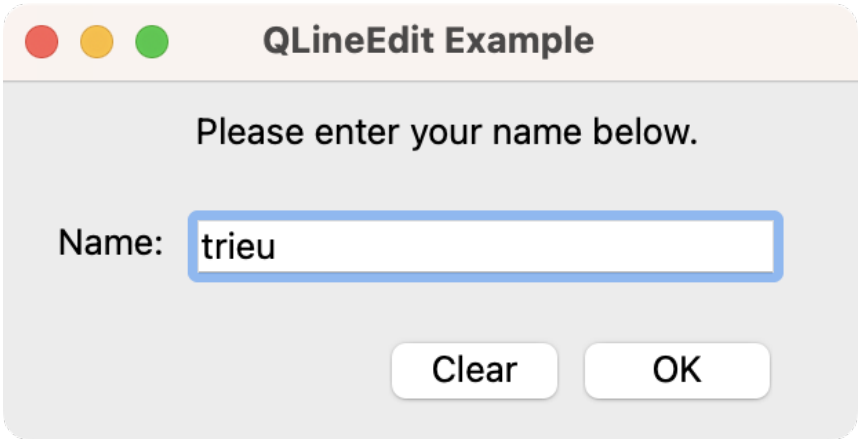
Explanation of QPushButton code

- The variable times_pressed will be used to keep track of how many times button is pressed.
- we can instead pass the text we want to display **as the first argument when instantiating the QLabel object**.
- Clicking on the button will emit the clicked signal, which is **connected to the buttonClicked() slot**
- **setAlignment()**: align the contents of widgets that display text (AlignLeft, AlignRight, AlignTop, AlignBottom...)
- self.close() is referring to the main window and closes the application

The QLineEdit Widget

With the QLineEdit widget, It is often **necessary for a user to input a single line of text**, such as a username, a password or collecting data from someone. QLineEdit also **supports normal text editing functions such as cut, copy, and paste, and redo or undo**.

Hình 4: QLineEdit and QPushButton widgets used for collecting and clearing text



The QLineEdit Widget I

```
1  import sys
2  #import different widget classes, including QLabel, QLineEdit, and QPushButton, as well
       as Qt from the QtCore module
3  from PyQt6.QtWidgets import (QApplication, QWidget,
4      QLabel, QLineEdit, QPushButton)
5  from PyQt6.QtCore import Qt
6
7  class MainWindow(QWidget):
8
9      def __init__(self):
10         super().__init__()
11         self.initializeUI()
12
13     def initializeUI(self):
14         """Set up the application's GUI."""
15         self.setMaximumSize(310, 130) #Sets the widget's maximum size
16         self.setWindowTitle("QLineEdit Example")
17
18         self.setUpMainWindow()
19         self.show()
20
21     def setUpMainWindow(self):
22         """Create and arrange widgets in the main window."""
23         QLabel("Please enter your name below.",
24                self).move(70, 10)
25         name_label = QLabel("Name:", self)
26         name_label.move(20, 50)
27
```

The QLineEdit Widget II

```
28          self.name_edit = QLineEdit(self)
29          self.name_edit.resize(210, 20)
30          self.name_edit.move(70, 50)
31
32          clear_button = QPushButton("Clear", self)
33          clear_button.move(140, 90)
34          clear_button.clicked.connect(self.clearText)
35
36          accept_button = QPushButton("OK", self)
37          accept_button.move(210, 90)
38          accept_button.clicked.connect(self.acceptText)
39
40      def clearText(self):
41          """Clear the QLineEdit input field."""
42          self.name_edit.clear()
43
44      def acceptText(self):
45          """Accept the user's input in the QLineEdit
46          widget and close the program."""
47          print(self.name_edit.text())
48          self.close()
49
50  if __name__ == '__main__':
51      app = QApplication(sys.argv)
52      window = MainWindow()
53      sys.exit(app.exec())
```

The QLineEdit Widget III

Explanation of QLineEdit Widget code

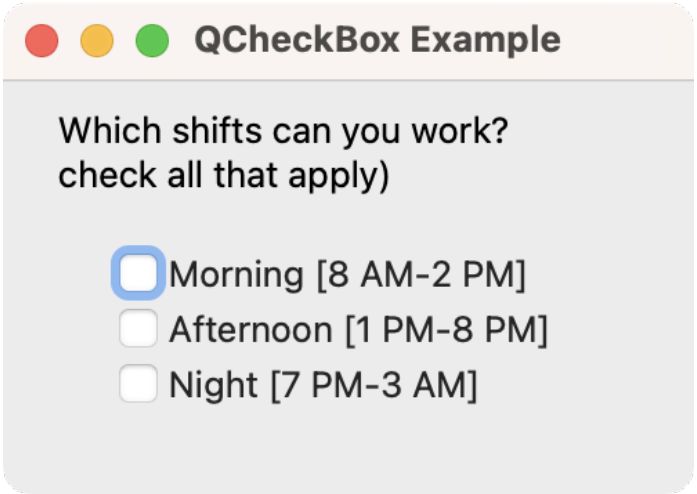
First, we create *two QLabel objects*, then *a QLineEdit widget*, and *two QPushButton objects*.

- The QLineEdit object, name_edit, is an example of how to modify a widget's size using the resize() method. You'll need to specify the widget's **desired height and width values**.
- The clear_button and accept_button objects are connected to the **clearText()** and **acceptText()** slots.
- When clear_button is clicked, it *emits a signal that is connected to the clearText() slot*, and the name_edit widget will react to the signal and clear its current text.
- print QLineEdit text and closing the window.

The QCheckBox Widget

The user is allowed to select all checkboxes that apply to them, and each time the user clicks a checkbox, we call a method to show how to determine the widget's current state.

Hình 5: Example that uses QCheckBox widgets



Example using QCheckBox widget I

```
1  import sys
2  from PyQt6.QtWidgets import (QApplication, QWidget, QCheckBox,
3      QLabel)
4  from PyQt6.QtCore import Qt
5
6  class MainWindow(QWidget):
7
8      def __init__(self):
9          super().__init__()
10         self.initializeUI()
11
12     def initializeUI(self):
13         """Set up the application's GUI."""
14         self.setGeometry(100, 100, 250, 150)
15         self.setWindowTitle("QCheckBox Example")
16
17         self.setUpMainWindow()
18         self.show()
19
20     def setUpMainWindow(self):
21         """Create and arrange widgets in the main window."""
22         header_label = QLabel("Which shifts can you work? \
23                                (Please check all that apply)", self)
24         header_label.setWordWrap(True)
25         header_label.move(20, 10)
26
27         # Set up the checkboxes
28         morning_cb = QCheckBox("Morning [8 AM-2 PM]", self)
```

Example using QCheckBox widget II

```
29          morning_cb.move(40, 60)
30          #morning_cb.toggle() # Uncomment to start checked
31          morning_cb.toggled.connect(self.printSelected)
32
33          after_cb = QCheckBox("Afternoon [1 PM-8 PM]", self)
34          after_cb.move(40, 80)
35          after_cb.toggled.connect(self.printSelected)
36
37          night_cb = QCheckBox("Night [7 PM-3 AM]", self)
38          night_cb.move(40, 100)
39          night_cb.toggled.connect(self.printSelected)
40
41      def printSelected(self, checked):
42          """Print the text of a QCheckBox object when selected
43          or deselected. Use sender() to determine which widget
44          is sending the signal."""
45          sender = self.sender()
46          if checked:
47              print(f"{sender.text()} Selected.")
48          else:
49              print(f"{sender.text()} Deselected.")
50
51  if __name__ == '__main__':
52      app = QApplication(sys.argv)
53      window = MainWindow()
54      sys.exit(app.exec())
```

Example using QCheckBox widget III

Explanation for Using QCheckBox

Three QCheckBox objects are created

- For labels with longer text that won't fit on one line, use the setWordWrap() method.
- The QCheckBox method toggle() can be used to toggle the checkbox on or off.
- When a checkbox is selected, rather than using the clicked signal like with QPushButton, use toggled to emit a signal that is connected to the slot printSelected()

The QMessageBox Dialog

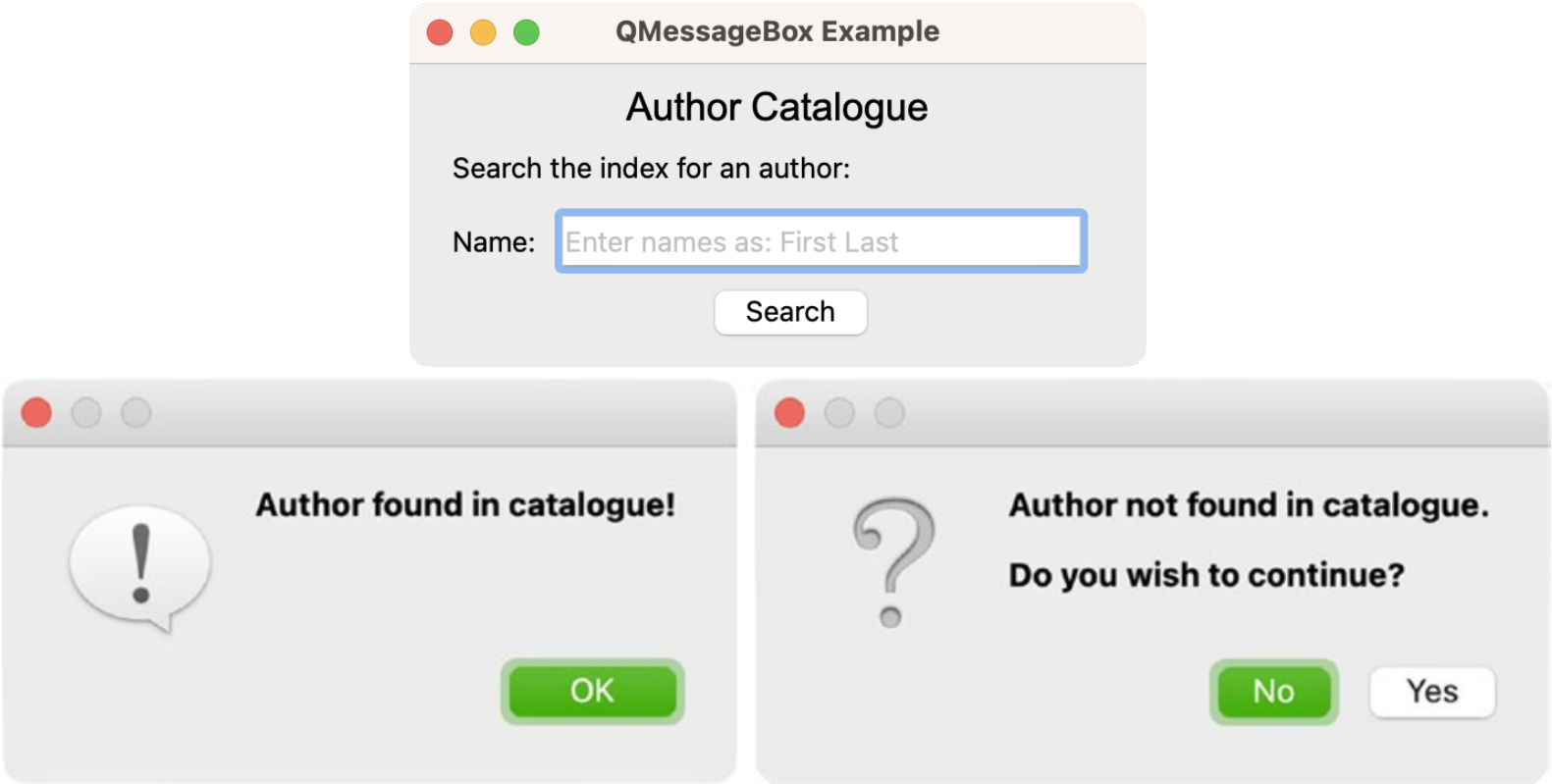
The QMessageBox class can be used to not only alert the user to a situation but also to decide how to handle the matter. When a user closes an application or saves their work, or an error occurs, they will typically see a dialog box pop up and display some sort of key information

QMessageBox Icons	Types	Details
	Question	ask the user a question
	information	display information during general operations
	warning	report noncritical errors report critical errors
	Critical	report critical errors

Bảng 1: Four types of static QMessageBox dialogs in PyQt.

The QMessageBox Dialog

Hình 6: Information dialog (left) that lets the user know that their search was successful. Question dialog (right) that asks if the user wants to continue searching if the author wasn't found



The QMessageBox Dialog I

```
1  import sys
2  from PyQt6.QtWidgets import (QApplication, QWidget, QLabel,
3      QMessageBox, QLineEdit, QPushButton)
4  from PyQt6.QtGui import QFont
5
6  class MainWindow(QWidget):
7
8      def __init__(self):
9          super().__init__()
10         self.initializeUI()
11
12     def initializeUI(self):
13         """Set up the application's GUI."""
14         self.setGeometry(100, 100, 340, 140)
15         self.setWindowTitle("QMessageBox Example")
16
17         self.setUpMainWindow()
18         self.show()
19
20     def setUpMainWindow(self):
21         """Create and arrange widgets in the main window."""
22         catalogue_label = QLabel("Author Catalogue", self)
23         catalogue_label.move(100, 10)
24         catalogue_label.setFont(QFont("Arial", 18))
25
26         search_label = QLabel("Search the index for an author:", self)
27         search_label.move(20, 40)
28
```

The QMessageBox Dialog II

```
29          # Create name QLabel and QLineEdit widgets
30          author_label = QLabel("Name:", self)
31          author_label.move(20, 74)
32
33          self.author_edit = QLineEdit(self)
34          self.author_edit.move(70, 70)
35          self.author_edit.resize(240, 24)
36          self.author_edit.setPlaceholderText(
37              "Enter names as: First Last")
38
39          # Create search QPushButton
40          search_button = QPushButton("Search", self)
41          search_button.move(140, 100)
42          search_button.clicked.connect(self.searchAuthors)
43
44      def searchAuthors(self):
45          """Search through catalogue of names.
46          If name is found, display Author Found dialog.
47          Otherwise, display Author Not Found dialog."""
48          file = "files/authors.txt"
49
50          try:
51              with open(file, "r") as f:
52                  authors = [line.rstrip("\n") for line in f]
53
54              # Check for name in authors list
55              if self.author_edit.text() in authors:
```

The QMessageBox Dialog III

```
56                  QMessageBox.information(self, "Author Found",
57                      "Author found in catalogue!", QMessageBox.StandardButton.Ok)
58              else:
59                  answer = QMessageBox.question(self, "Author Not Found",
60                      """<p>Author not found in catalogue.</p>
61                      <p>Do you wish to continue?</p>""",
62                      QMessageBox.StandardButton.Yes | \
63                      QMessageBox.StandardButton.No,
64                      QMessageBox.StandardButton.Yes)
65
66                  if answer == QMessageBox.StandardButton.No:
67                      print("Closing application.")
68                      self.close()
69          except FileNotFoundError as error:
70              QMessageBox.warning(self, "Error",
71                  f"""<p>File not found.</p>
72                  <p>Error: {error}</p>
73                  Closing application.""",
74                  QMessageBox.StandardButton.Ok)
75              self.close()
76
77  if __name__ == '__main__':
78      app = QApplication(sys.argv)
79      window = MainWindow()
80      sys.exit(app.exec())
```

The QMessageBox Dialog IV

Explanation for Using QMessageBox

In this example:

- there are three of the predefined QMessageBox message types: **question, information, and warning**
- a few *QLabel objects, a QLineEdit widget for the user to enter an author's name*, and a *QPushButton object that emits a signal when pressed and searches for the text in a text file.*
- When the user clicks on search_button, the program will try to open the **authors. txt** file and store its contents in the authors list. If the user enters a name in author_ edit that is contained in the authors.txt file, an information dialog appears like left Figure 6

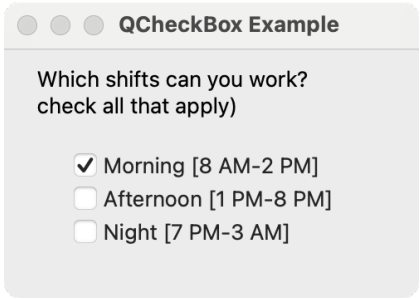
The QMessageBox Dialog V

Note

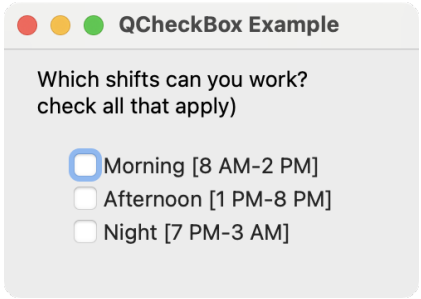
PyQt text widgets are able to display rich text using a subset of the HyperText Markup Language (HTML) and Cascading Style Sheets (CSS).

QCheckBox+ QMessageBox

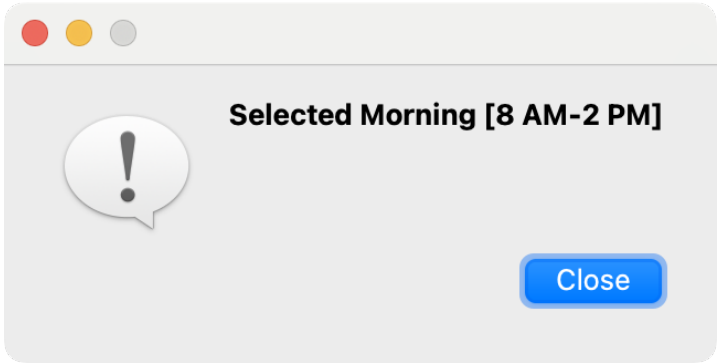
Hãy kết hợp **QCheckBox+ QMessageBox** để thu được GUI có chức năng như hình



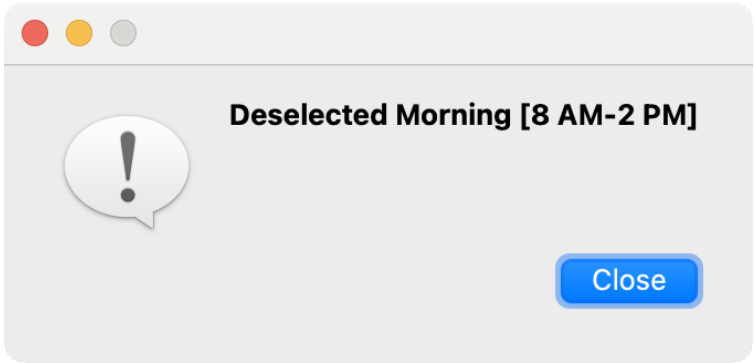
(a)



(b)



(c)

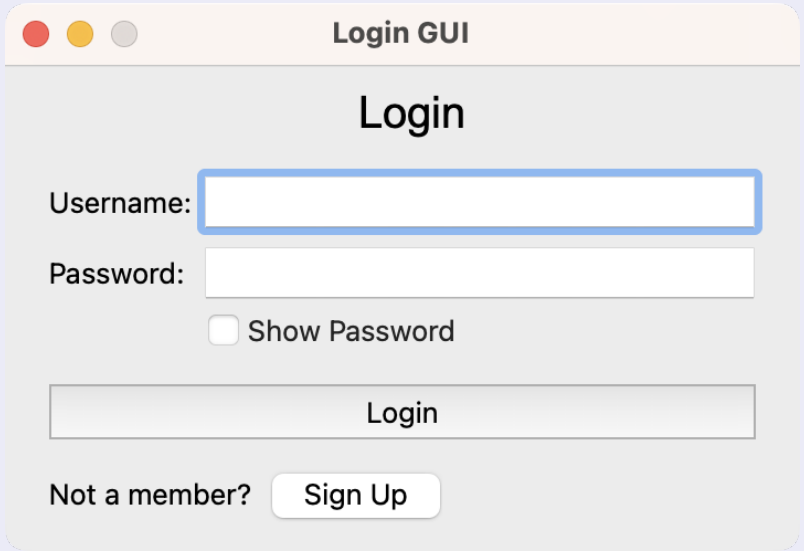


(d)

Project: Login GUI and Registration Dialog

In this example, you will create three different windows. The first window that will appear is the login GUI

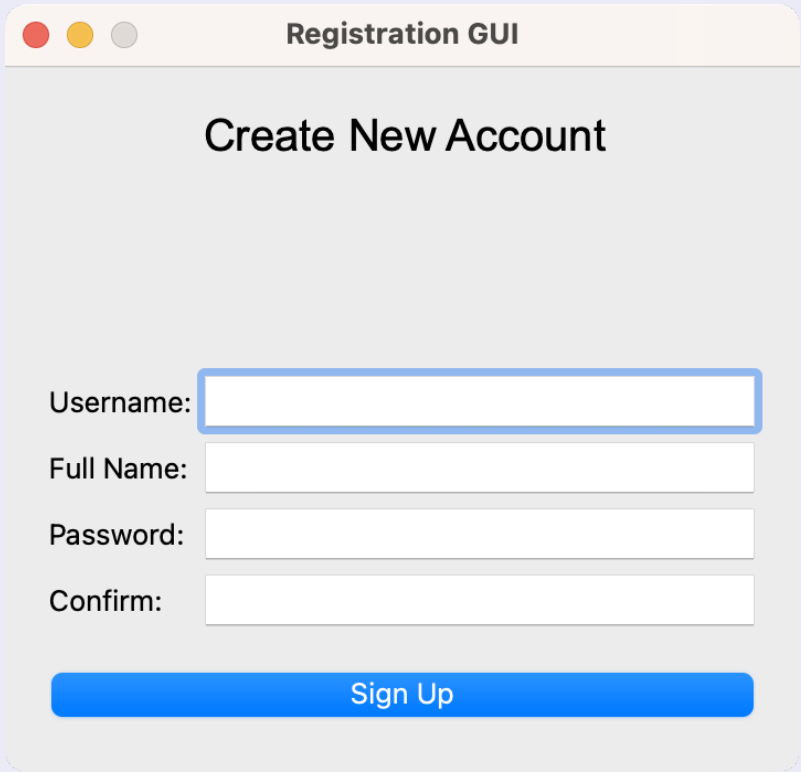
Hình 7: The Login window



Project: Login GUI and Registration Dialog

The second window that will appear is the Registration dialog when a user clicks the Sign Up button in Figure 8.

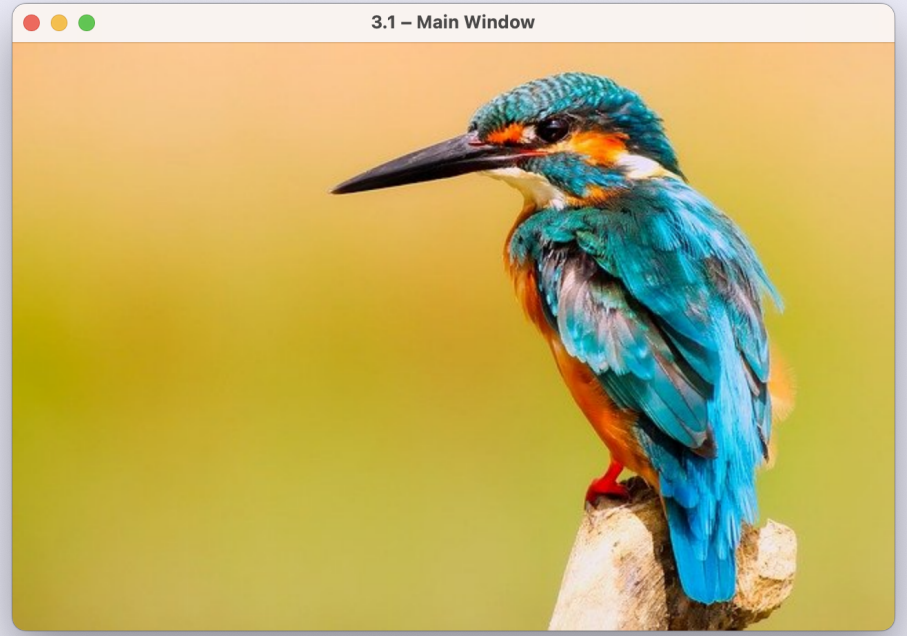
Hình 8: The Registration dialog for creating a new user



Project: Login GUI and Registration Dialog

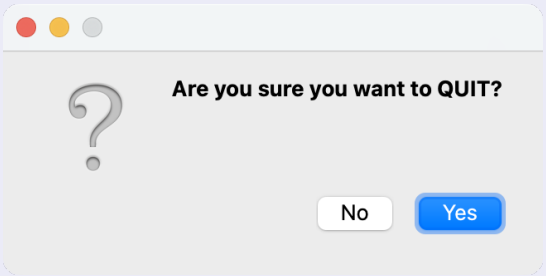
If a user successfully logs in, they will be greeted to the window in Figure 10 that simply displays a QLabel widget with the image of a kingfisher.

Hình 9: The application's main window that appears if a user successfully logs in. The image of the kingfisher is from `https://pixabay.com`

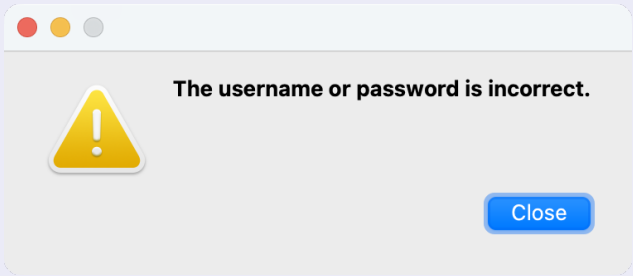


Project: Login GUI and Registration Dialog

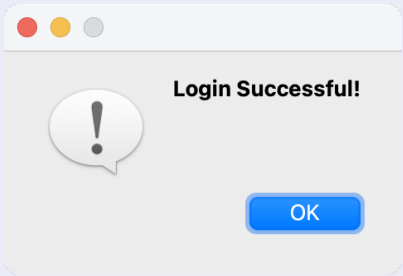
This project also demonstrates **how to open and close other windows and dialogs and begins looking at how to use event handlers.**



(a) QMessageBox that appears before quitting the application



(b) The warning dialog that informs the user of an error



(c) The information dialog that informs the user that their input was correct

Code for Project: **login_gui.py** I

login_gui.py

```
1  import sys
2  from PyQt6.QtWidgets import (QApplication, QWidget, QLabel,
3       QLineEdit, QPushButton, QCheckBox, QMessageBox)
4  from PyQt6.QtGui import QFont, QPixmap
5  from PyQt6.QtCore import Qt
6  from registration import NewUserDialog
7
8  class LoginWindow(QWidget):
9
10     def __init__(self):
11         super().__init__()
12         self.initializeUI()
13
14     def initializeUI(self):
15         """Set up the application's GUI."""
16         self.setFixedSize(360, 220)
17         self.setWindowTitle("Login GUI")
18
19         self.setUpWindow()
20         self.show()
21
22     def setUpWindow(self):
23         """Create and arrange widgets in the main window."""
24         self.login_is_successful = False
25
```

Code for Project: **login_gui.py** II

```
26          login_label = QLabel("Login", self)
27          login_label.setFont(QFont("Arial", 20))
28          login_label.move(160, 10)
29
30          # Create widgets for username and password
31          username_label = QLabel("Username:", self)
32          username_label.move(20, 54)
33
34          self.username_edit = QLineEdit(self)
35          self.username_edit.resize(250, 24)
36          self.username_edit.move(90, 50)
37
38          password_label = QLabel("Password:", self)
39          password_label.move(20, 86)
40
41          self.password_edit = QLineEdit(self)
42          self.password_edit.setEchoMode(QLineEdit.EchoMode.Password)
43          self.password_edit.resize(250, 24)
44          self.password_edit.move(90, 82)
45
46          # Create QCheckBox for displaying password
47          self.show_password_cb = QCheckBox("Show Password", self)
48          self.show_password_cb.move(90, 110)
49          self.show_password_cb.toggled.connect(self.displayPasswordIfChecked)
50
51          # Create QPushButton for signing in
52          login_button = QPushButton("Login", self)
```

Code for Project: **login_gui.py** III

```
53          login_button.resize(320, 34)
54          login_button.move(20, 140)
55          login_button.clicked.connect(self.clickLoginButton)
56
57          # Create sign up QLabel and QPushButton
58          not_member_label = QLabel("Not a member?", self)
59          not_member_label.move(20, 186)
60
61          sign_up_button = QPushButton("Sign Up", self)
62          sign_up_button.move(120, 180)
63          sign_up_button.clicked.connect(self.createNewUser)
64
65      def clickLoginButton(self):
66          """Check if username and password match any existing
67          entries in users.txt.
68          If they exist, display QMessageBox and close program.
69          If they don't, display a warning QMessageBox."""
70          users = {} # Dictionary to store user information
71          file = "files/users.txt"
72
73          try:
74              with open(file, 'r') as f:
75                  for line in f:
76                      user_info = line.split(" ")
77                      username_info = user_info[0]
78                      password_info = user_info[1].strip('\n')
79                      users[username_info] = password_info
```

Code for Project: **login_gui.py** IV

```
80
81              # Collect user and password information
82              username = self.username_edit.text()
83              password = self.password_edit.text()
84
85              if (username, password) in users.items():
86                  QMessageBox.information(self, "Login Successful!",
87                      "Login Successful!", QMessageBox.StandardButton.Ok,
88                      QMessageBox.StandardButton.Ok)
89                  self.login_is_successful = True
90                  self.close() # Close the login window
91                  self.openApplicationWindow()
92              else:
93                  QMessageBox.warning(self, "Error Message",
94                      "The username or password is incorrect.",
95                      QMessageBox.StandardButton.Close,
96                      QMessageBox.StandardButton.Close)
97          except FileNotFoundError as error:
98              QMessageBox.warning(self, "Error",
99                  f"""<p>File not found.</p>
100                 <p>Error: {error}</p>""",
101                 QMessageBox.StandardButton.Ok)
102             # Create file if it doesn't exist
103             f = open(file, "w")
104
105     def displayPasswordIfChecked(self, checked):
106         """If QCheckButton is enabled, view password.
```

Code for Project: login_gui.py V

```
107         Else, mask the password so others can not see it."""
108         if checked:
109             self.password_edit.setEchoMode(QLineEdit.EchoMode.Normal)
110         elif checked == False:
111             self.password_edit.setEchoMode(QLineEdit.EchoMode.Password)
112
113     def createNewUser(self):
114         """Open a dialog for creating a new account."""
115         self.create_new_user_window =NewUserDialog()  # this slot will be linked to
                registraion.py file
116         self.create_new_user_window.show()
117
118     def openApplicationWindow(self):
119         """Open a mock main window after the user logs in."""
120         self.main_window = MainWindow()
121         self.main_window.show()
122
123     def closeEvent(self, event):
124         """Reimplement the closing event to display a QMessageBox
125         before closing."""
126         if self.login_is_successful == True:
127             event.accept()
128         else:
129             answer = QMessageBox.question(self, "Quit Application?",
130                 "Are you sure you want to QUIT?",
```

Code for Project: **login_gui.py** VI

```
131                 QMessageBox.StandardButton.No | \
132                 QMessageBox.StandardButton.Yes,
133                 QMessageBox.StandardButton.Yes)
134             if answer == QMessageBox.StandardButton.Yes:
135                 event.accept()
136             if answer == QMessageBox.StandardButton.No:
137                 event.ignore()
138
139 class MainWindow(QWidget):
140
141     def __init__(self):
142         super().__init__()
143         self.initializeUI()
144
145     def initializeUI(self):
146         """Set up the application's GUI."""
147         self.setMinimumSize(640, 426)
148         self.setWindowTitle("Main Window")
149         self.setUpMainWindow()
150
151     def setUpMainWindow(self):
152         """Create and arrange widgets in the main window."""
153         image = "images/background_kingfisher.jpg"
154
155         try:
156             with open(image):
157                 main_label = QLabel(self)
```

Code for Project: **login_gui.py** VII

```
158                 pixmap = QPixmap(image)
159                 main_label.setPixmap(pixmap)
160                 main_label.move(0, 0)
161         except FileNotFoundError as error:
162             print(f"Image not found.\nError: {error}")
163
164 if __name__ == '__main__':
165     app = QApplication(sys.argv)
166     window = LoginWindow()
167     sys.exit(app.exec())
```

Code for Project: **registration.py** I

registration.py

```
1  import sys
2  from PyQt6.QtWidgets import (QApplication, QDialog, QLabel,
3      QPushButton, QLineEdit, QMessageBox)
4  from PyQt6.QtGui import QFont, QPixmap
5
6  class NewUserDialog(QDialog):
7
8      def __init__(self):
9          super().__init__()
10         self.setModal(True)
11         self.initializeUI()
12
13     def initializeUI(self):
14         """Set up the application's GUI."""
15         self.setFixedSize(360, 320)
16         self.setWindowTitle("Registration GUI")
17         self.setUpWindow()
18
19     def setUpWindow(self):
20         """Create and arrange widgets in the window for
21         collecting new account information."""
22         login_label = QLabel("Create New Account", self)
23         login_label.setFont(QFont("Arial", 20))
24         login_label.move(90, 20)
25
```

Code for Project: **registration.py** II

```
26          # Create QLabel for image
27          user_image = "images/new_user_icon.png"
28
29          try:
30              with open(user_image):
31                  user_label = QLabel(self)
32                  pixmap = QPixmap(user_image)
33                  user_label.setPixmap(pixmap)
34                  user_label.move(150, 60)
35          except FileNotFoundError as error:
36              print(f"Image not found. Error: {error}")
37
38          # Create name QLabel and QLineEdit widgets
39          name_label = QLabel("Username:", self)
40          name_label.move(20, 144)
41
42          self.name_edit = QLineEdit(self)
43          self.name_edit.resize(250, 24)
44          self.name_edit.move(90, 140)
45
46          full_name_label = QLabel("Full Name:", self)
47          full_name_label.move(20, 174)
48
49          full_name_edit = QLineEdit(self)
50          full_name_edit.resize(250, 24)
51          full_name_edit.move(90, 170)
52
```

Code for Project: **registration.py** III

```
53          # Create password QLabel and QLineEdit widgets
54          new_pswd_label = QLabel("Password:", self)
55          new_pswd_label.move(20, 204)
56
57          self.new_pswd_edit = QLineEdit(self)
58          self.new_pswd_edit.setEchoMode(QLineEdit.EchoMode.Password)
59          self.new_pswd_edit.resize(250, 24)
60          self.new_pswd_edit.move(90, 200)
61
62          confirm_label = QLabel("Confirm:", self)
63          confirm_label.move(20, 234)
64
65          self.confirm_edit = QLineEdit(self)
66          self.confirm_edit.setEchoMode(QLineEdit.EchoMode.Password)
67          self.confirm_edit.resize(250, 24)
68          self.confirm_edit.move(90, 230)
69
70          # Create sign up QPushButton
71          sign_up_button = QPushButton("Sign Up", self)
72          sign_up_button.resize(320, 32)
73          sign_up_button.move(20, 270)
74          sign_up_button.clicked.connect(self.confirmSignUp)
75
76      def confirmSignUp(self):
77          """Check if user information is entered and correct.
78          If so, append username and password text to file."""
79          name_text = self.name_edit.text()
```

Code for Project: **registration.py** IV

```
80          pswd_text = self.new_pswd_edit.text()
81          confirm_text = self.confirm_edit.text()
82
83          if name_text == "" or pswd_text == "":
84              # Display QMessageBox if passwords don't match
85              QMessageBox.warning(self, "Error Message",
86                  "Please enter username or password values.",
87                  QMessageBox.StandardButton.Close,
88                  QMessageBox.StandardButton.Close)
89          elif pswd_text != confirm_text:
90              # Display QMessageBox if passwords don't match
91              QMessageBox.warning(self, "Error Message",
92                  "The passwords you entered do not match.",
93                  QMessageBox.StandardButton.Close,
94                  QMessageBox.StandardButton.Close)
95          else:
96              # Return to login window if passwords match
97              with open("files/users.txt", "a+") as f:
98                  f.write("\n" + name_text + " ")
99                  f.write(pswd_text)
100             self.close()
```

Using Layout Managers in PyQt

Layout management is the useful practice of arranging widgets in GUIs

- When organizing widgets, you'll need to consider a number of situations, including a **widget's size** and **position relative to other widgets**, what to do **if the window is resized**, and how to **handle widgets when they are added or removed**.
- Allowing for child and parent widgets to communicate, ensuring that widgets utilize the space in a window more efficiently whenever changes occur.

Simple Example: Using Layout Managers in PyQt

Let's look at a brief example, create an instance of the layout manager for arranging widgets vertically, **QVBoxLayout**:

```
1 #create your two widget objects:
2 label = QLabel("Name")
3 line_edit = QLineEdit()yout() # Create layout manager instance
4 v_box.addWidget(label) # Add widgets to the layout
5 v_box.addWidget(line_edit)
6 parent_widget.setLayout(v_box) # Set the layout for the parent
```

- To add widgets to the layout, use the addWidget() method and pass the widget to be added
- Then apply the layout used in the parent widget by calling the QWidget method setLayout().
- **The parent widget could be a widget**, a window, or even a dialog

Simple Example: Using Layout Managers in PyQt

- you can also add layouts to other layouts to **create a nested layout** (*use the method addLayout() to pass the layout to the parent layout*)

```
1 h_box = QHBoxLayout()
2 v_box.addLayout(h_box)
```

Here, h_box is the child layout and v_box is the parent, and h_box becomes an inner layout, or a child of a parent layout.

Advantages

Absolute positioning can be most useful for setting the position and size values of widgets that are **contained within other widgets**, or perhaps for repositioning a window's location on the desktop.

Drawbacks

- First, resizing the main window will not cause the widgets in it to adjust their size or position.
- Second, the differences between operating systems, such as fonts and font sizes, could drastically change the look and layout of the widgets in an interface on different platforms.
- Finally, using absolute positioning is expensive in terms of a developer's time, as they will need to calculate the exact size and position of each widget.

Horizontal and Vertical Layouts with Box Layouts

QBoxLayout uses the space it is provided from a parent to assign each widget a box with a certain amount of space. PyQt also has two separate convenience classes that derive from QBoxLayout and deliver functionality based on the desired orientation of the widgets:

- **QHBoxLayout** - Arranges widgets horizontally from left to right or vice versa
- **QVBoxLayout** - Arranges widgets vertically from top to bottom or vice versa

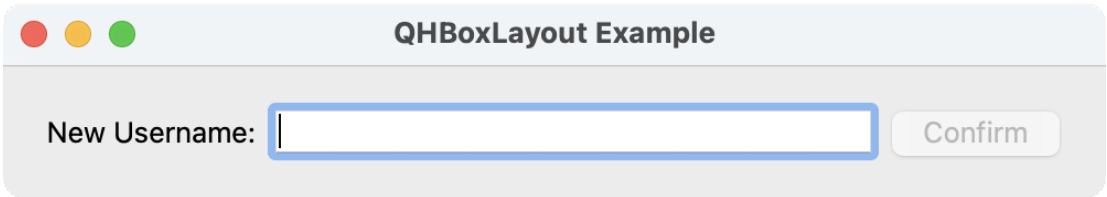
Optional parameters can be passed to the addWidget() method for the box layouts as seen in the following line:

```
addWidget(widget, stretch, alignment)
```

Horizontal and Vertical Layouts with Box Layouts: **QHBoxLayout**

For this project, the GUI in Figure 11 consists of three basic widgets: QLabel, QLineEdit, and QPushButton. These widgets are arranged horizontally using QHBoxLayout.

Hình 10: A simple QHBoxLayout example with each widget arranged horizontally in the GUI



Horizontal and Vertical Layouts with Box Layouts: **QHBoxLayout** I

```
1  import sys
2  from PyQt6.QtWidgets import (QApplication, QWidget, QLabel,
3      QLineEdit, QPushButton, QHBoxLayout)
4
5  class MainWindow(QWidget):
6
7      def __init__(self):
8          super().__init__()
9          self.initializeUI()
10
11     def initializeUI(self):
12         """Set up the application's GUI."""
13         self.setMinimumWidth(500)
14         self.setFixedHeight(60)
15         self.setWindowTitle('QHBoxLayout Example')
16
17         self.setUpMainWindow()
18         self.show()
19
20     def setUpMainWindow(self):
21         """Create and arrange widgets in the main window."""
22         name_label = QLabel("New Username:")
23
24         name_edit = QLineEdit()
25         name_edit.setClearButtonEnabled(True)
```

Horizontal and Vertical Layouts with Box Layouts: **QHBoxLayout** II

```
26          name_edit.textEdited.connect(self.checkUserInput)
27
28          self.confirm_button = QPushButton("Confirm")
29          self.confirm_button.setEnabled(False)
30          self.confirm_button.clicked.connect(self.close)
31
32          main_h_box = QHBoxLayout()
33          main_h_box.addWidget(name_label)
34          main_h_box.addWidget(name_edit)
35          main_h_box.addWidget(self.confirm_button)
36
37          # Set the layout for the main window
38          self.setLayout(main_h_box)
39
40      def checkUserInput(self, text):
41          """Check the length and content of name_edit."""
42          if len(text) > 0 \
43              and all(t.isalpha() or t.isdigit() for t in text):
44              self.confirm_button.setEnabled(True)
45          else: self.confirm_button.setEnabled(False)
46
47  if __name__ == '__main__':
48      app = QApplication(sys.argv)
49      window = MainWindow()
```

Horizontal and Vertical Layouts with Box Layouts: **QHBoxLayout** III

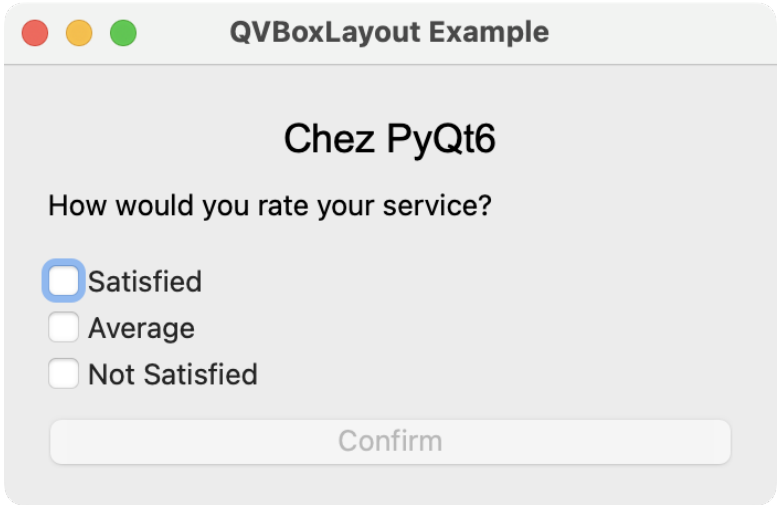
```
50      sys.exit(app.exec())
```

- the window's minimum width is set with setMinimumWidth() so that it can be resized horizontally, but the height is fixed using setFixedHeight().
- When the user edits text in name_edit, the textEdited signal will trigger checkUserInput(). This signal gives us access to the current text. If the length of text is at least 1 and only contains letters or numbers, then accept_button is enabled.

Horizontal and Vertical Layouts with Box Layouts: **QVBoxLayout**

In the following program, we will take a look at how to use the QVBoxLayout class to create a simple window that displays a question to the user and allows them to select an answer.

Hình 11: A simple QVBoxLayout example with each widget arranged vertically in the GUI



Horizontal and Vertical Layouts with Box Layouts: **QVBoxLayout** I

```
1  import sys
2  from PyQt6.QtWidgets import (QApplication, QWidget, QLabel,
3      QCheckBox, QPushButton, QButtonGroup, QVBoxLayout)
4  from PyQt6.QtCore import Qt
5  from PyQt6.QtGui import QFont
6
7  class MainWindow(QWidget):
8
9      def __init__(self):
10         super().__init__()
11         self.initializeUI()
12
13     def initializeUI(self):
14         """Set up the application's GUI."""
15         self.setMinimumSize(350, 200)
16         self.setWindowTitle("QVBoxLayout Example")
17
18         self.setUpMainWindow()
19         self.show()
20
21     def setUpMainWindow(self):
22         """Create and arrange widgets in the main window."""
23         header_label = QLabel("Chez PyQt6")
24         header_label.setFont(QFont("Arial", 18))
25         header_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

Horizontal and Vertical Layouts with Box Layouts: **QVBoxLayout** II

```
26          question_label = QLabel(
27              "How would you rate your service?")
28          question_label.setAlignment(Qt.AlignmentFlag.AlignTop)
29
30          ratings = ["Satisfied", "Average", "Not Satisfied"]
31          ratings_group = QButtonGroup(self)
32          ratings_group.buttonClicked.connect(self.checkboxClicked)
33
34          self.confirm_button = QPushButton("Confirm")
35          self.confirm_button.setEnabled(False)
36          self.confirm_button.clicked.connect(self.close)
37
38          # Organize the widgets into a layout
39          main_v_box = QVBoxLayout()
40          main_v_box.addWidget(header_label)
41          main_v_box.addWidget(question_label)
42
43          # Create the QCheckBox objects, then add
44          # them to the QButtonGroup and the main layout
45          for cb in range(len(ratings)):
46              rating_cb = QCheckBox(ratings[cb])
47              ratings_group.addButton(rating_cb)
48              main_v_box.addWidget(rating_cb)
49
```

Horizontal and Vertical Layouts with Box Layouts: **QVBoxLayout** III

```
50          main_v_box.addWidget(self.confirm_button)
51
52          # Set the layout for the main window
53          self.setLayout(main_v_box)
54
55      def checkboxClicked(self, button):
56          """Check if a QCheckBox in the QButtonGroup has
57          been clicked."""
58          print(button.text())
59          self.confirm_button.setEnabled(True)
60
61  if __name__ == '__main__':
62      app = QApplication(sys.argv)
63      window = MainWindow()
64      sys.exit(app.exec())
```

Horizontal and Vertical Layouts with Box Layouts: **QVBoxLayout** IV

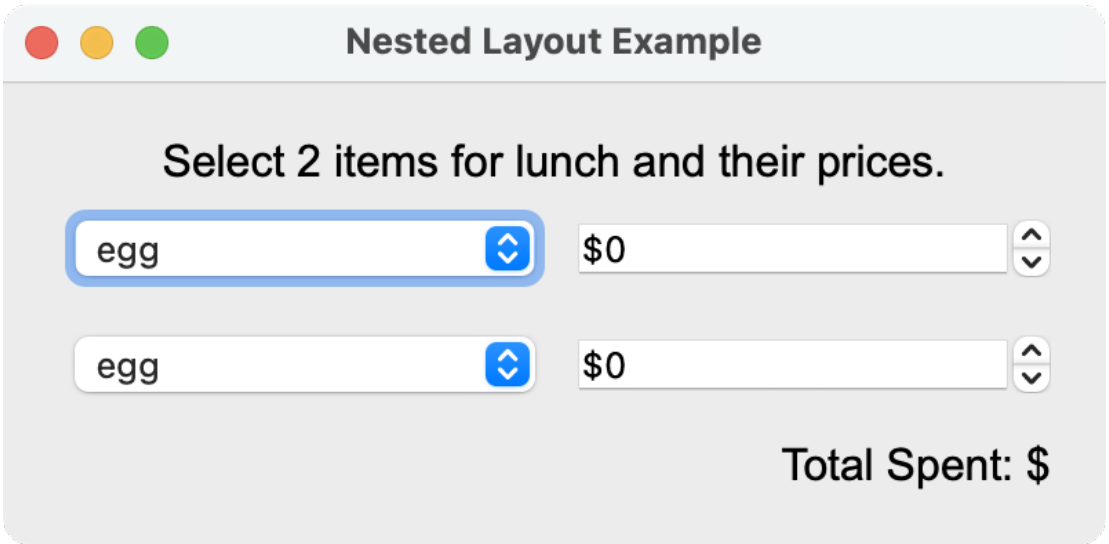
The QCheckBox widgets are also arranged in the window using a layout manager, but we'll need to first discuss **how to manage groups of related buttons.**

You may often have a few checkboxes or buttons that need to be grouped together to make it easier to manage them. PyQt has the **QButtonGroup** class to help manage associated buttons. QButtonGroup is not actually a widget, but an abstract container around the buttons added to it.

Creating Nested Layouts

Handling this matter isn't too difficult with PyQt as you can arrange layouts inside of other layouts to solve intricate arrangement issues.

Hình 12: Widgets arranged using a combination of layout managers: some widgets are arranged vertically, while others are arranged horizontally



Creating Nested Layouts: Example I

In figure 13, the widgets arranged side by side are placed in inner layouts, and those layouts are then added to the parent layout using the layout manager method **addLayout()**.

```
1  import sys
2  from PyQt6.QtWidgets import (QApplication, QWidget, QLabel,
3      QComboBox, QSpinBox, QHBoxLayout, QVBoxLayout)
4  from PyQt6.QtCore import Qt
5  from PyQt6.QtGui import QFont
6
7  class MainWindow(QWidget):
8
9      def __init__(self):
10         super().__init__()
11         self.initializeUI()
12
13     def initializeUI(self):
14         """Set up the application's GUI."""
15         self.setMinimumSize(400, 160)
16         self.setWindowTitle('Nested Layout Example')
17
18         self.setUpMainWindow()
19         self.show()
```

Creating Nested Layouts: Example II

```
20
21      def setUpMainWindow(self):
22          """Create and arrange widgets in the main window."""
23          info_label = QLabel(
24              "Select 2 items for lunch and their prices.")
25          info_label.setFont(QFont("Arial", 16))
26          info_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
27
28          # Create a list of food items and two separate
29          # QComboBox widgets to display all of the items
30          food_list = ["egg", "turkey sandwich", "ham sandwich",
31              "cheese", "hummus", "yogurt", "apple", "banana",
32              "orange", "waffle", "carrots", "bread", "pasta",
33              "crackers", "pretzels", "coffee", "soda", "water"]
34
35          food_combo1 = QComboBox()
36          food_combo1.addItems(food_list)
37          food_combo2 = QComboBox()
38          food_combo2.addItems(food_list)
39
40          # Create two QSpinBox widgets to display prices
41          self.price_sb1 = QSpinBox()
42          self.price_sb1.setRange(0, 100)
43          self.price_sb1.setPrefix("$")
44          self.price_sb1.valueChanged.connect(self.calculateTotal)
45
46          self.price_sb2 = QSpinBox()
```

Creating Nested Layouts: Example III

```
47          self.price_sb2.setRange(0, 100)
48          self.price_sb2.setPrefix("$")
49          self.price_sb2.valueChanged.connect(self.calculateTotal)
50
51          # Create two horizontal layouts for the QComboBox
52          # and QSpinBox widgets
53          item1_h_box = QHBoxLayout()
54          item1_h_box.addWidget(food_combo1)
55          item1_h_box.addWidget(self.price_sb1)
56
57          item2_h_box = QHBoxLayout()
58          item2_h_box.addWidget(food_combo2)
59          item2_h_box.addWidget(self.price_sb2)
60
61          self.totals_label = QLabel("Total Spent: $")
62          self.totals_label.setFont(QFont("Arial", 16))
63          self.totals_label.setAlignment(Qt.AlignmentFlag.AlignRight)
64
65          # Organize widgets and layouts in the main window
66          main_v_box = QVBoxLayout()
67          main_v_box.addWidget(info_label)
68          main_v_box.addLayout(item1_h_box)
69          main_v_box.addLayout(item2_h_box)
70          main_v_box.addWidget(self.totals_label)
71
72          # Set the layout for the main window
73          self.setLayout(main_v_box)
```

Creating Nested Layouts: Example IV

```
74
75      def calculateTotal(self, value):
76          """Calculate the total price and update
77          totals_label."""
78          total = self.price_sb1.value() + \
79              self.price_sb2.value()
80          self.totals_label.setText(f"Total Spent: ${total}")
81
82  if __name__ == '__main__':
83      app = QApplication(sys.argv)
84      window = MainWindow()
85      sys.exit(app.exec())
```

Creating Nested Layouts: Example V

The QSpinBox and QComboBox Widgets

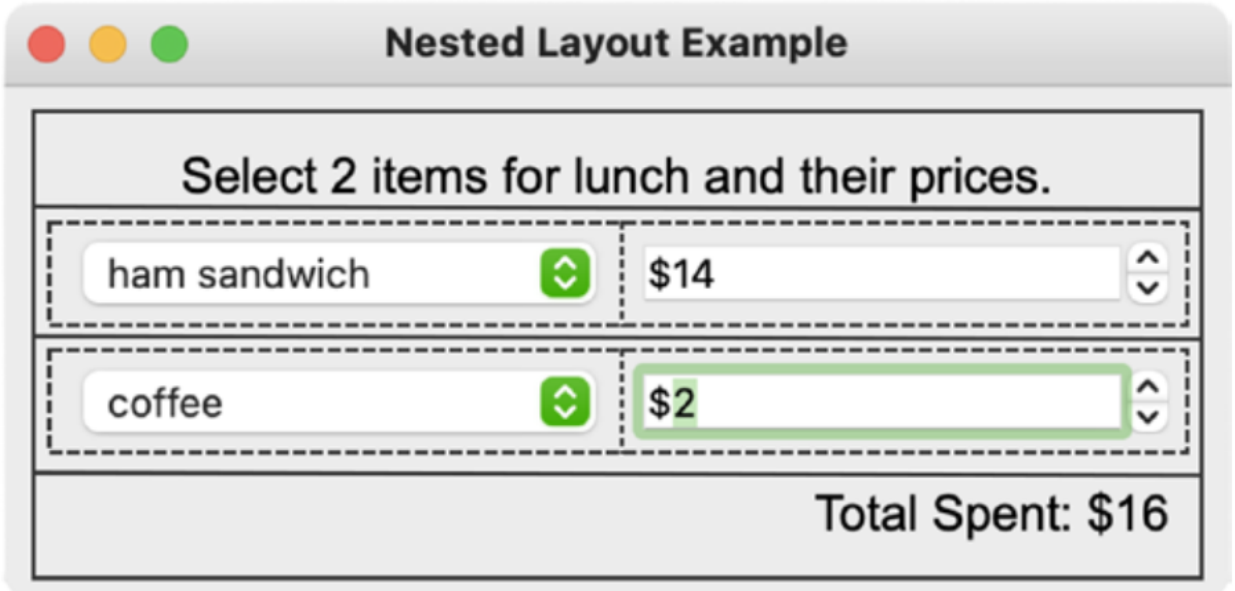
Sometimes, you may want a user to only be allowed to select from a list of predetermined values or numerical ranges.

- **QSpinBox** allows the user to *select integer values* by either typing a value into the widget or by clicking on up and down arrows. **QDoubleSpinBox** is used for selecting floating-point numbers. **QDateTimeEdit** or one of its variations is useful for selecting date and time values.
- The **QComboBox** widget displays a drop-down list of options for the user to select when a user clicks on the widget's arrow button.

We create two separate combo boxes, food_combo1 and food_combo2, two separate spin boxes are created: price_sb1 and price_sb2 and using the **addItems()** method

Explanation for Nested Layouts

Hình 13: Visualization of the nested layout



We change the values in the spin boxes, they both send a signal that is connected to the **calculateTotal()** method. This will dynamically update the value for **totals_label**

Grids: QGridLayout

The QGridLayout layout manager is used to **arrange widgets in rows and columns similar to a spreadsheet or matrix**. Adding space between widgets, creating a border, or stretching widgets across multiple rows or columns is also possible.

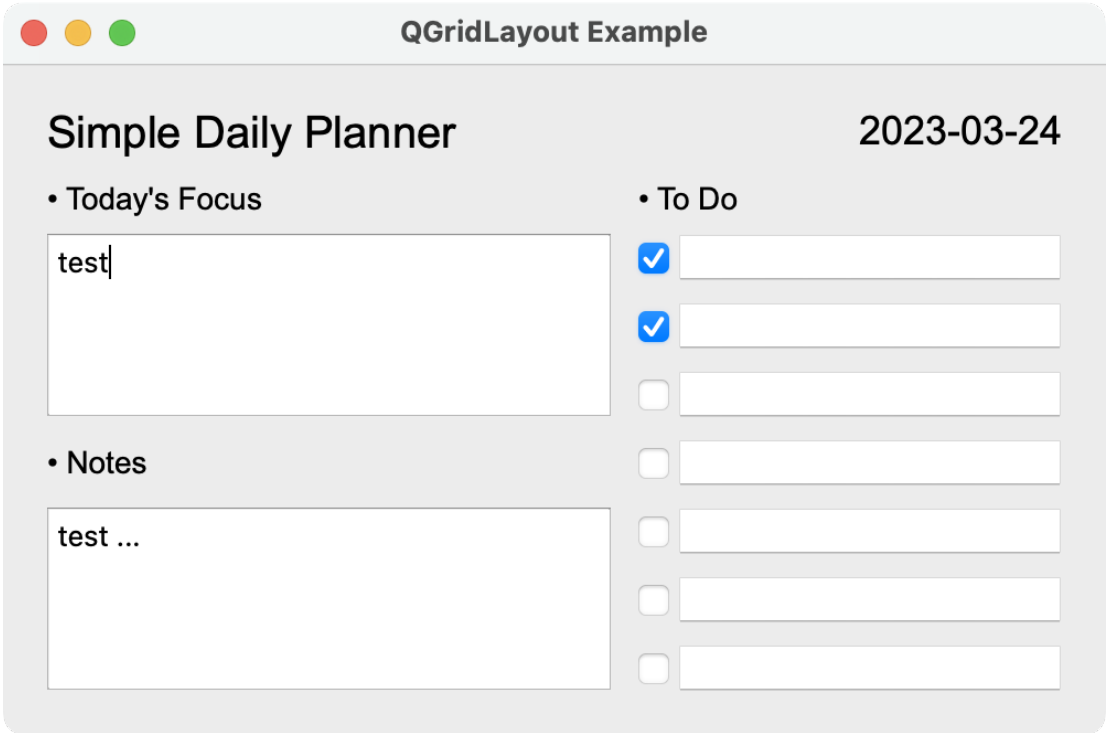
- The **index values in the grid start at (0,0)**, which is the **top left-most cell**.
- The **addWidget()** method for QGridLayout has two forms

```
addWidget(widget, row, column, alignment)
```

QGridLayout Example

The application is composed of three parts: an area to write today's most important task, a place to jot down notes, and a section to write daily tasks.

Hình 14: A simple daily planner that uses QGridLayout to organize widgets



QGridLayout I

```
1  import sys, json
2  from PyQt6.QtWidgets import (QApplication, QWidget, QLabel,
3      QLineEdit, QCheckBox, QTextEdit, QGridLayout)
4  from PyQt6.QtCore import Qt, QDate
5  from PyQt6.QtGui import QFont
6
7  class MainWindow(QWidget):
8
9      def __init__(self):
10         super().__init__()
11         self.initializeUI()
12
13     def initializeUI(self):
14         """Set up the application's GUI."""
15         self.setMinimumSize(500, 300)
16         self.setWindowTitle("QGridLayout Example")
17
18         self.setUpMainWindow()
19         self.loadWidgetValuesFromFile()
20         self.show()
21
22     def setUpMainWindow(self):
23         """Create and arrange widgets in the main window."""
24         name_label = QLabel("Simple Daily Planner")
25         name_label.setFont(QFont("Arial", 20))
26         name_label.setAlignment(Qt.AlignmentFlag.AlignLeft)
27
28         # Create widgets for the left side of the window
```

QGridLayout II

```
29          today_label = QLabel("Today's Focus")
30          today_label.setFont(QFont("Arial", 14))
31          self.today_tedit = QTextEdit()
32
33          notes_label = QLabel("Notes")
34          notes_label.setFont(QFont("Arial", 14))
35          self.notes_tedit = QTextEdit()
36
37          # Organize the left side widgets into a column 0
38          # of the QGridLayout
39          self.main_grid = QGridLayout()
40          self.main_grid.addWidget(name_label, 0, 0)
41          self.main_grid.addWidget(today_label, 1, 0)
42          self.main_grid.addWidget(self.today_tedit, 2, 0, 3, 1)
43          self.main_grid.addWidget(notes_label, 5, 0)
44          self.main_grid.addWidget(self.notes_tedit, 6, 0, 3, 1)
45
46          # Create widgets for the right side of the window
47          today = QDate.currentDate().toString(Qt.DateFormat.ISODate)
48          date_label = QLabel(today)
49          date_label.setFont(QFont("Arial", 18))
50          date_label.setAlignment(Qt.AlignmentFlag.AlignRight)
51
52          todo_label = QLabel("To Do")
53          todo_label.setFont(QFont("Arial", 14))
54
55          # Organize the right side widgets into columns 1 and 2
```

QGridLayout III

```
56          # of the QGridLayout
57          self.main_grid.addWidget(date_label, 0, 2)
58          self.main_grid.addWidget(todo_label, 1, 1, 1, 2)
59
60          # Create 7 rows, from indexes 2-8
61          for row in range(2, 9):
62              item_cb = QCheckBox()
63              item_edit = QLineEdit()
64              self.main_grid.addWidget(item_cb, row, 1)
65              self.main_grid.addWidget(item_edit, row, 2)
66
67          # Set the layout for the main window
68          self.setLayout(self.main_grid)
69
70      def saveWidgetValues(self):
71          """Collect and save the values for the different widgets."""
72          details = {"focus": self.today_tedit.toPlainText(),
73                     "notes": self.notes_tedit.toPlainText()}
74          remaining_todo = []
75
76          # Check the values of the QCheckBox widgets
77          for row in range(2, 9):
78              # Retrieve the QLayoutItem object
79              item = self.main_grid.itemAtPosition(row, 1)
80              # Retrieve the widget (QCheckBox)
81              widget = item.widget()
82              if widget.isChecked() == False:
```

QGridLayout IV

```
83                  # Retrieve the QLayoutItem object
84                  item = self.main_grid.itemAtPosition(row, 2)
85                  # Retrieve the widget (QLineEdit)
86                  widget = item.widget()
87                  text = widget.text()
88                  if text != "":
89                      remaining_todo.append(text)
90              # Save text from QLineEdit widgets
91              details["todo"] = remaining_todo
92
93          with open("details.txt", "w") as f:
94              f.write(json.dumps(details))
95
96      def loadWidgetValuesFromFile(self):
97          """Retrieve the user's previous values from the last session."""
98          # Check if file exists first
99          try:
100             with open("details.txt", "r") as f:
101                 details = json.load(f)
102                 # Retrieve and set values for the widgets
103                 self.today_tedit.setText(details["focus"])
104                 self.notes_tedit.setText(details["notes"])
105
106                 # Set the text for QLineEdit widgets
107                 for row in range(len(details["todo"])):
108                     # Retrieve the QLayoutItem object
109                     item = self.main_grid.itemAtPosition(row + 2, 2)
```

QGridLayout V

```
110                     # Retrieve the widget (QLineEdit)
111                     widget = item.widget()
112                     widget.setText(details["todo"][row])
113             except FileNotFoundError as error:
114                 # Create the file since it doesn't exist
115                 f = open("details.txt", "w")
116
117     def closeEvent(self, event):
118         """Save widget values when closing the window."""
119         self.saveWidgetValues()
120
121 if __name__ == '__main__':
122     app = QApplication(sys.argv)
123     window = MainWindow()
124     sys.exit(app.exec())
```

QTextEdit class is well suited for modifying either plain or rich text and includes built-in editing features, such as copy, paste, and cut.

QGridLayout

Adding Widgets and Spanning Rows and Columns in QGridLayout

When you add a new object to the layout, you must specify the row and column values as parameters of the **addWidget()** or **addLayout()** method.

- The **name_label** object is added to the main_grid layout at the position where **row equals 0 and column equals 0**

QGridLayout

Extending across multiple rows, columns, or both

If you have a widget in a row or a column that needs to take up more space in the vertical or horizontal direction? Technique to have widgets utilize space in QGridLayout is to **specify the number of rows and columns that you want a widget to span**.

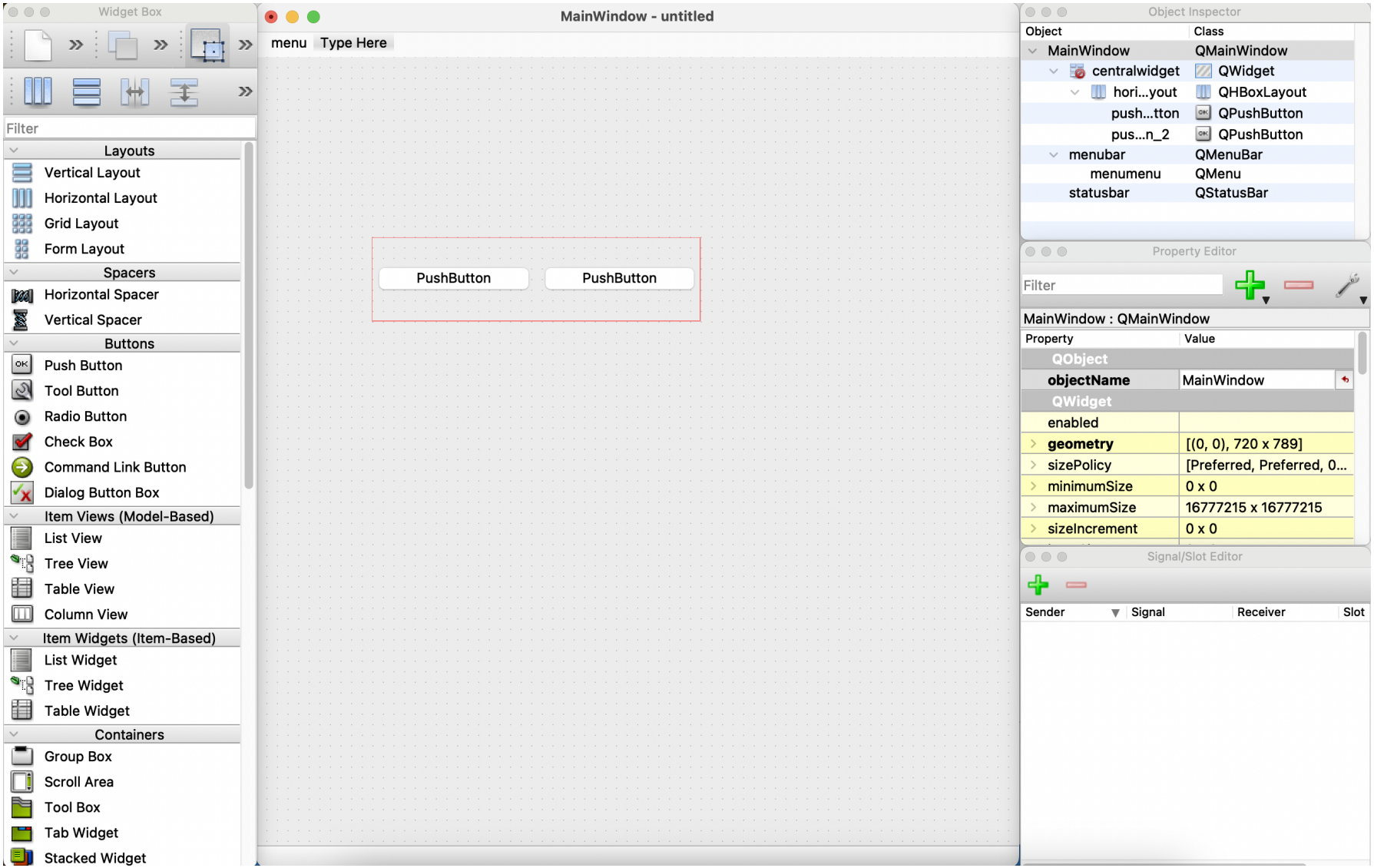
```
self.main_grid.addWidget(self.today_tedit, 2, 0, 3, 1)
```

The extra two parameters at the end, *3 and 1*, tell the layout manager that the widget will *span three rows and one column*.

saveWidgetValues(): When a user closes the daily planner, a record is kept of the previous day's notes and unfinished tasks.

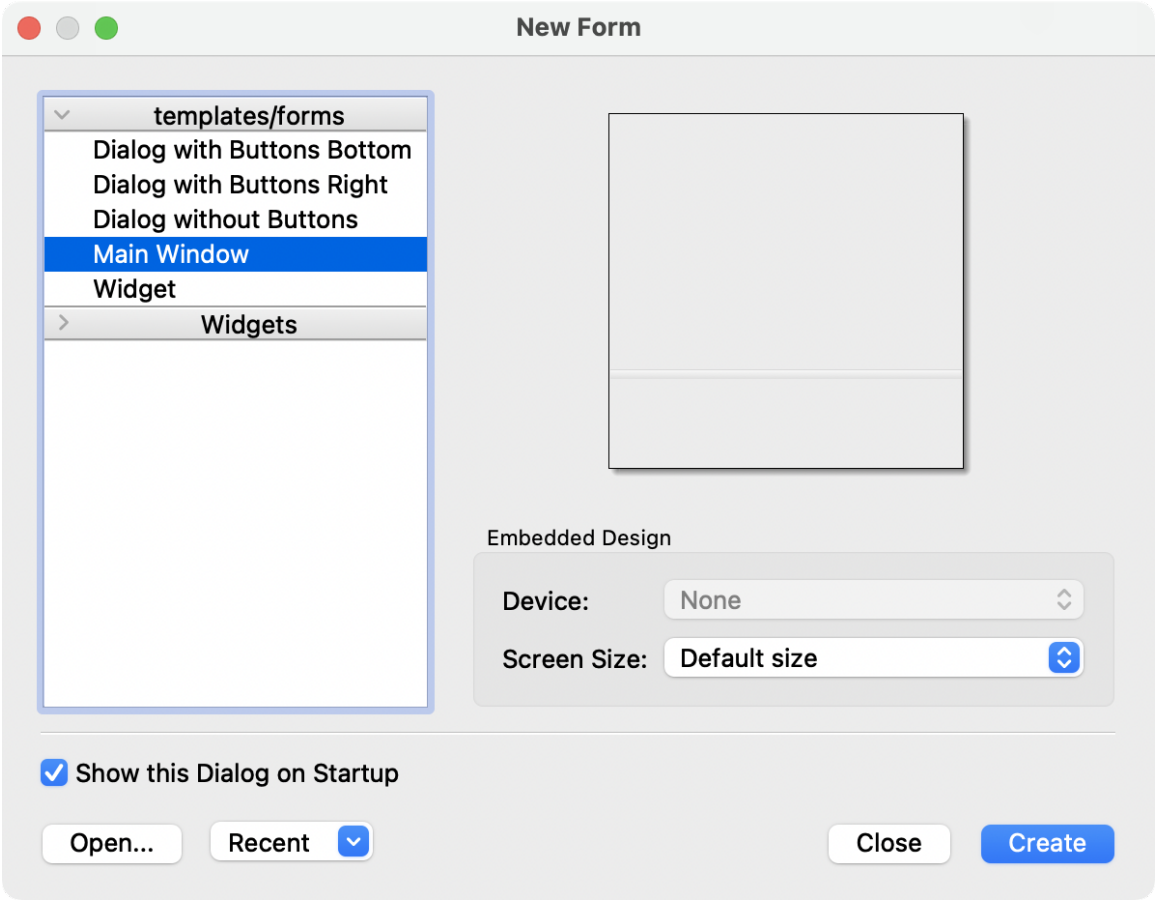
Qt Designer

Hình 15: The Qt Designer interface



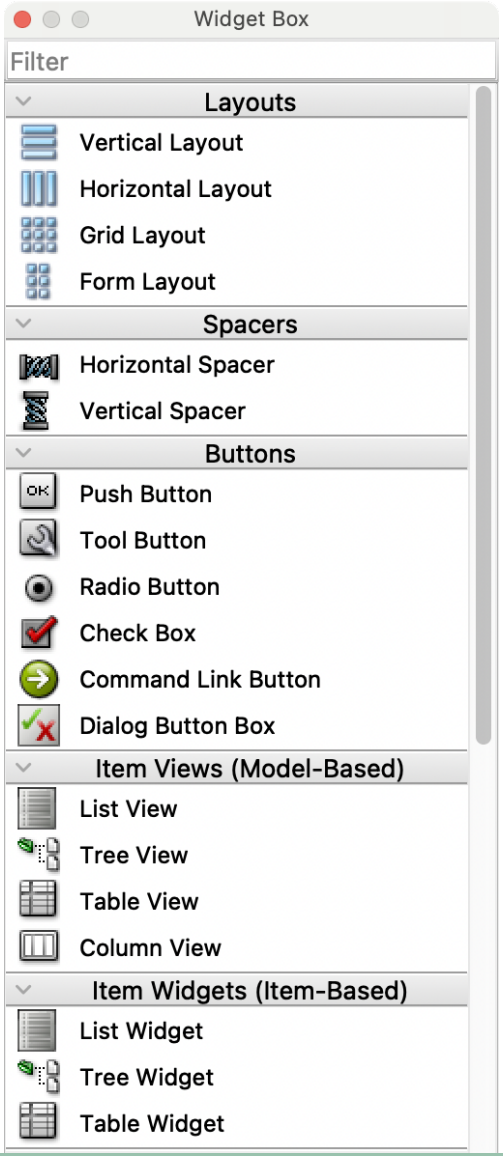
Qt Designer

Hình 16: The New Form dialog box for selecting what type of form to build



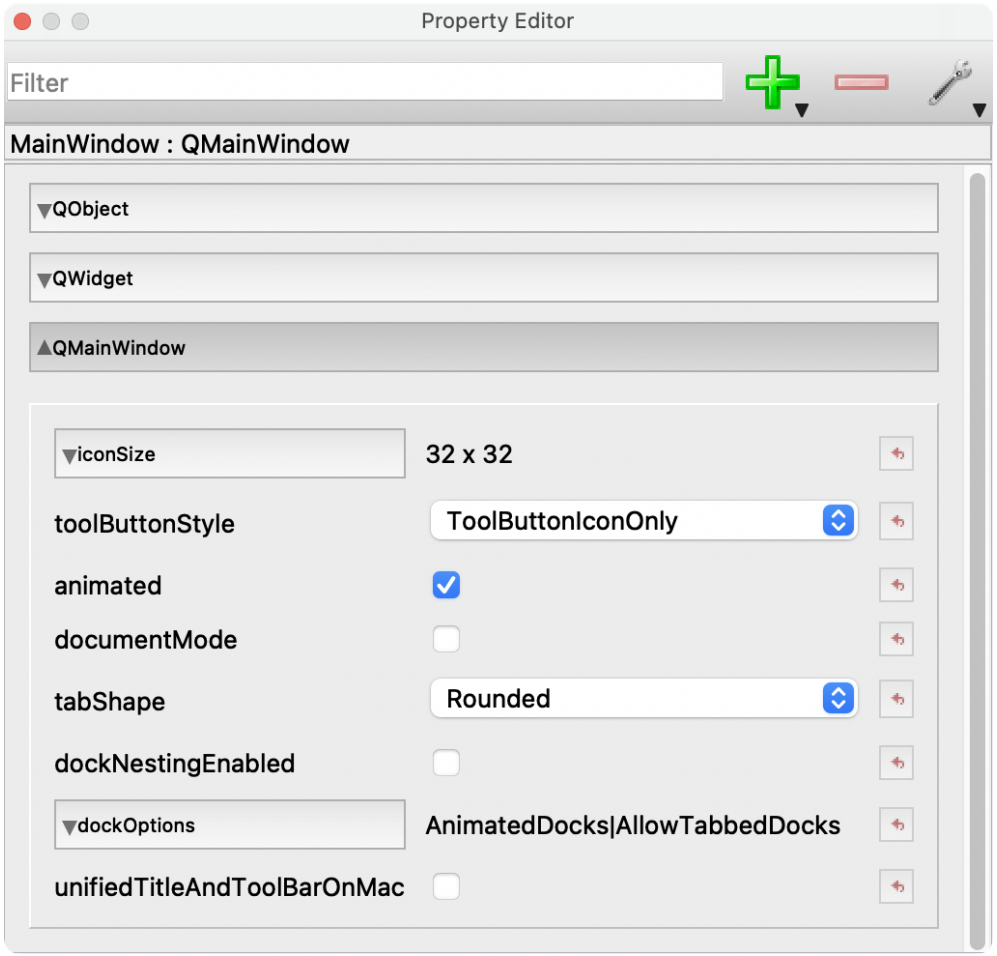
Qt Designer

Hình 17: The Widget Box dock widget for selecting layouts and widgets



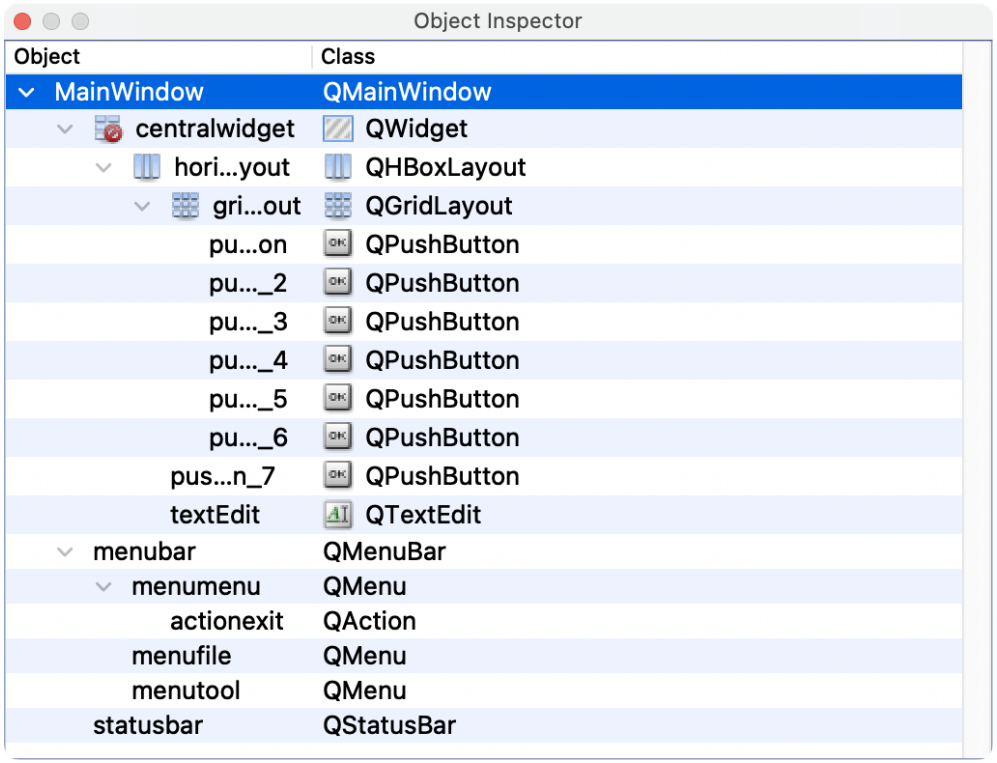
Qt Designer

Hình 18: The Property Editor dock widget for setting the attributes of widgets



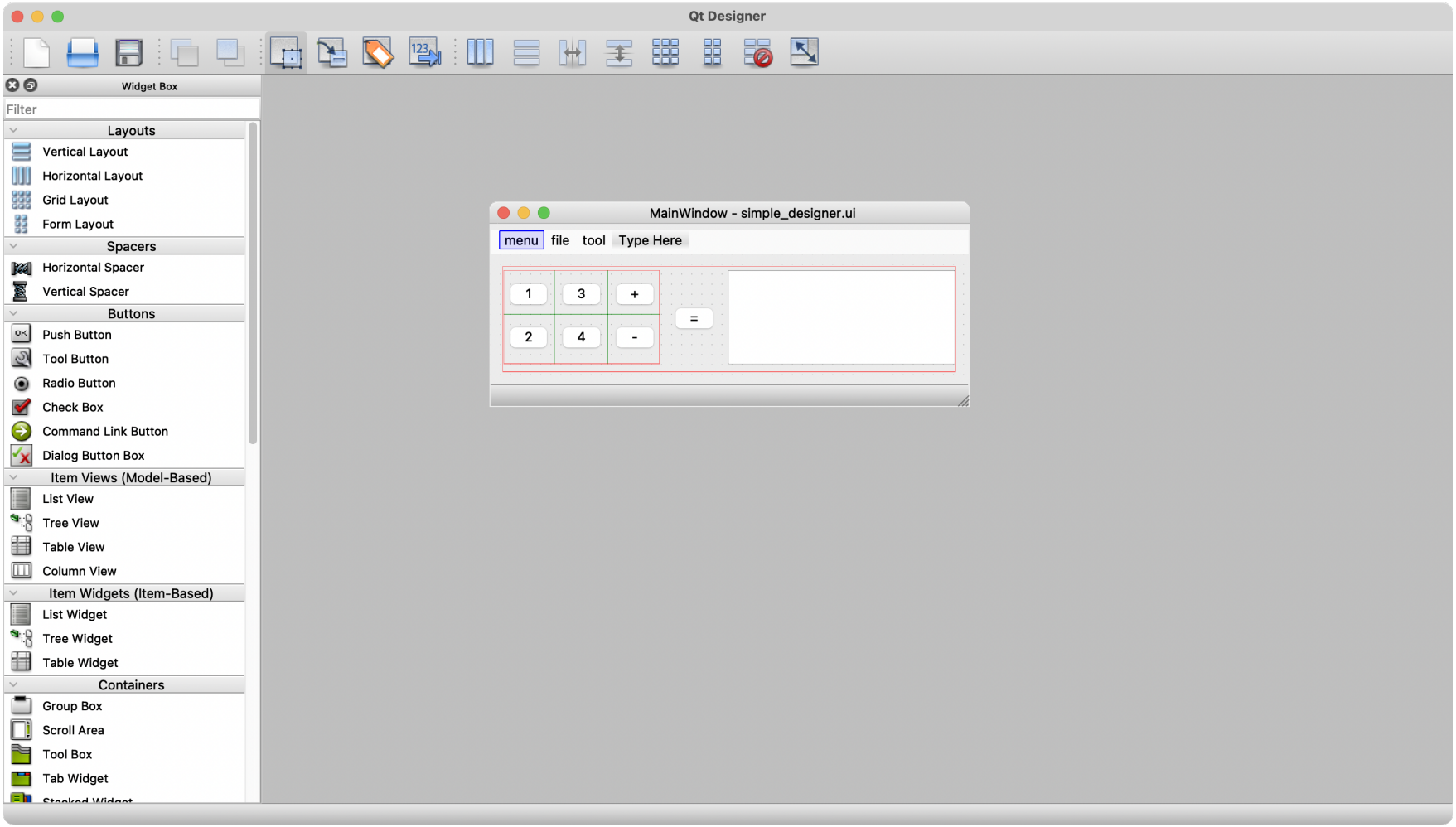
Qt Designer

Hình 19: The Object Inspector displays the widget, layout, and menu objects



Simple Example: Qt Designer

Hình 20: Simple example using Qt Designer



Qt Designer: Create and edit Python code

Example: Generating Code Using pyuic6

After creating UI and saving UI with extension ***.ui**, *the pyuic compiler is used to convert the UI file to readable and editable Python code.*

To generate the simple_designer.py file, navigate to where you saved simple_designer.ui and run the following line:

```
pyuic6 -o simple_designer.py -x simple_designer.ui
```

Qt Designer: Create and edit Python code I

```
# Form implementation generated from reading ui file 'simple_designer.ui'
#
# Created by: PyQt6 UI code generator 6.4.2
#
# WARNING: Any manual changes made to this file will be lost when pyuic6 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt6 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(475, 175)
        self.centralwidget = QtWidgets.QWidget(parent=MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout.setObjectName("verticalLayout")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.gridLayout = QtWidgets.QGridLayout()
        self.gridLayout.setObjectName("gridLayout")
        self.pushButton_3 = QtWidgets.QPushButton(parent=self.centralwidget)
        self.pushButton_3.setObjectName("pushButton_3")
        self.gridLayout.addWidget(self.pushButton_3, 0, 1, 1, 1)
        self.pushButton_4 = QtWidgets.QPushButton(parent=self.centralwidget)
        self.pushButton_4.setObjectName("pushButton_4")
```

Qt Designer: Create and edit Python code II

```
29          self.gridLayout.addWidget(self.pushButton_4, 1, 1, 1, 1)
30          self.pushButton_2 = QtWidgets.QPushButton(parent=self.centralwidget)
31          self.pushButton_2.setObjectName("pushButton_2")
32          self.gridLayout.addWidget(self.pushButton_2, 1, 0, 1, 1)
33          self.pushButton = QtWidgets.QPushButton(parent=self.centralwidget)
34          self.pushButton.setObjectName("pushButton")
35          self.gridLayout.addWidget(self.pushButton, 0, 0, 1, 1)
36          self.pushButton_5 = QtWidgets.QPushButton(parent=self.centralwidget)
37          self.pushButton_5.setObjectName("pushButton_5")
38          self.gridLayout.addWidget(self.pushButton_5, 0, 2, 1, 1)
39          self.pushButton_6 = QtWidgets.QPushButton(parent=self.centralwidget)
40          self.pushButton_6.setObjectName("pushButton_6")
41          self.gridLayout.addWidget(self.pushButton_6, 1, 2, 1, 1)
42          self.horizontalLayout.addLayout(self.gridLayout)
43          self.pushButton_7 = QtWidgets.QPushButton(parent=self.centralwidget)
44          self.pushButton_7.setObjectName("pushButton_7")
45          self.horizontalLayout.addWidget(self.pushButton_7)
46          self.textEdit = QtWidgets.QTextEdit(parent=self.centralwidget)
47          self.textEdit.setObjectName("textEdit")
48          self.horizontalLayout.addWidget(self.textEdit)
49          self.verticalLayout.addLayout(self.horizontalLayout)
50          MainWindow.setCentralWidget(self.centralwidget)
51          self.menubar = QtWidgets.QMenuBar(parent=MainWindow)
52          self.menubar.setGeometry(QtCore.QRect(0, 0, 475, 24))
53          self.menubar.setObjectName("menubar")
54          self.menumenu = QtWidgets.QMenu(parent=self.menubar)
55          self.menumenu.setObjectName("menumenu")
```

Qt Designer: Create and edit Python code III

```
56          self.menufile = QtWidgets.QMenu(parent=self.menubar)
57          self.menufile.setObjectName("menufile")
58          self.menutool = QtWidgets.QMenu(parent=self.menubar)
59          self.menutool.setObjectName("menutool")
60          MainWindow.setMenuBar(self.menubar)
61          self.statusbar = QtWidgets.QStatusBar(parent=MainWindow)
62          self.statusbar.setObjectName("statusbar")
63          MainWindow.setStatusBar(self.statusbar)
64          self.actionexit = QtGui.QAction(parent=MainWindow)
65          self.actionexit.setObjectName("actionexit")
66          self.menumenu.addAction(self.actionexit)
67          self.menubar.addAction(self.menumenu.menuAction())
68          self.menubar.addAction(self.menufile.menuAction())
69          self.menubar.addAction(self.menutool.menuAction())
70
71          self.retranslateUi(MainWindow)
72          QtCore.QMetaObject.connectSlotsByName(MainWindow)
73
74      def retranslateUi(self, MainWindow):
75          _translate = QtCore.QCoreApplication.translate
76          MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
77          self.pushButton_3.setText(_translate("MainWindow", "3"))
78          self.pushButton_4.setText(_translate("MainWindow", "4"))
79          self.pushButton_2.setText(_translate("MainWindow", "2"))
80          self.pushButton.setText(_translate("MainWindow", "1"))
81          self.pushButton_5.setText(_translate("MainWindow", "+"))
82          self.pushButton_6.setText(_translate("MainWindow", "-"))
```

Qt Designer: Create and edit Python code IV

```
83          self.pushButton_7.setText(_translate("MainWindow", "="))
84          self.menumenu.setTitle(_translate("MainWindow", "menu"))
85          self.menufile.setTitle(_translate("MainWindow", "file"))
86          self.menutool.setTitle(_translate("MainWindow", "tool"))
87          self.actionexit.setText(_translate("MainWindow", "exit"))
88
89
90  if __name__ == "__main__":
91      import sys
92      app = QtWidgets.QApplication(sys.argv)
93      MainWindow = QtWidgets.QMainWindow()
94      ui = Ui_MainWindow()
95      ui.setupUi(MainWindow)
96      MainWindow.show()
97      sys.exit(app.exec())
```

Tài liệu tham khảo


Joshua M Willman
Beginning PyQt: A Hands-on Approach to GUI Programming with PyQt6.
https://link.springer.com/book/10.1007/978-1-4842-7999-1, 2022.


Mark Lutz
Learning Python (5th Edition). *O'Reilly Media, Inc, 2013.*


Luciano Ramalho
Fluent Python (2nd Edition). *O'Reilly Media, Inc, 2021.*


Python Software Foundation
`https://docs.python.org/3/tutorial/`