Xử lý sự kiện
Signals (tín hiệu) và Slots (khe cắm)

Nội dung

- Dùng **Signal/Slot** để kết nối sự kiện.
- Sao chép/chuyển đổi dữ liệu giữa các **Line Edit**.
- Tạo máy tính mini với **Spin Box**.
- Dùng **Scroll Bar** và **Slider** để điều khiển giá trị.
- Quản lý danh sách với **List Widget**: chọn, thêm, hiển thị nhiều mục.
- Dùng **Combo Box** và **Font Combo Box** để chọn giá trị.
- Hiển thị tiến trình với **Progress Bar**.

Giới thiệu

Xử lý sự kiện: Là cơ chế giúp ứng dụng nhận biết sự kiện và thực hiện hành động phù hợp. Hành động này quyết định tiến trình của ứng dụng. Python có cách riêng để lắng nghe và xử lý các sự kiện.

Signal & Slot trong PyQt

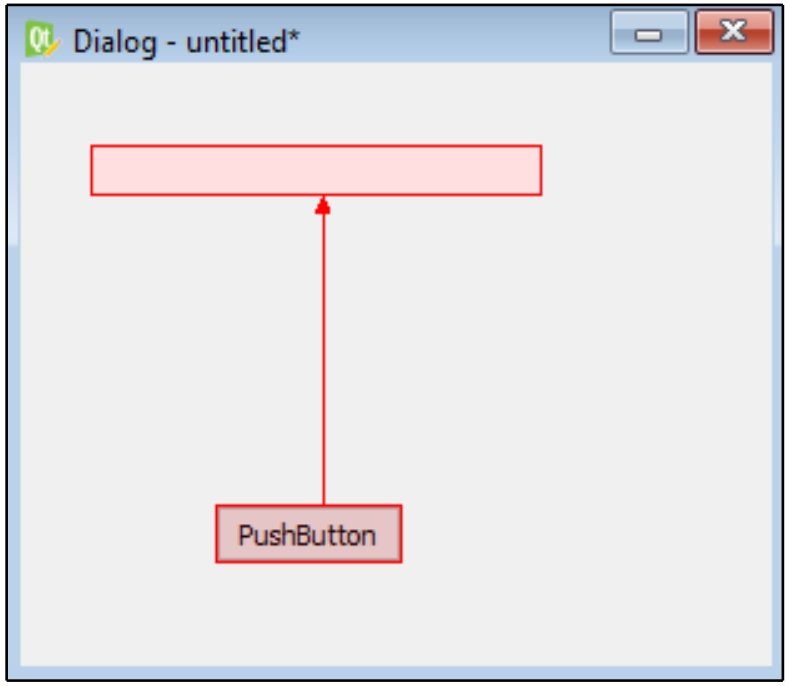
- **Signal (tín hiệu):** sự kiện do widget phát ra (nhấn nút, chọn checkbox, nhấp chuột…).
- **Slot (khe cắm):** phương thức thực thi khi signal xảy ra.
- Hầu hết widget đã có slot sẵn, không cần tự viết.
- Có thể chỉnh sửa kết nối signal-slot qua **Edit | Edit Signals/Slots**.

Làm thế nào để cài đặt nó...

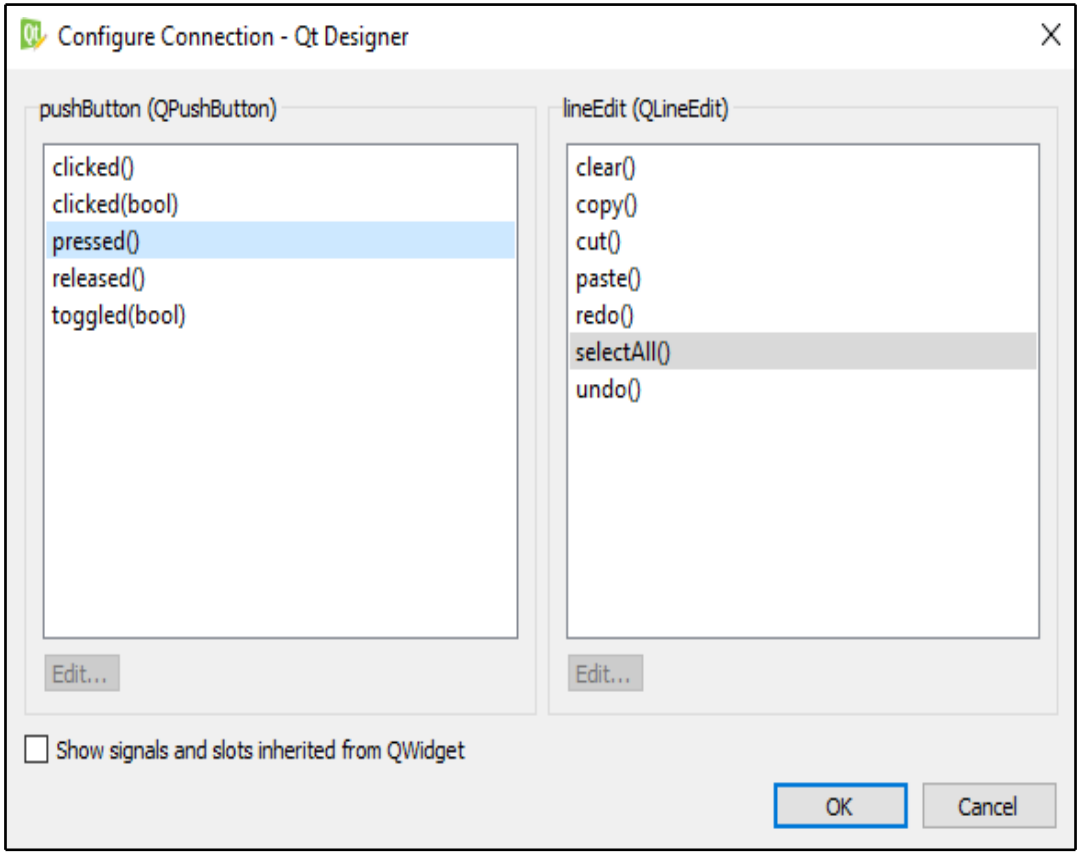
1. Bật chế độ **Edit Signals/Slots** (F4 hoặc từ thanh công cụ).
2. Kết nối hai widget: chọn widget nguồn → kéo sang widget đích → thả chuột.
3. Nhấn **Esc** để hủy khi kéo.
4. Chọn **signal** từ widget nguồn và **slot** từ widget đích trong hộp thoại.
5. Nhấn **OK** để hoàn tất kết nối.

Làm thế nào để cài đặt nó... (t.t)

Ảnh chụp màn hình sau đây cho thấy việc kéo **Push Button** lên widget **Line Edit**:



6. Khi nhả nút chuột trên widget **Line Edit**, ta sẽ nhận được danh sách các signal và slot xác định trước, như sau:



Quản lý kết nối Signal và Slot trong Qt Designer

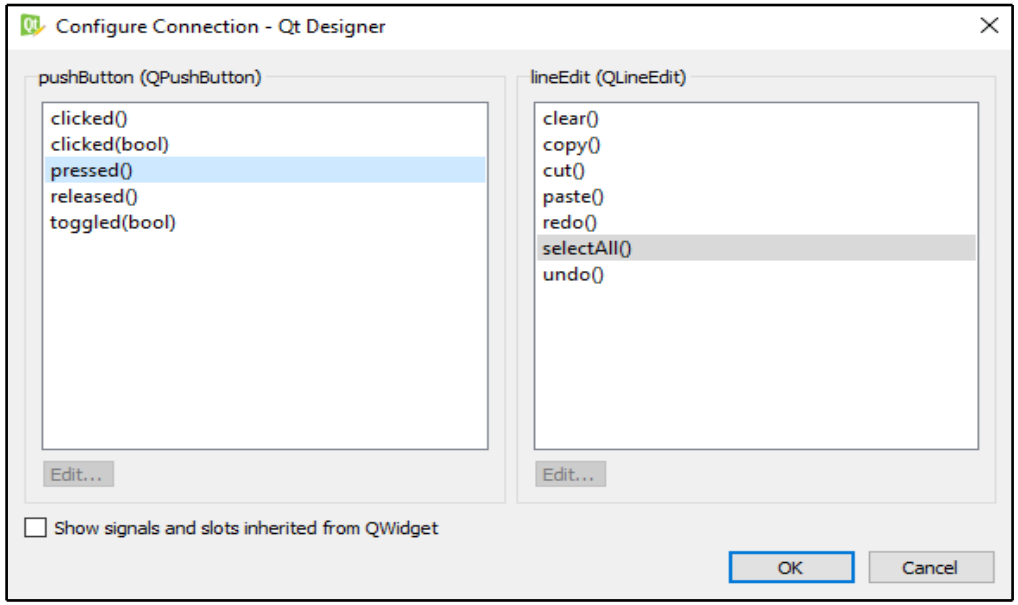
- Khi kết nối, **signal và slot** hiển thị dưới dạng nhãn trên mũi tên nối hai widget.
- **Chỉnh sửa**: Nhấp đúp vào mũi tên hoặc nhãn → hộp thoại *Configure Connection* → sửa signal/slot.
- **Xóa**: Chọn mũi tên → nhấn **Delete**.
- Có thể kết nối **widget với form**: kéo widget → thả trên form (hiện biểu tượng nối đất).
- **Thoát chế độ chỉnh sửa**: Edit | Edit Widgets hoặc nhấn **F3**.

Sao chép và dán văn bản từ một widget Line Edit sang widget khác

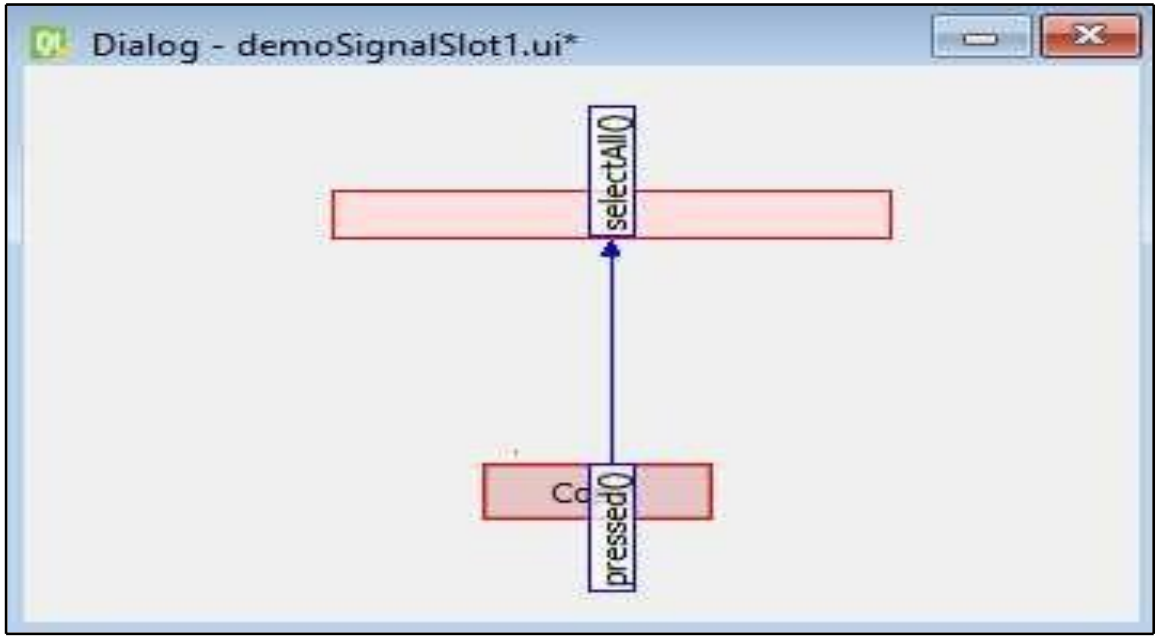
- Nhấn nút → **selectAll()** chọn toàn bộ nội dung trong Line Edit.
- Thả nút → **copy()** sao chép nội dung vào clipboard.
- Nhấn nút khác → **paste()** dán nội dung từ clipboard vào Line Edit khác.

Thực hành...

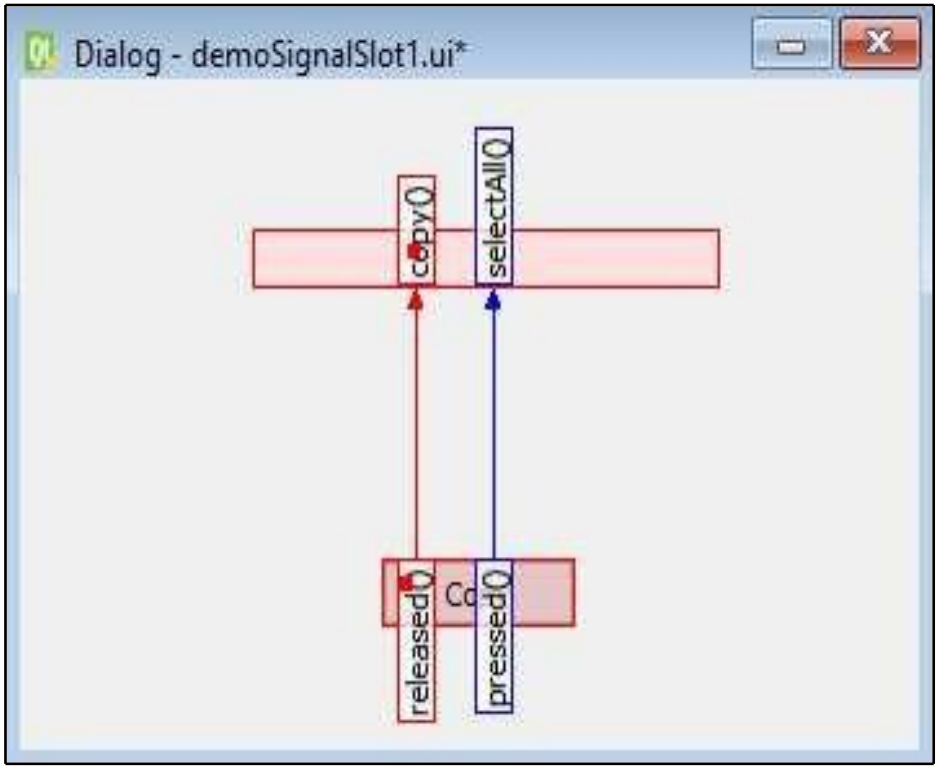
- Tạo ứng dụng có **2 QLineEdit** và **2 QPushButton**.
- Nút 1: **sao chép** nội dung từ Line Edit thứ nhất.
- Nút 2: **dán** nội dung đã sao chép vào Line Edit thứ hai.
- Thiết kế form bằng mẫu **Dialog without Buttons**, kéo-thả widget từ **Widget Box**.
- Kết nối **signal–slot**: chuyển sang chế độ *Edit Signals/Slots*, kéo nút ấn sang Line Edit.
- Chọn signal **pressed()** của QPushButton và slot **selectAll()** của QLineEdit.



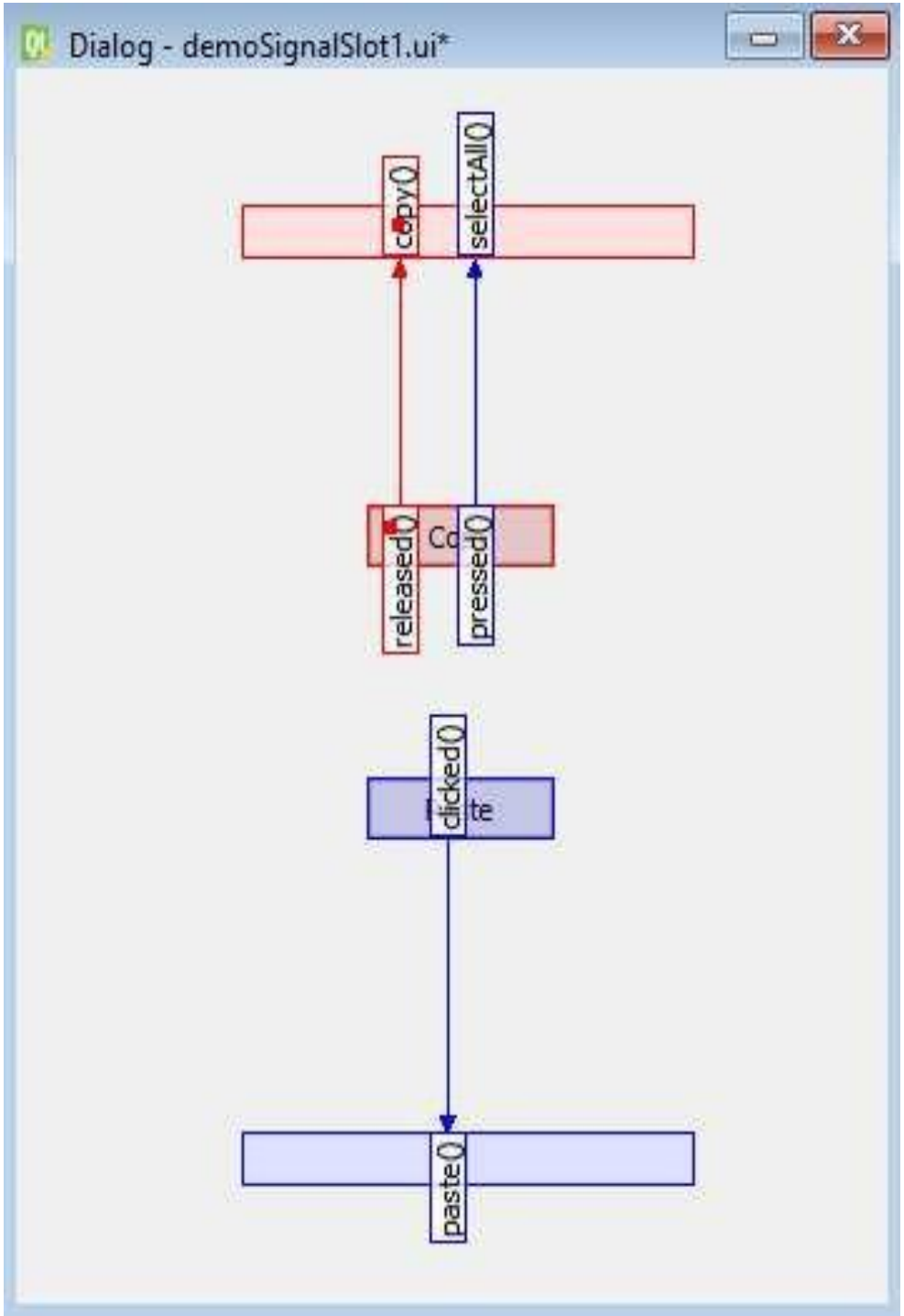
Signal của **Push Button** khi kết nối với **slot của Line Edit** được hiển thị bằng **mũi tên**, cho biết mối liên kết tín hiệu–khe giữa hai widget.



- Đặt **text** của nút **Push Button** là **Copy** để thể hiện chức năng sao chép.
- Kết nối **signal released()** của Push Button với **slot copy()** của **Line Edit** bằng cách kéo thả.
- Trên form sẽ xuất hiện mũi tên biểu thị kết nối **signal–slot** giữa hai widget.



- Thêm **Push Button** và **Line Edit** vào form
- Đặt text cho nút là **Paste**
- Kết nối sự kiện **clicked()** của nút với slot **paste()** của Line Edit
- Lưu form với tên **demoSignal1.ui**



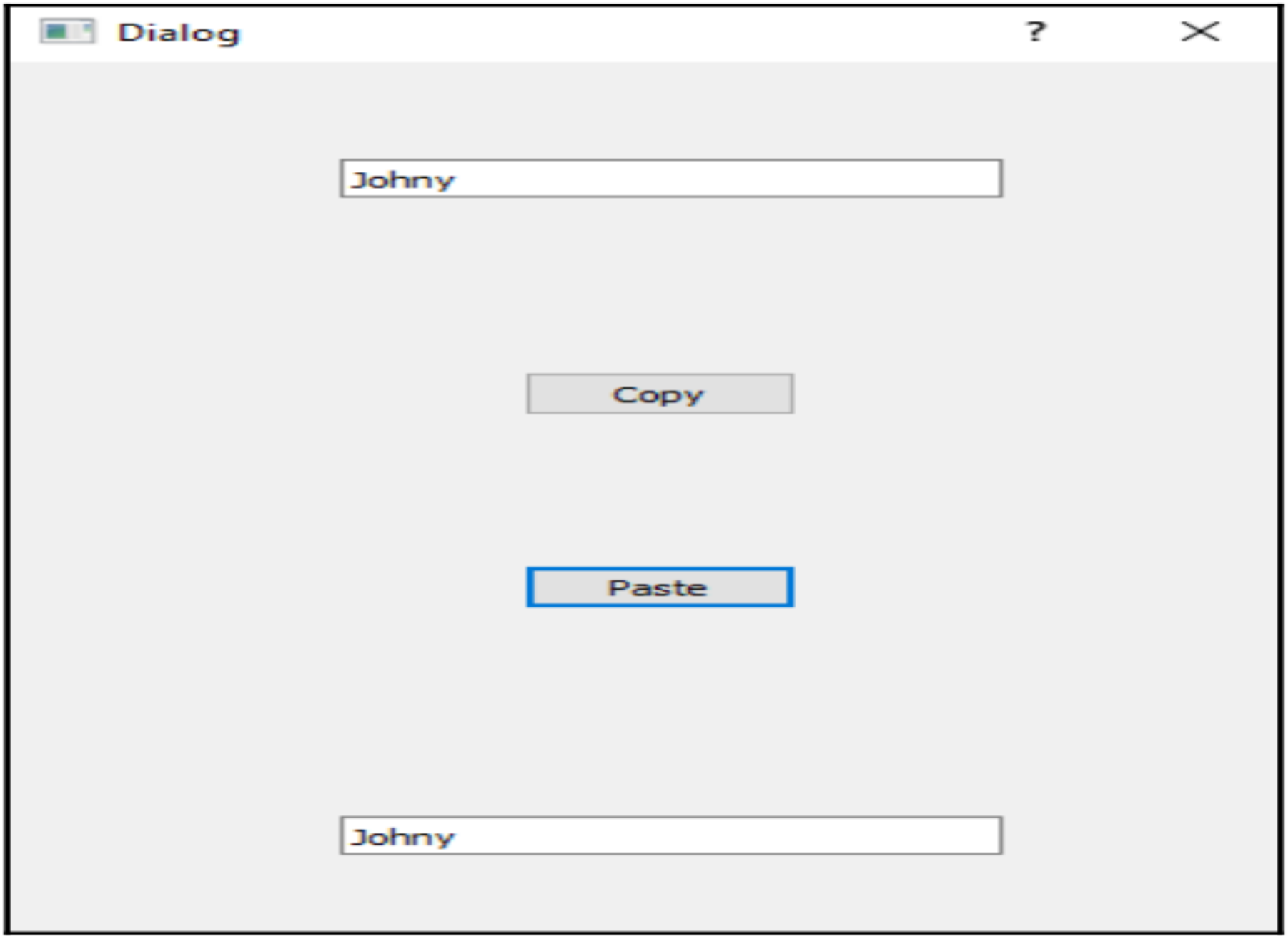
- Giao diện được thiết kế và lưu trong tệp **.ui** (dạng XML), chứa thông tin form, widget và bố cục.
- Tệp **.ui** cần được chuyển sang mã Python bằng công cụ **pyuic5** để sử dụng trong chương trình.
- Trong file Python, ứng dụng dùng hai mô-đun chính:
  - **QtCore**: Cung cấp nền tảng của Qt (xử lý sự kiện, signal–slot, I/O, thời gian, chuỗi…).
  - **QtGui**: Cung cấp các thành phần giao diện người dùng như nút bấm, ô nhập, checkbox, vẽ đồ họa…

Tóm lại: **.ui → pyuic5 → Python + QtCore & QtGui để xây dựng ứng dụng GUI**.

calldemoSignal1.pyw

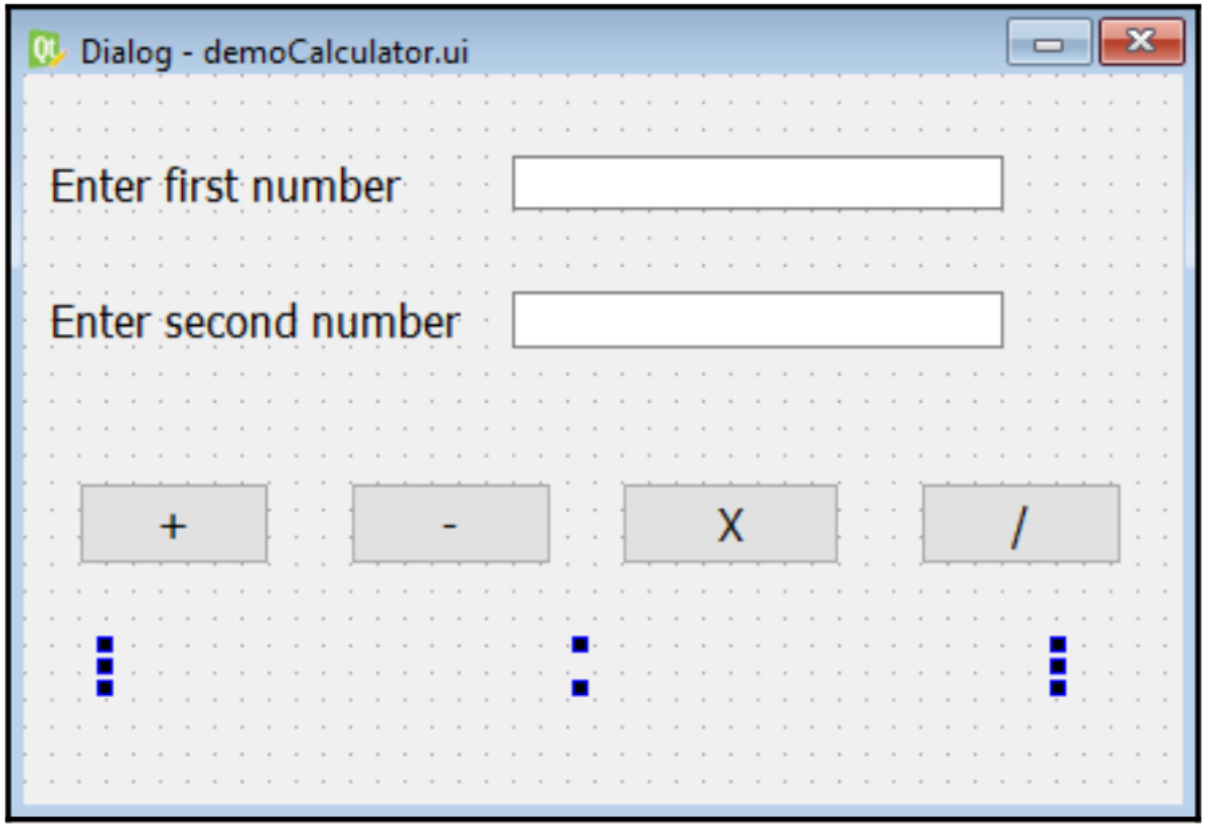
```
import sys
from PyQt5.QtWidgets import QDialog, QApplication
from demoSignalSlot1 import *

class MyForm(QDialog):
  def init (self):
    super().__init__()
    self.ui = Ui_Dialog()
    self.ui.setupUi(self)
    self.show()

if name ==" main ":
  app = QApplication(sys.argv)
  w = MyForm()
  w.show()
  sys.exit(app.exec_())
```

- Mô-đun sys cung cấp tham số dòng lệnh qua sys.argv.
- Mọi ứng dụng PyQt cần đối tượng QApplication để quản lý thông tin hệ thống và vòng lặp sự kiện.
- Tạo cửa sổ chính (MyForm) và gọi show() để hiển thị giao diện.
- app.exec_() khởi động vòng lặp sự kiện, xử lý mọi sự kiện hệ thống và thao tác người dùng; khi đóng cửa sổ chính, ứng dụng kết thúc sạch.
- Trong PyQt, mọi widget đều có thể là cửa sổ cấp cao nhất; super().__init__() gọi hàm tạo của lớp cha (QDialog).
- Giao diện được tạo bằng Ui_Dialog.setupUi(), với QDialog là widget cha của các thành phần giao diện.
- Ứng dụng gồm Line Edit và Push Button: nhập, sao chép và dán văn bản; khi nhấn **Paste**, nội dung đã sao chép được dán vào Line Edit.

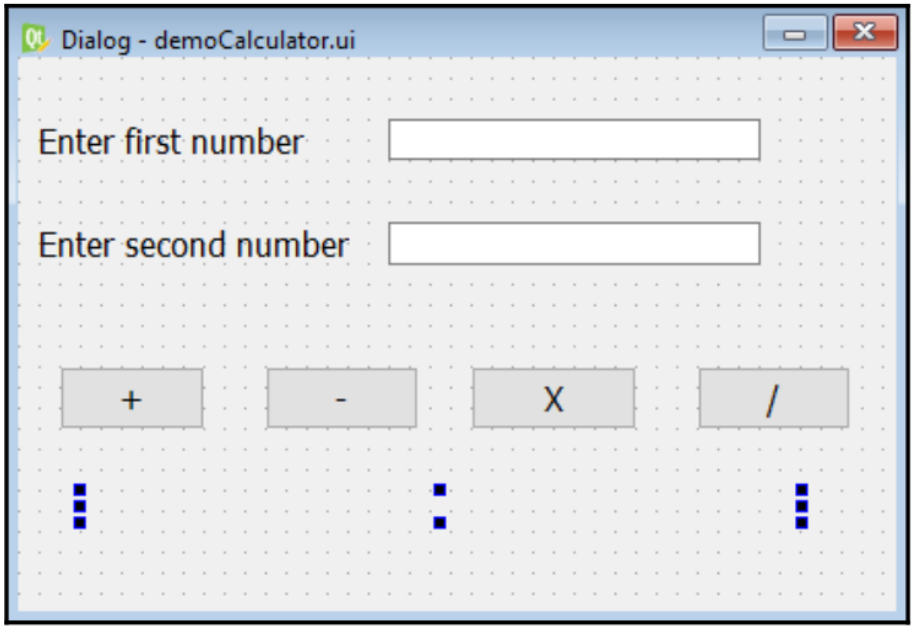

Dialog
?
Johny
Copy
Paste
Johny

Chuyển đổi các loại dữ liệu và tạo một máy tính nhỏ

Line Edit dùng để nhập dữ liệu một dòng và mặc định là **chuỗi**. Khi cần tính toán số nguyên, phải **chuyển chuỗi → số**, tính toán xong rồi **chuyển lại thành chuỗi** để hiển thị lên Label.



- Tạo ứng dụng **Dialog without Buttons**.
- Thêm **3 QLabel, 2 QLineEdit, 4 QPushButton** vào form.
- Đặt **text** cho 2 label nhập số và **objectName** cho tất cả widget theo yêu cầu.
- Gán **text** cho các nút: +, -, x, /.
- Label kết quả để trống, sẽ được gán giá trị bằng **Python** khi tính toán.
- Điều chỉnh độ rộng label để hiển thị kết quả.
- Lưu giao diện là **demoCalculator.ui**.
- Tạo file **callCalculator.pyw** để import giao diện, lấy dữ liệu từ LineEdit và hiển thị kết quả tính.

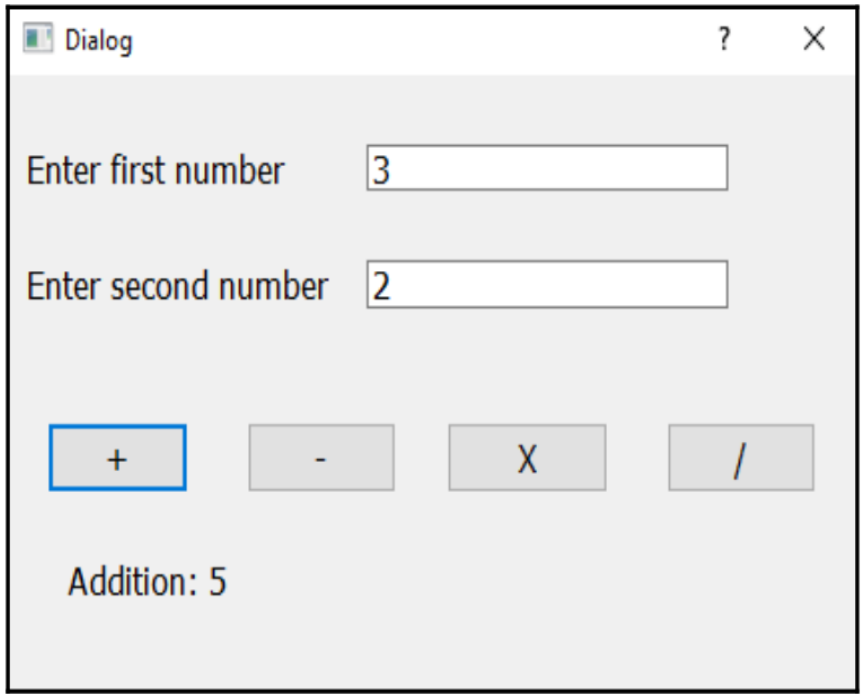


```
import sys
from PyQt5.QtWidgets import QDialog, QApplication
from demoCalculator import *
class MyForm(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.ui.pushButtonPlus.clicked.connect(self.addtwonum)
        self.ui.pushButtonSubtract.clicked.connect
        (self.subtracttwonum)
        self.ui.pushButtonMultiply.clicked.connect
        (self.multiplytwonum)
        self.ui.pushButtonDivide.clicked.connect(self.dividetwonum)
        self.show()
```

Sự kiện **clicked()** của pushButtonPlus được kết nối với phương thức **addtwonum()** để hiển thị tổng các số được nhập trong hai widget Line Edit.

```
def addtwonum(self):
    if len(self.ui.lineEditFirstNumber.text())!=0:
            a=int(self.ui.lineEditFirstNumber.text())
    else:
            a=0
    if len(self.ui.lineEditSecondNumber.text())!=0:
            b=int(self.ui.lineEditSecondNumber.text())
    else:
            b=0
            sum=a+b
    self.ui.labelResult.setText("Addition: " +str(sum))
```

Trong phương thức `addtwonum()`, trước tiên xác thực `lineEditFirstNumber` và `lineEditSecondNumber` để đảm bảo rằng nếu **Line Edit** bị bỏ trống, thì giá trị của mặc định bằng không.

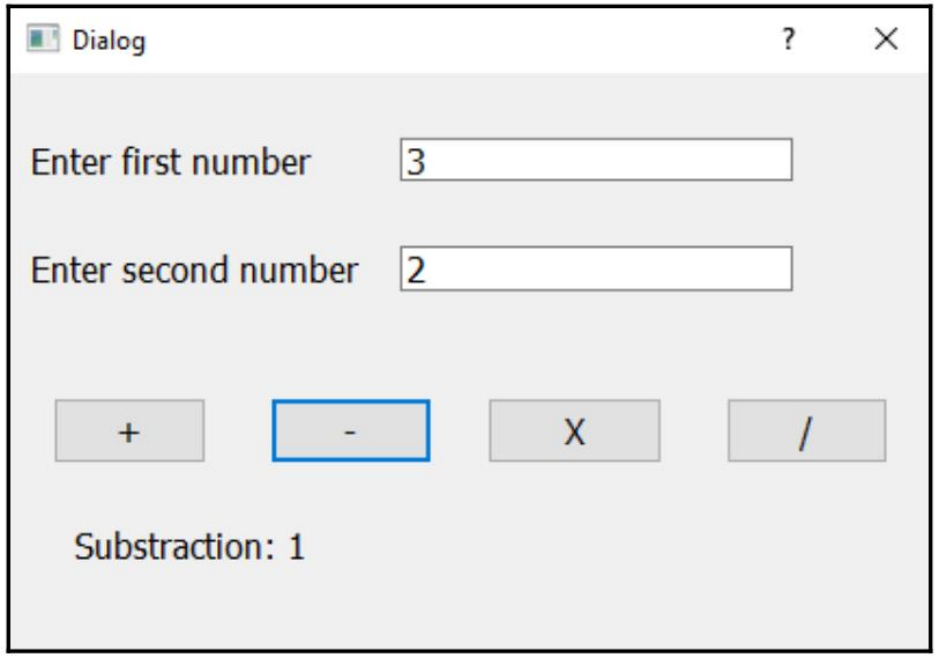


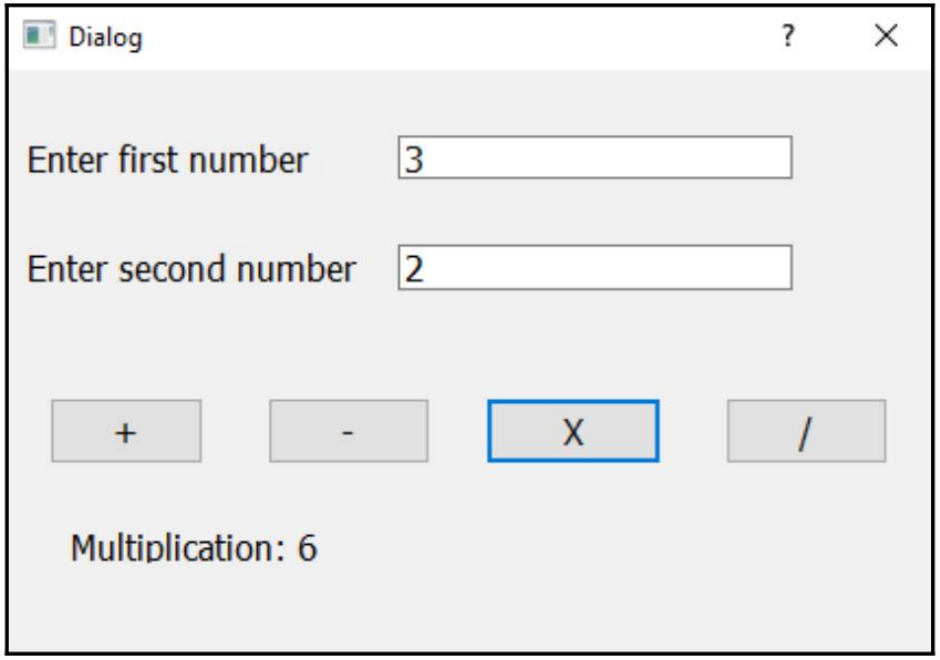
```
def subtracttwonum(self):
    if len(self.ui.lineEditFirstNumber.text())!=0:
            a=int(self.ui.lineEditFirstNumber.text())
    else:
            a=0
    if len(self.ui.lineEditSecondNumber.text())!=0:
            b=int(self.ui.lineEditSecondNumber.text())
    else:
            b=0
            diff=a-b
    self.ui.labelResult.setText("Substraction: " +str(diff))
```

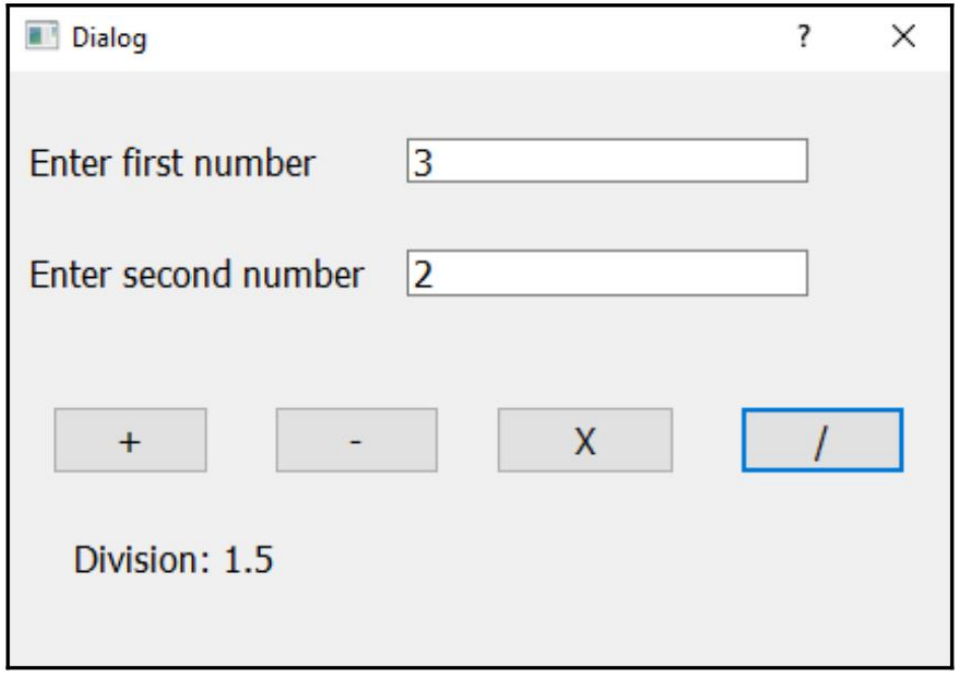
Phương thức subtracttwonum() lấy dữ liệu từ hai ô nhập, chuyển thành số nguyên, thực hiện phép trừ và hiển thị kết quả lên nhãn kết quả.

```
def multiplytwonum(self):
    if len(self.ui.lineEditFirstNumber.text())!=0:
            a=int(self.ui.lineEditFirstNumber.text())
    else:
            a=0
    if len(self.ui.lineEditSecondNumber.text())!=0:
            b=int(self.ui.lineEditSecondNumber.text())
    else:
            b=0
            mult=a*b
    self.ui.labelResult.setText("Multiplication: " +str(mult))
def dividetwonum(self):
    if len(self.ui.lineEditFirstNumber.text())!=0:
            a=int(self.ui.lineEditFirstNumber.text())
    else:
            a=0
    if len(self.ui.lineEditSecondNumber.text())!=0:
            b=int(self.ui.lineEditSecondNumber.text())
    else:
            b=0
            division=a/b
    self.ui.labelResult.setText("Division: "+str(round
    (division,2)))
```

```
if __name__=="__main__":
    app = QApplication(sys.argv)
    w = MyForm()
    w.show()
    sys.exit(app.exec_())
```







Sử dụng widget **Spin Box**

- Spin Box là tiện ích cho phép người dùng chọn giá trị **nguyên, số thực hoặc văn bản** từ các tùy chọn có sẵn, không nhập dữ liệu tùy ý. Giá trị có thể tăng/giảm bằng **nút lên/xuống**, **phím mũi tên** hoặc **nhập trực tiếp**.
- Spin Box gồm hai loại: **QSpinBox** (số nguyên) và **QDoubleSpinBox** (số thực), với các hàm chính như lấy/đặt giá trị, tiền tố–hậu tố, bước nhảy và giới hạn min–max.

Signal được phát ra bởi QSpinBox như sau:

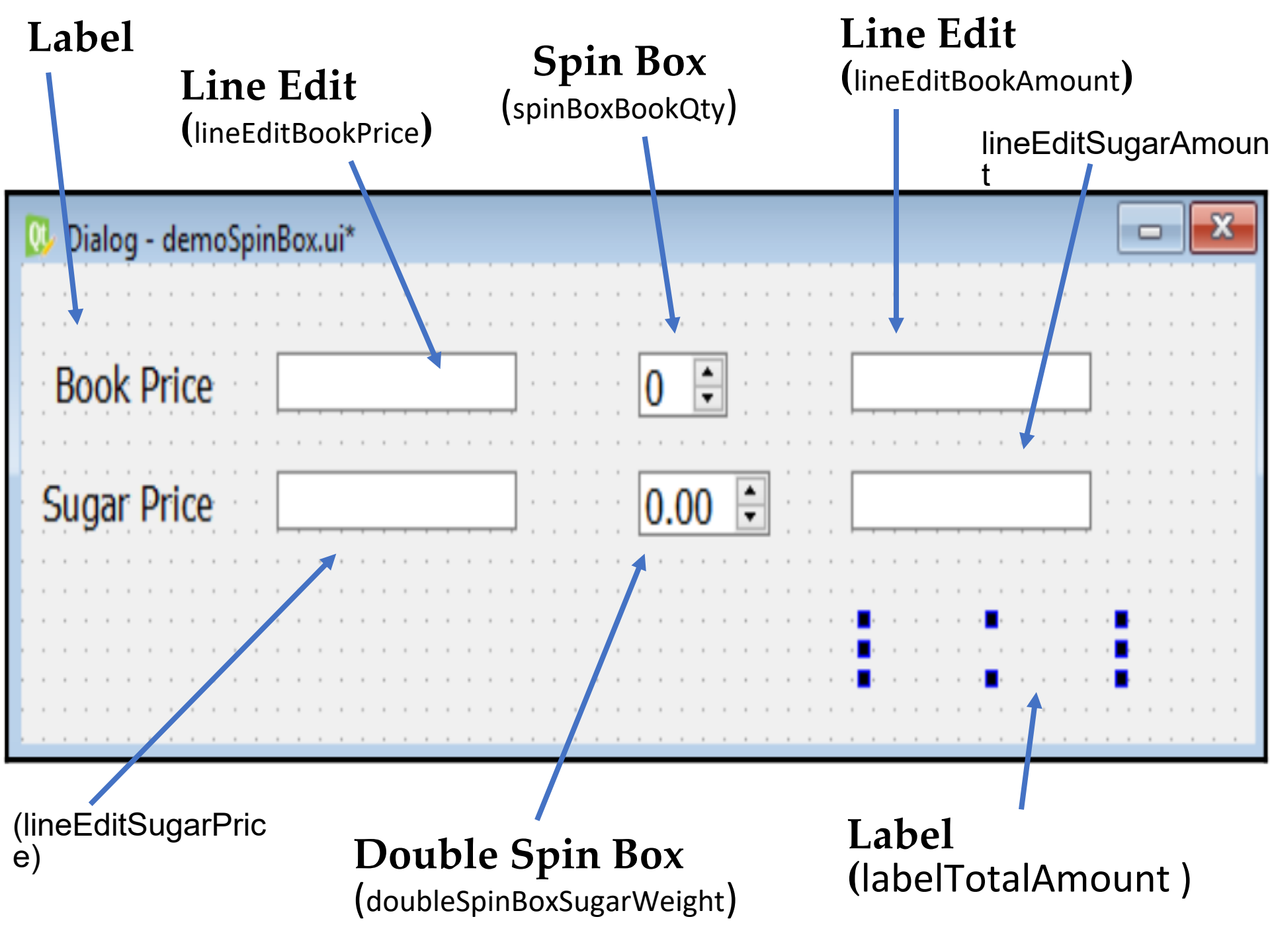
- **`valueChanged()`:** khi chọn nút tăng/giảm hoặc sử dụng phương thức `setValue()`
- **`editingFinished()`**: khi hết focus trên nó

QDoubleSpinBox dùng để nhập và xử lý số thực (float), có các phương thức giống QSpinBox. Mặc định hiển thị 2 chữ số thập phân và có thể điều chỉnh độ chính xác bằng cách làm tròn số.

Ghi chú: Các thuộc tính **minimum**, **maximum**, **singleStep**, **value** mặc định là **0, 99, 1** và **0** và của double spin box là **0.000000**, **99.990000**, **1.000000** và **0.000000**

Bài tập SpinBox

- Viết chương trình cho người dùng nhập giá (price) cho một cuốn sách, tiếp theo là số lượng (quantity) những cuốn sách đã mua của khách hàng, và sẽ hiển thị tổng tiền sách.
- Ứng dụng sẽ nhắc nhập giá cho 1 kg đường, sau đó là số lượng đường mà người dùng đã mua, và ứng dụng sẽ hiển thị tổng lượng đường.
- Số lượng sách và đường sẽ được nhập thông qua spin box và double spin box.


Label
Line Edit
(lineEditBookPrice)
Spin Box
(spinBoxBookQty)
Line Edit
(lineEditBookAmount)
lineEditSugarAmoun
t
Dialog - demoSpinBox.ui*
Book Price
0
Sugar Price
0.00
(lineEditSugarPric
e)
Double Spin Box
(doubleSpinBoxSugarWeight)
Label
(labelTotalAmount)

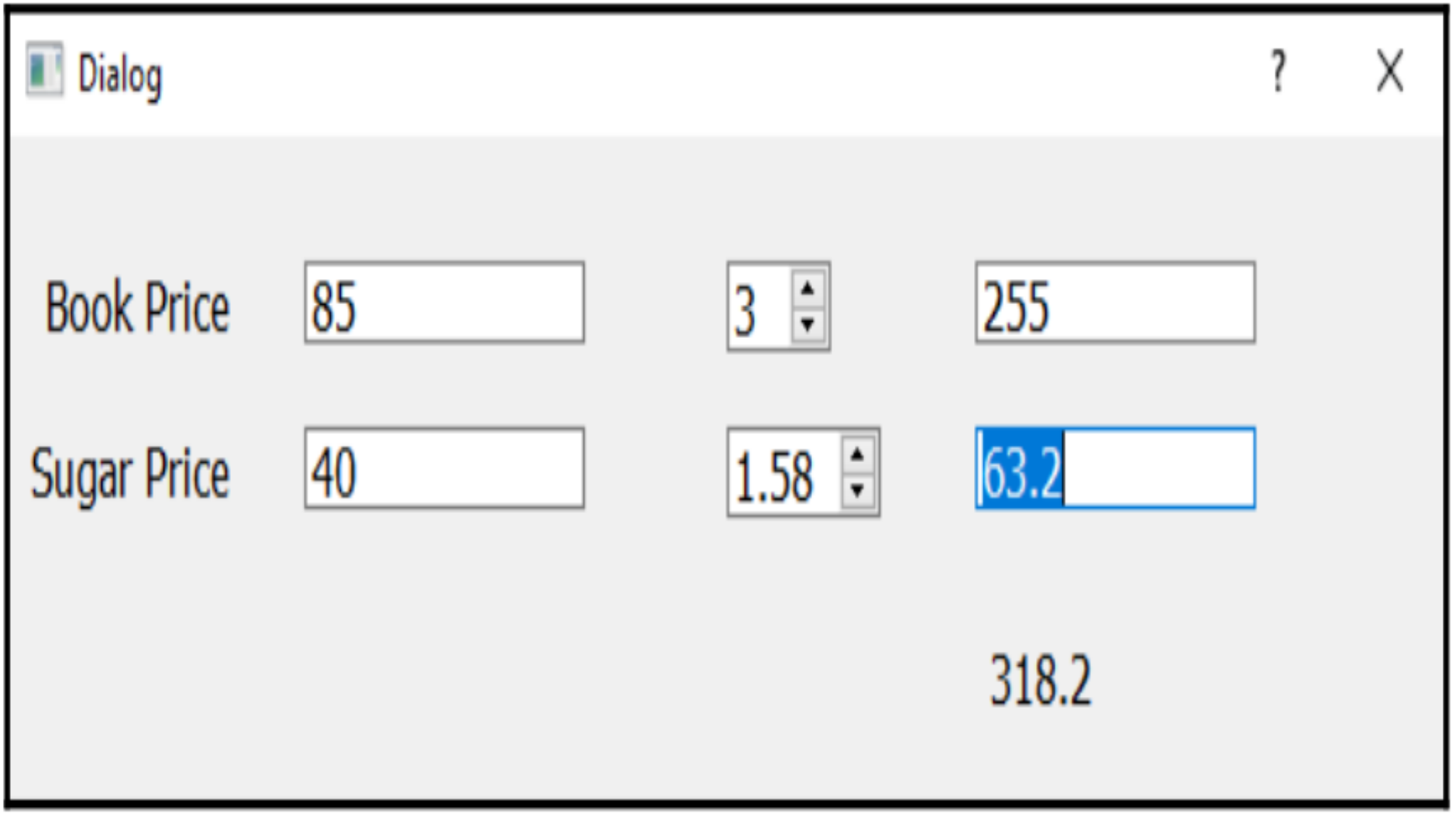
Gợi ý Code xử lý

slot cho gắn với Spin Box

```
def result1(self):
  if len(self.ui.lineEditBookPrice.text())!=0:
    bookPrice=int(self.ui.lineEditBookPrice.text())
  else:
    bookPrice=0
    totalBookAmount=self.ui.spinBoxBookQty.value() * bookPrice
    self.ui.lineEditBookAmount.setText(str(totalBookAmount))
```

slot cho gắn với Double Spin Box

```
def result2(self):
  if len(self.ui.lineEditSugarPrice.text())!=0:
    sugarPrice=float(self.ui.lineEditSugarPrice.text())
  else:
    sugarPrice=0
    totalSugarAmount=self.ui.doubleSpinBoxSugarWeight.value() * suga
rPrice
    self.ui.lineEditSugarAmount.setText(str(round(totalSugarAmount,2
)))
    totalBookAmount=int(self.ui.lineEditBookAmount.text())
    totalAmount=totalBookAmount+totalSugarAmount
    self.ui.labelTotalAmount.setText(str(round(totalAmount,2)))
```


Dialog
?
X
Book Price
85
3
255
Sugar Price
40
1.58
63.2
318.2

Thanh cuộn (scrollbars) và thanh trượt (sliders)

- **Scrollbar**: Dùng để xem phần nội dung hoặc hình ảnh bị ẩn khi không gian hiển thị bị giới hạn, bằng cách kéo tay cầm thanh cuộn.
- **Slider**: Dùng để chọn **một giá trị** trong một **khoảng xác định** (từ minimum đến maximum) bằng cách di chuyển tay cầm.

Ghi chú: `QScrollBar` chỉ cung cấp các giá trị nguyên.

Scrollbars (thanh cuộn ngang/dọc) được tạo bằng HorizontalScrollBar và VerticalScrollBar, thuộc lớp QScrollBar.

Một scrollbar gồm 3 phần chính:

Slider handle: kéo để di chuyển đến vị trí bất kỳ trong nội dung.

Scroll arrows: mũi tên dùng để cuộn từng bước.

Page control: vùng nền của thanh cuộn; khi nhấp sẽ cuộn theo từng trang, số bước cuộn được xác định bởi thuộc tính pageStep (thiết lập bằng setPageStep()).

Phương thức được sử dụng cụ thể để thiết lập và truy xuất giá trị từ scrollbars *là* `value()`

Signal tạo ra từ **`QScrollBar`** là

- **valueChanged()**: Phát ra khi giá trị scrollbar thay đổi.
- **sliderPressed()**: Phát ra khi bắt đầu kéo thanh trượt.
- **sliderMoved()**: Phát ra trong lúc đang kéo thanh trượt.
- **sliderReleased()**: Phát ra khi thả thanh trượt.
- **actionTriggered()**: Phát ra khi thanh cuộn thay đổi do thao tác người dùng.

Sliders (QSlider) dùng để nhập và biểu diễn **giá trị số nguyên** một cách **trực quan và tương tác**. Khác với scrollbar (dùng để cuộn nội dung), slider cho phép người dùng **kéo tay cầm** trên rãnh ngang hoặc dọc để chọn giá trị. Khi thao tác, slider phát ra các **tín hiệu** như valueChanged(), sliderPressed(), sliderMoved(), sliderReleased().

Ghi chú*: Slide có các thuộc tính **minimum, maximum, singleStep, pageStep, value** với các giá trị mặc định lần lượt là **0, 99, 1, 10** và **0**.*

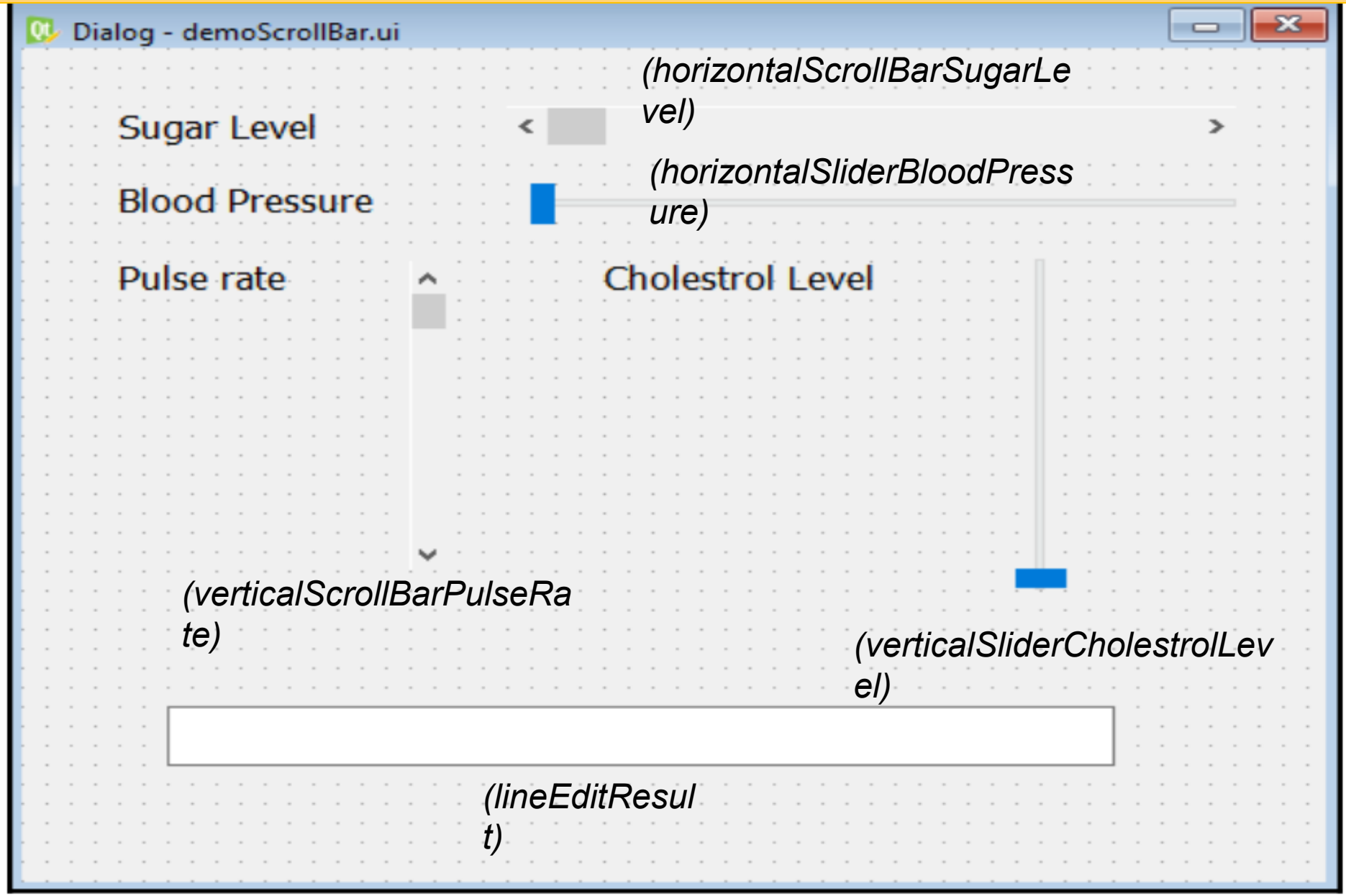
Bài tập về scrollbars và sliders

Tạo ứng dụng bao gồm các scrollbar/slider như sau.

Scrollbar nằm ngang và slider ngang sẽ tương ứng với mức đường và huyết áp. Nghĩa là, khi di chuyển thanh cuộn ngang, mức đường của bệnh nhân sẽ được hiển thị thông qua widget **Line Edit**. Tương tự, slider ngang, khi được di chuyển, sẽ biểu thị huyết áp và sẽ được hiển thị thông qua widget **Line Edit**.

scrollbar/slider dọc sẽ tương ứng với nhịp tim và mức cholesterol. Khi di chuyển scrollbar dọc thì nhịp tim sẽ được hiển thị thông qua widget Line Edit và khi di chuyển slider dọc, mức cholesterol sẽ được hiển thị thông qua widget **Line Edit**.

Tạo ứng dụng theo mẫu **Dialog without Buttons** và thiết kế form như sau

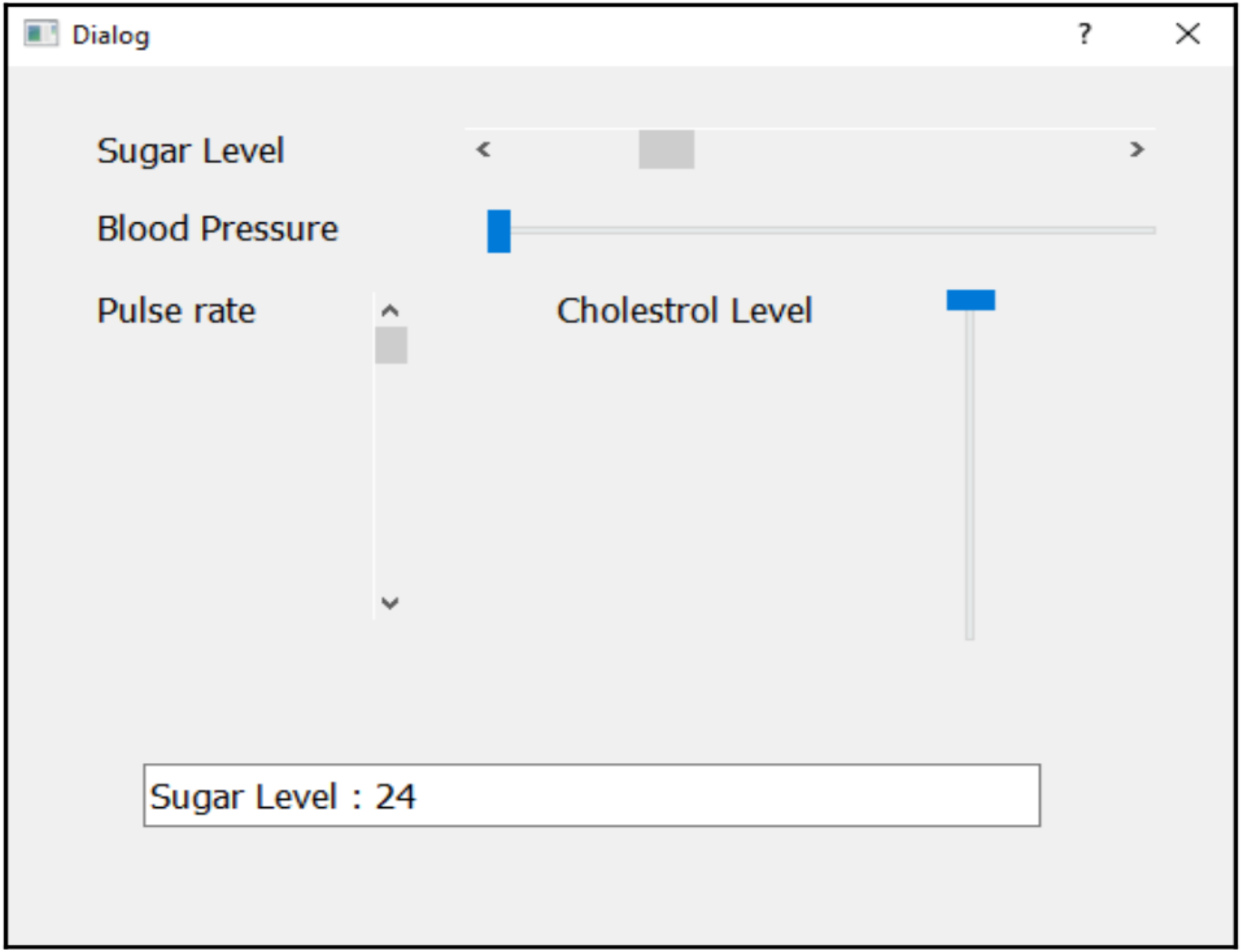


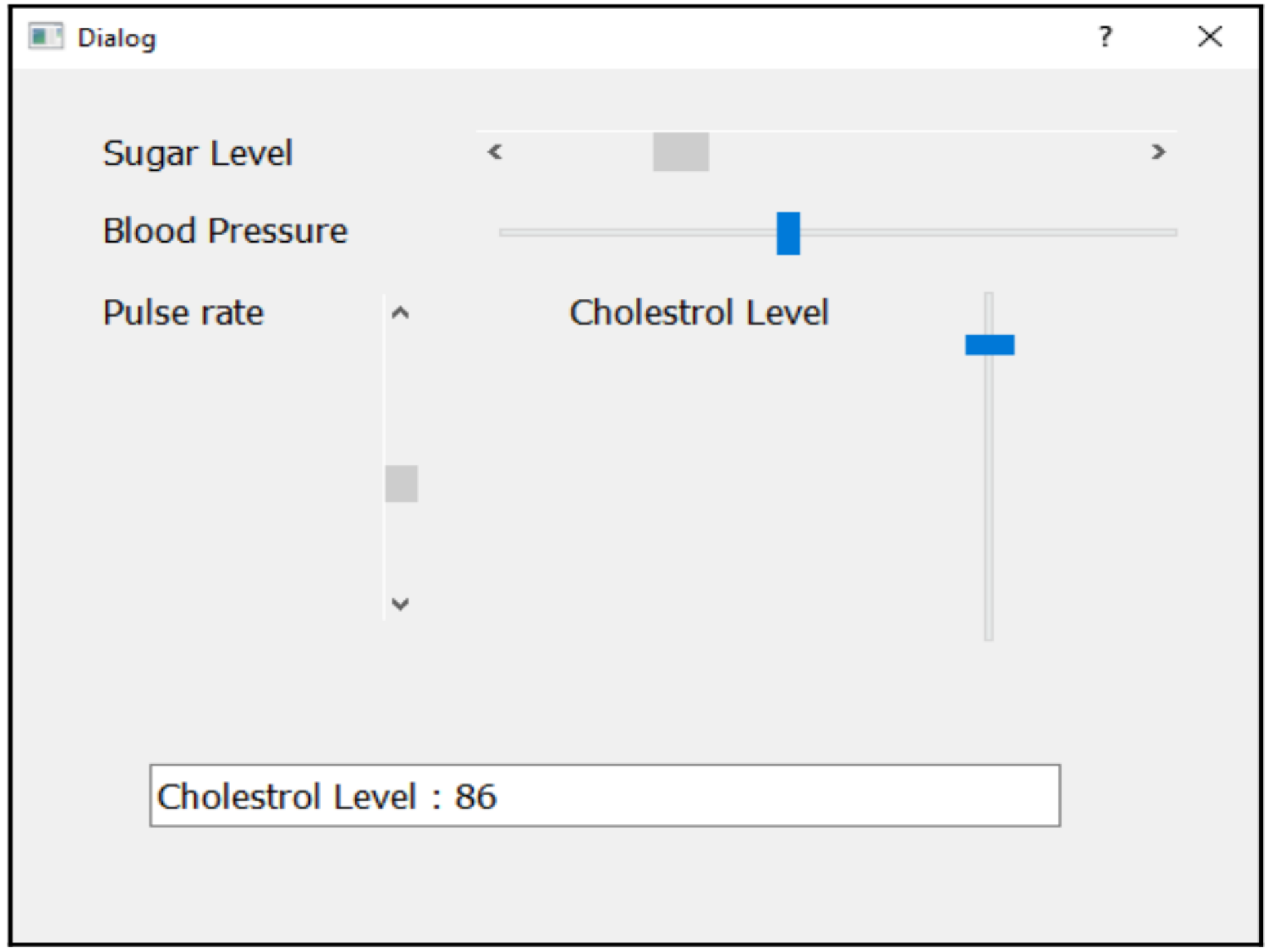
Gợi ý code gọi giao diện ra dùng

```
import sys
from PyQt5.QtWidgets import QDialog, QApplication
from demoScrollBar import *
class MyForm(QDialog):
  def __init__(self):
    super().__init__()
    self.ui = Ui_Dialog()
    self.ui.setupUi(self)
    self.ui.horizontalScrollBarSugarLevel.valueChanged.connect(se
lf.scrollhorizontal)

    self.ui.verticalScrollBarPulseRate.valueChanged.connect(self.
scrollvertical)

    self.ui.horizontalSliderBloodPressure.valueChanged.connect(se
lf.sliderhorizontal)

    self.ui.verticalSliderCholestrolLevel.valueChanged.connect(se
lf.slidervertical)

    self.show()
```

Gợi ý code xử lý

```
def scrollhorizontal(self,value):
  self.ui.lineEditResult.setText("Sugar Level :
 " + str(value))

def scrollvertical(self, value):
  self.ui.lineEditResult.setText("Pulse Rate :
" + str(value))

def sliderhorizontal(self, value):
  self.ui.lineEditResult.setText("Blood Pressur
e : " + str(value))

def slidervertical(self, value):
  self.ui.lineEditResult.setText("Cholestrol Le
vel : " + str(value))
```


Dialog
?
Sugar Level
Blood Pressure
Pulse rate
Cholestrol Level
Sugar Level : 24

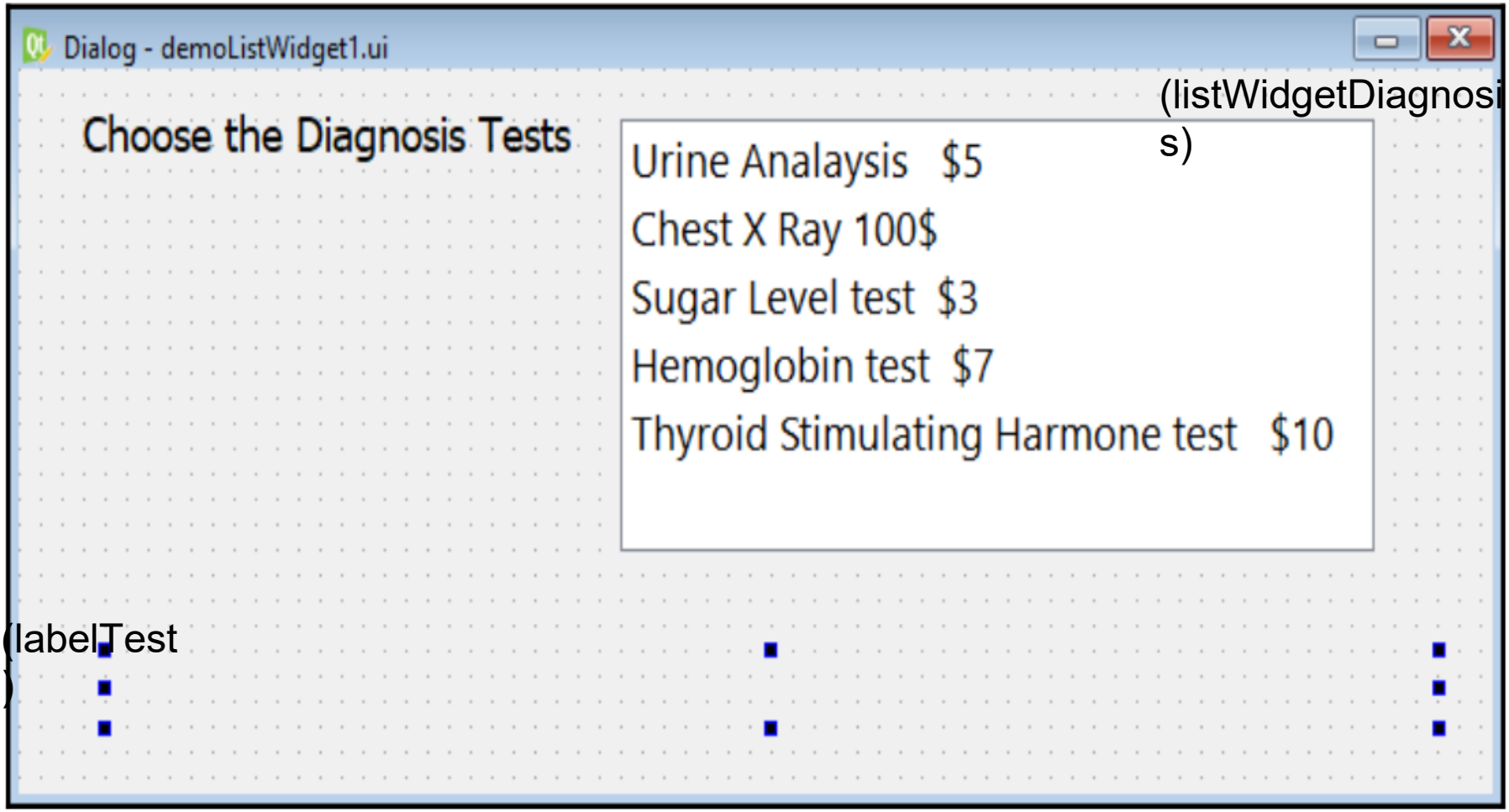

Dialog
Sugar Level
Blood Pressure
Pulse rate
Cholestrol Level
Cholestrol Level : 86

Sử dụng **List** widget

- **QListWidget (List Widget)** dùng để hiển thị danh sách nhiều giá trị, dễ quản lý và mở rộng.
- Cho phép **xem, thêm, sửa, xóa** các mục trong danh sách.
- Mỗi mục trong danh sách là một **QListWidgetItem**.
- Cung cấp các **hàm** để thêm, xóa, đếm và chọn item.
- Có các **signal** để theo dõi thay đổi item, dòng và nội dung được chọn.

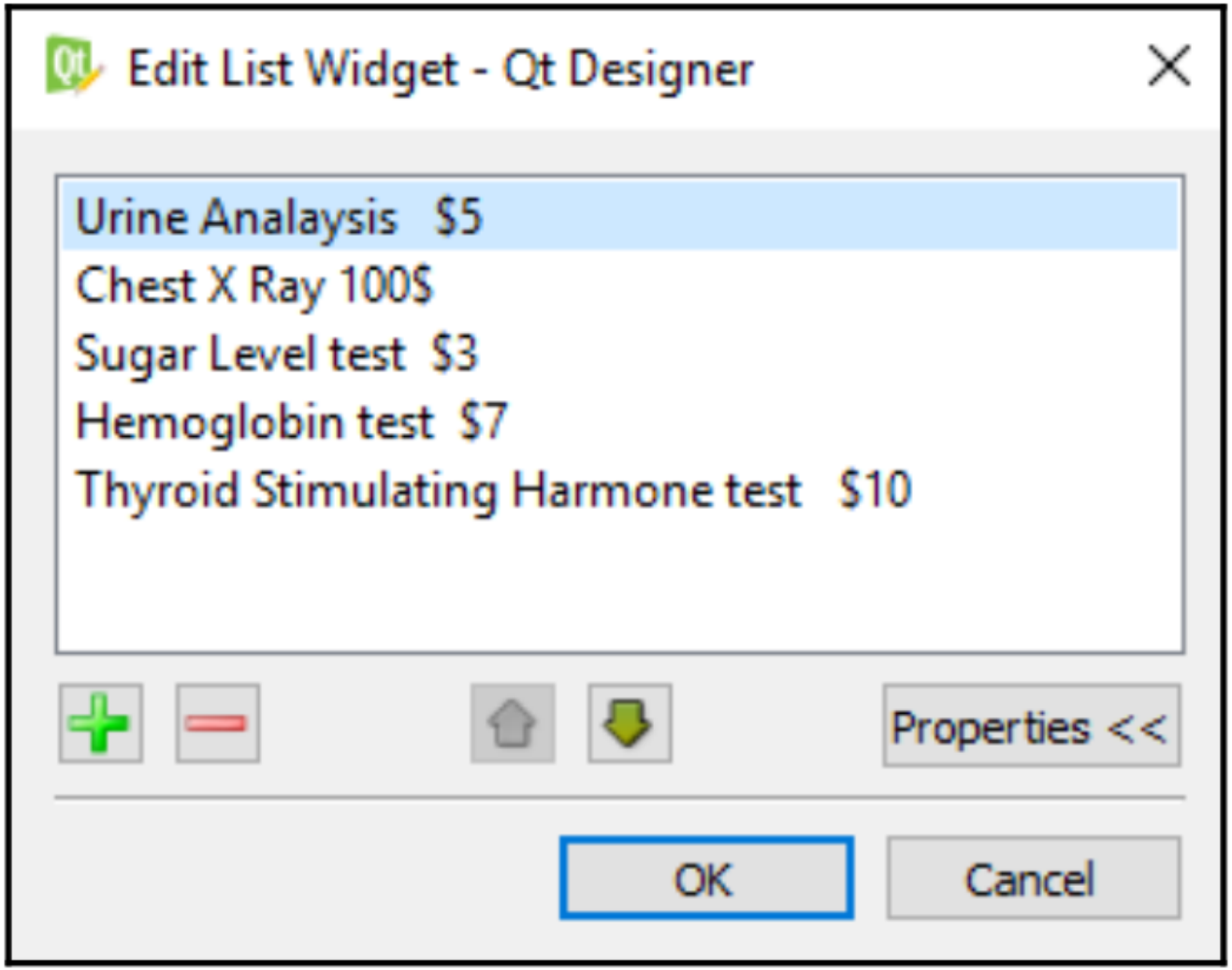
Bài tập ứng dụng với List widget như sau

Ứng dụng hiển thị danh sách chẩn đoán bằng **List widget**; khi người dùng chọn một mục, nội dung sẽ được hiển thị trên **Label widget**.



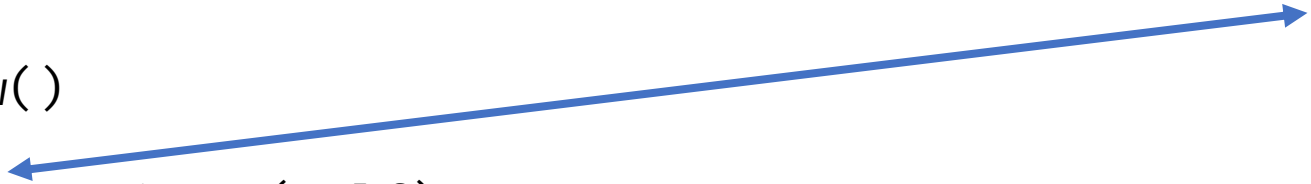
Gợi ý cách thức thêm item vào List

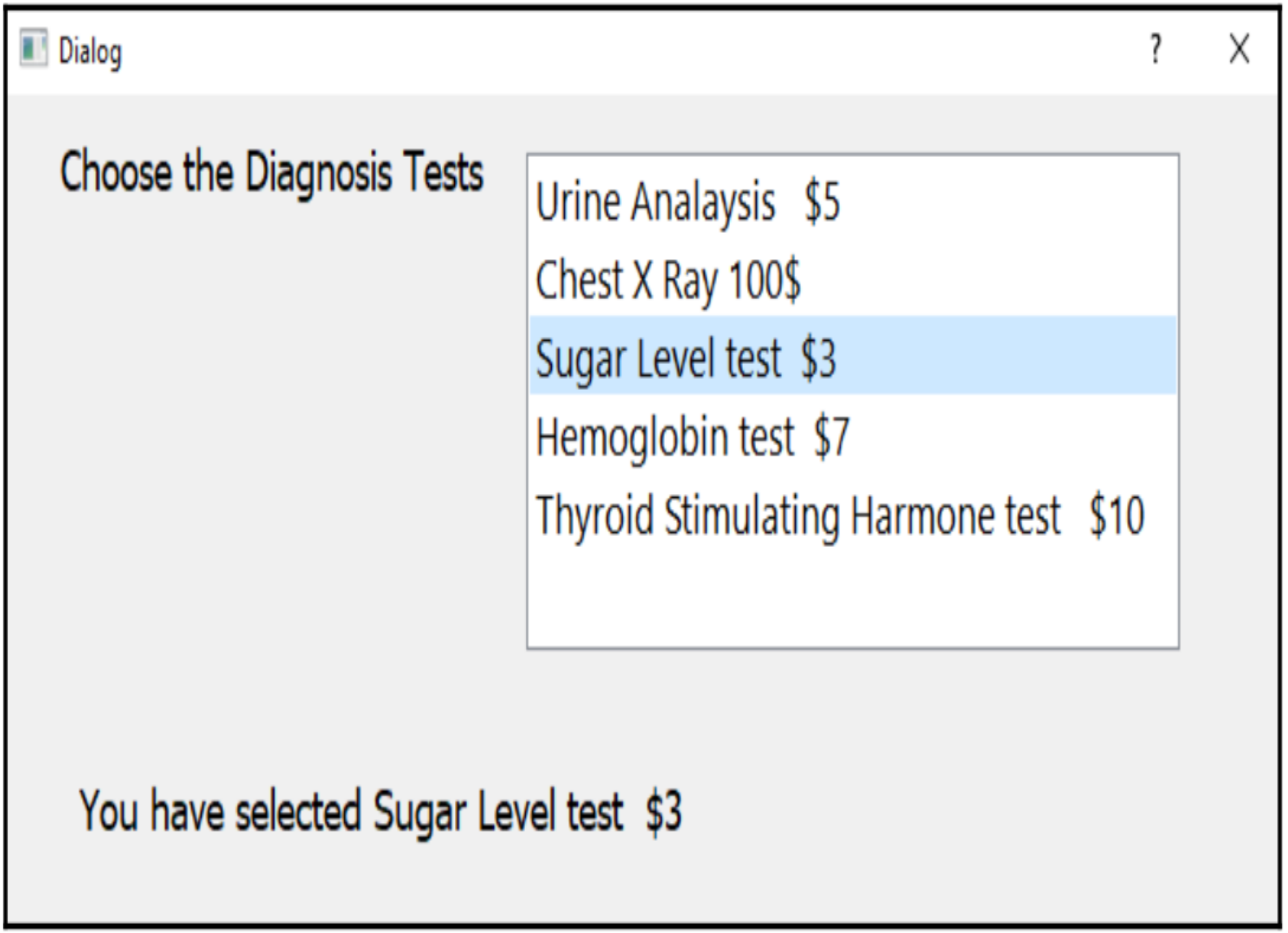
Nhấp chuột phải vào List widget và từ menu ngữ cảnh mở ra, chọn tùy chọn Edit Items, thêm các xét nghiệm chẩn đoán từng cái một, sau đó nhấp vào nút + ở dưới cùng:



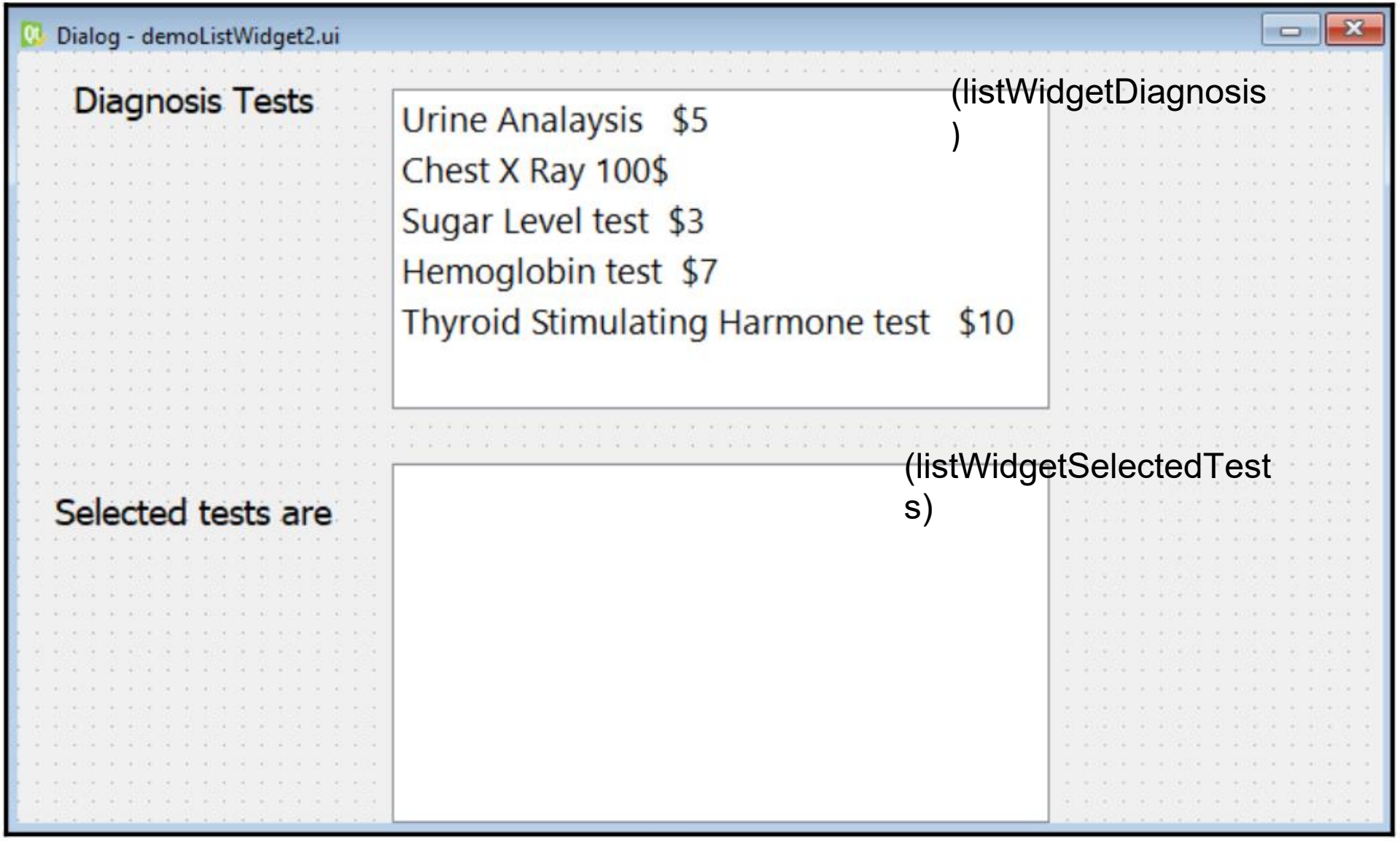
Gợi ý code gọi giao diện và xử lý

```
import sys
from PyQt5.QtWidgets import QDialog, QApplication
from demoListWidget1 import *
class MyForm(QDialog):
  def __init__(self):
    super().__init__()
    self.ui = Ui_Dialog()
    self.ui.setupUi(self)
    self.ui.listWidgetDiagnosis.itemClicked.connect(self.dispSele
ctedTest)
    self.show()

  def dispSelectedTest(self):
    self.ui.labelTest.setText("You have selected" +
              self.ui.listWidgetDiagnosis.currentItem().text())

if __name__=="__main__":
  app = QApplication(sys.argv)
  w = MyForm()
  w.show()
  sys.exit(app.exec_())
```




Dialog
?
X
Choose the Diagnosis Tests
Urine Analaysis $5
Chest X Ray 100$
Sugar Level test $3
Hemoglobin test $7
Thyroid Stimulating Harmone test $10
You have selected Sugar Level test $3

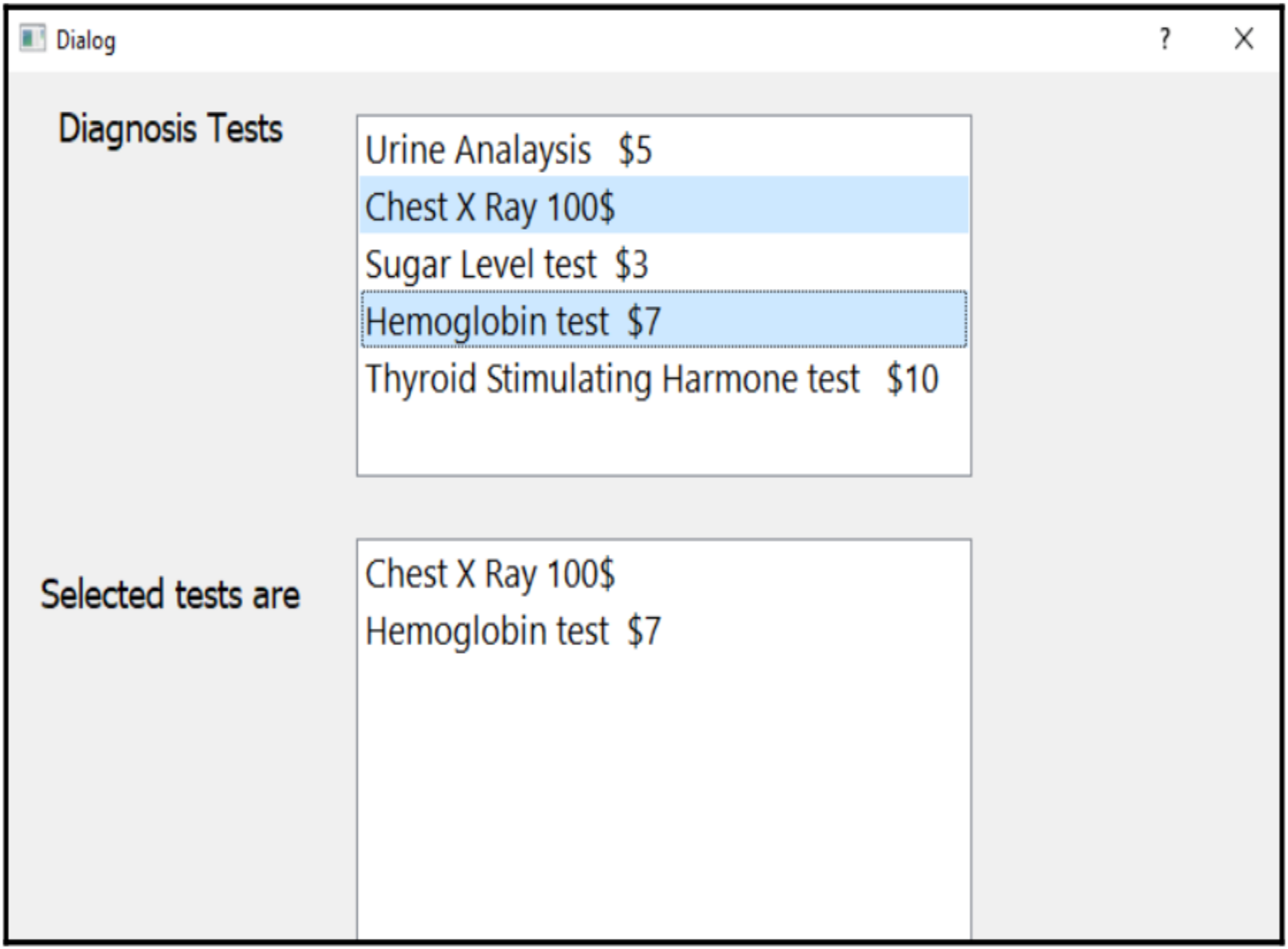
Chọn nhiều mục danh sách từ một List Widget và hiển thị chúng trong List Widget khác



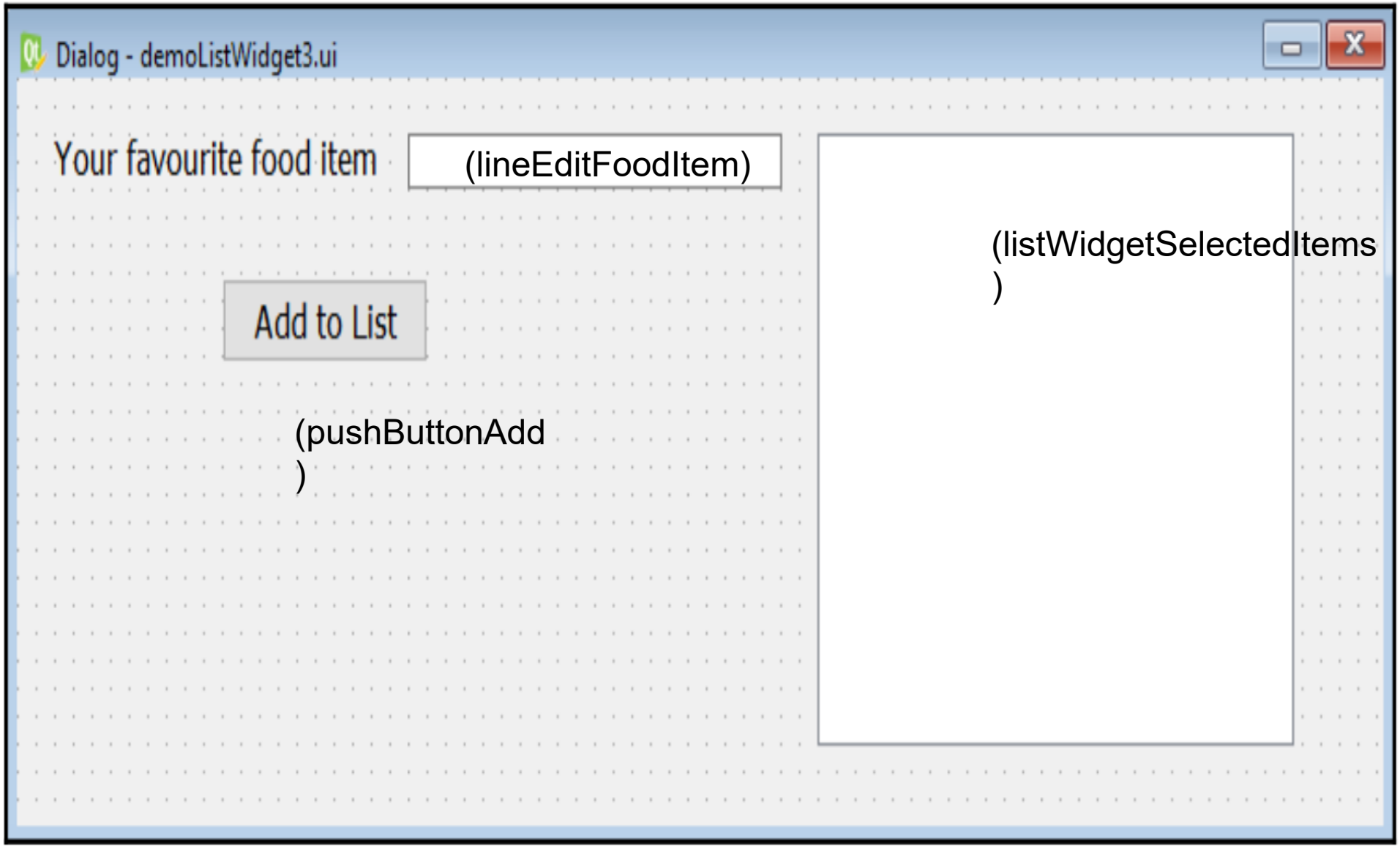
Thiết lập List widget chọn được nhiều

Để kích hoạt tính năng cho phép chọn nhiều mục từ List widget, ta thiết lập thuộc tính **selectMode** từ `SingleSelection` thành `MultiSelection`.

```
class MyForm(QDialog):
  def __init__(self):
    super().__init__()
    self.ui = Ui_Dialog()
    self.ui.setupUi(self)
    self.ui.listWidgetDiagnosis.itemSelectionChanged.conne
ct(self.dispSelectedTest)
    self.show()

  def dispSelectedTest(self):
    self.ui.listWidgetSelectedTests.clear()
    items = self.ui.listWidgetDiagnosis.selectedItems()
    for i in list(items):
      self.ui.listWidgetSelectedTests.addItem(i.text())
```

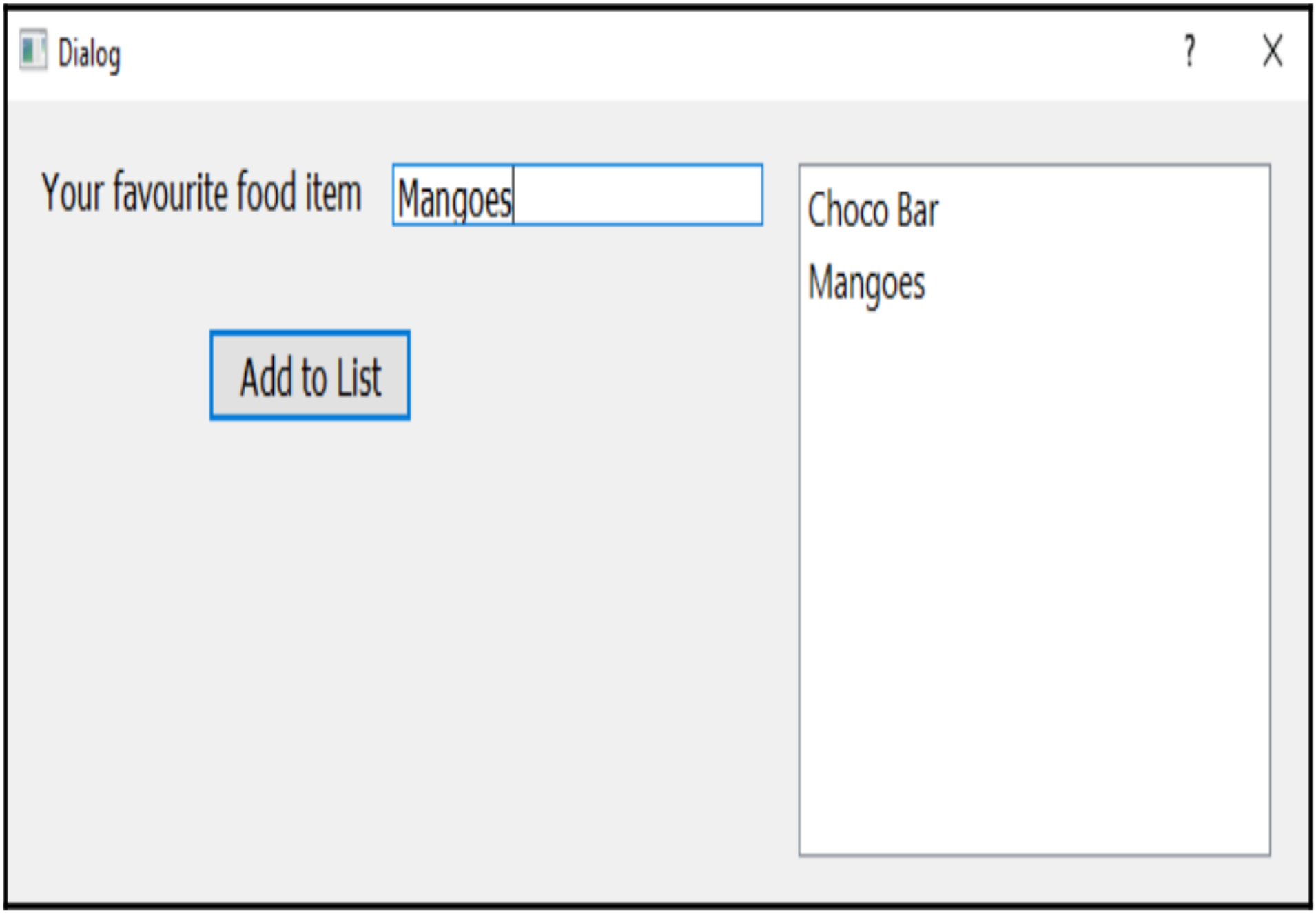

Dialog
?
Diagnosis Tests
Urine Analaysis $5
Chest X Ray 100$
Sugar Level test $3
Hemoglobin test $7
Thyroid Stimulating Harmone test $10
Selected tests are
Chest X Ray 100$
Hemoglobin test $7

Bài tập: tạo GUI thêm item vào List



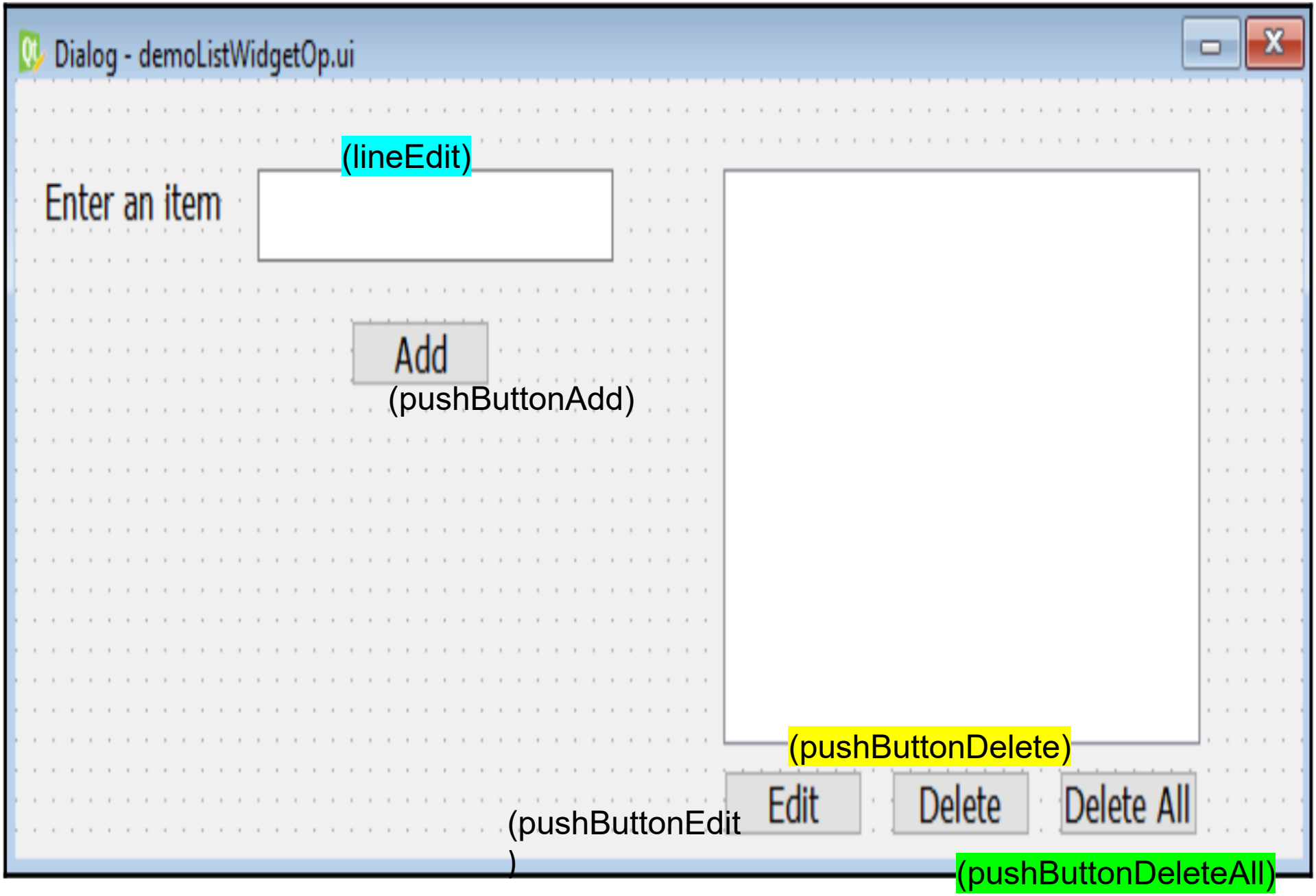
Gợi ý code nhấn nút sẽ thêm item và List

```
class MyForm(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.ui.pushButtonAdd.clicked.connect(se
lf.addlist)
        self.show()

    def addlist(self):
        self.ui.listWidgetSelectedItems.addItem(
                self.ui.lineEditFoodItem.text())
        self.ui.lineEditFoodItem.setText('')
        self.ui.lineEditFoodItem.setFocus()
```


Dialog
?
X
Your favourite food item
Mangoes
Add to List
Choco Bar
Mangoes

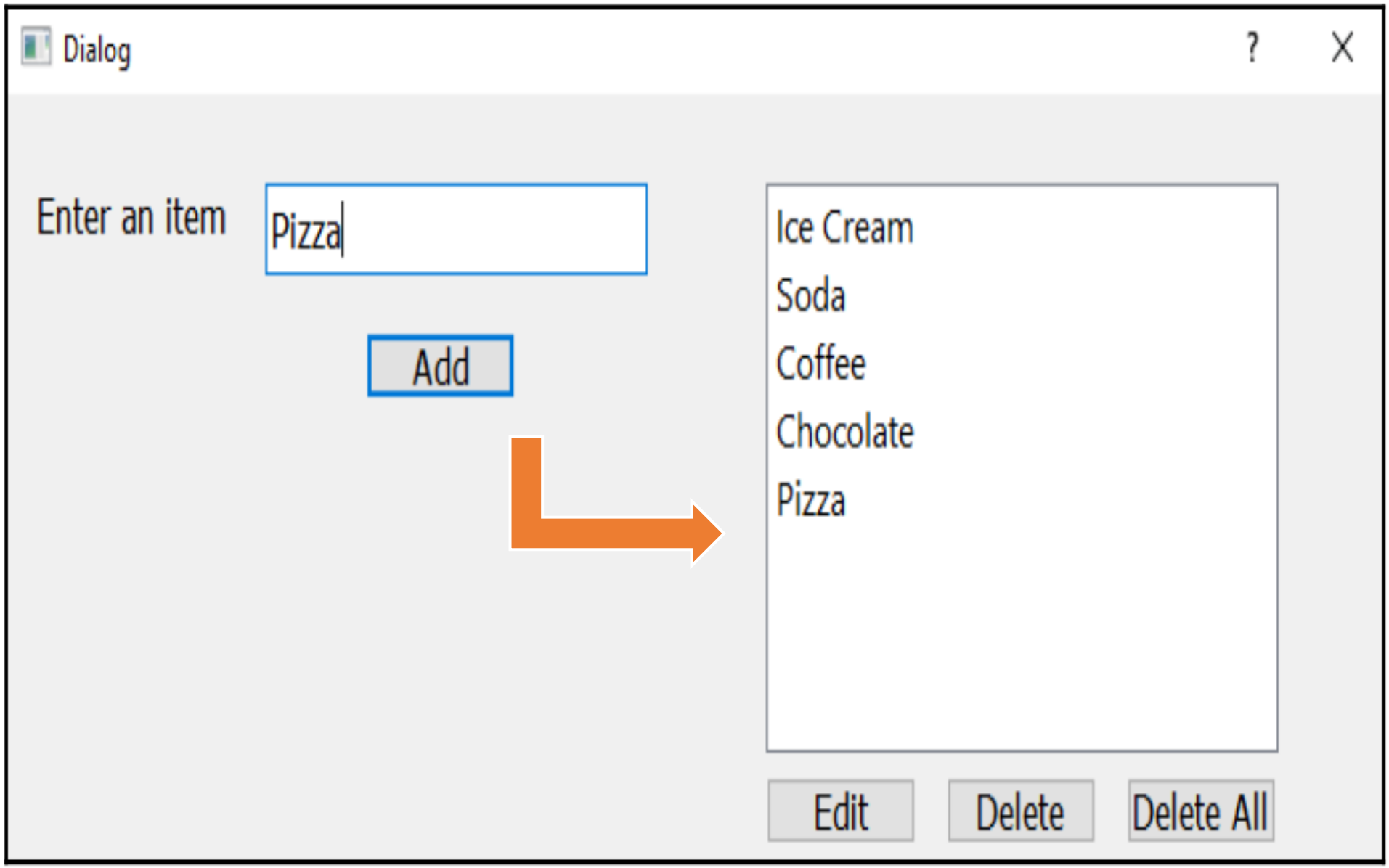
Các thao tác trong List widget

- List Widget gồm nhiều **list item** (đối tượng QListWidgetItem).
- Có thể **thêm item** vào List bằng addItem() hoặc insertItem().
- Mỗi item có thể là **văn bản hoặc biểu tượng**, có thể **được chọn (check)** hoặc không.
- QListWidgetItem hỗ trợ các thao tác như: setText(), checkState(), setHidden(), isHidden().

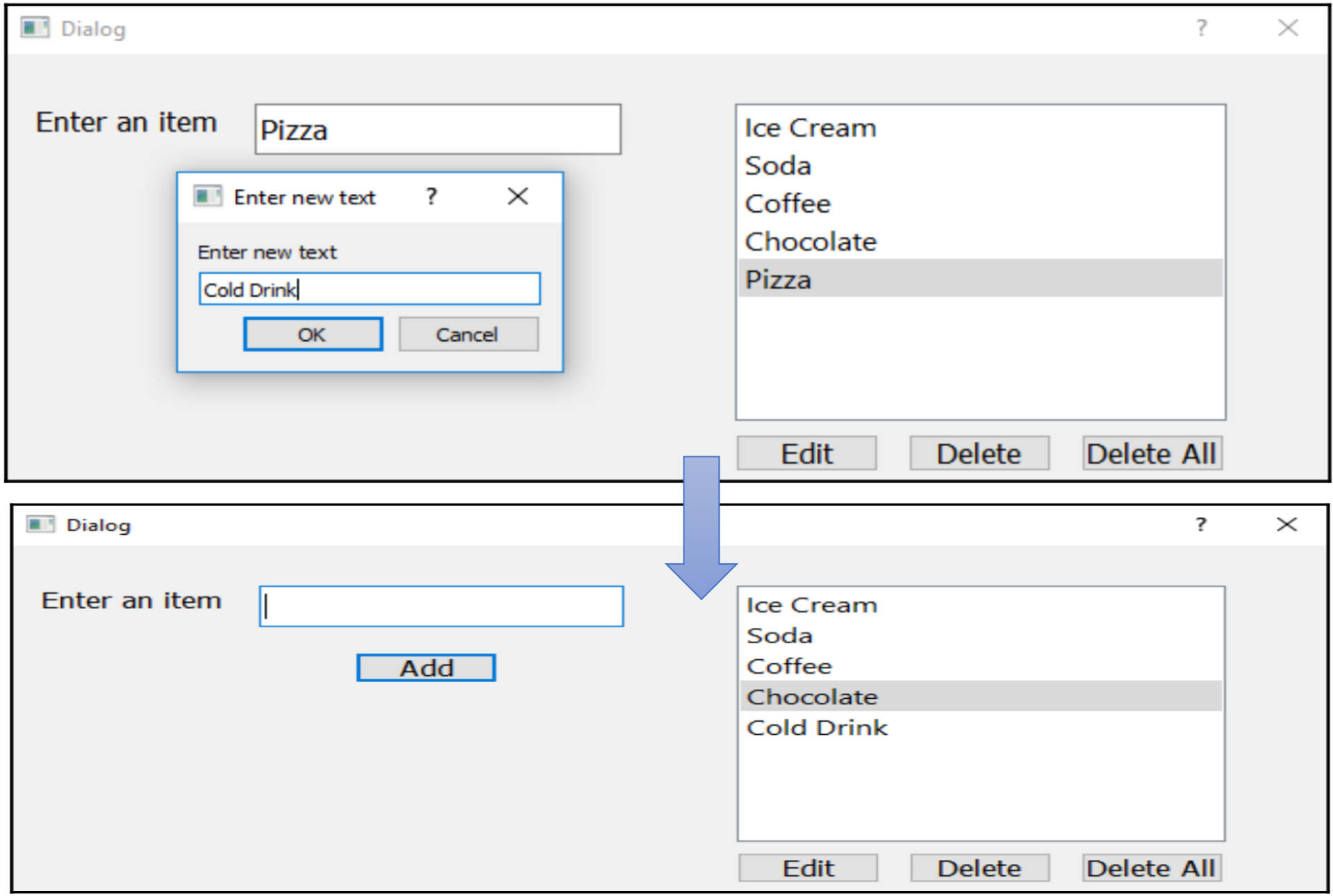

Dialog - demoListWidgetOp.ui
(lineEdit)
Enter an item
Add
(pushButtonAdd)
(pushButtonDelete)
Edit
Delete
Delete All
(pushButtonEdit
)
(pushButtonDeleteAll)

```
def addlist(self):
  self.ui.listWidget.addItem(self.ui.lineEdit.text())
  self.ui.lineEdit.setText('')
  self.ui.lineEdit.setFocus()

def editlist(self):
  row=self.ui.listWidget.currentRow()
  newtext, ok=QInputDialog.getText(self,"Enter new text“,

                          “nhap gia tri”)
  if ok and (len(newtext) !=0):
    self.ui.listWidget.takeItem(

      self.ui.listWidget.currentRow())
    self.ui.listWidget.insertItem(row,

      QListWidgetItem(newtext))
def delitem(self):
  self.ui.listWidget.takeItem(

      self.ui.listWidget.currentRow())
```

Add item vào List



Edit item trong List



Sử dụng Combo Box widget

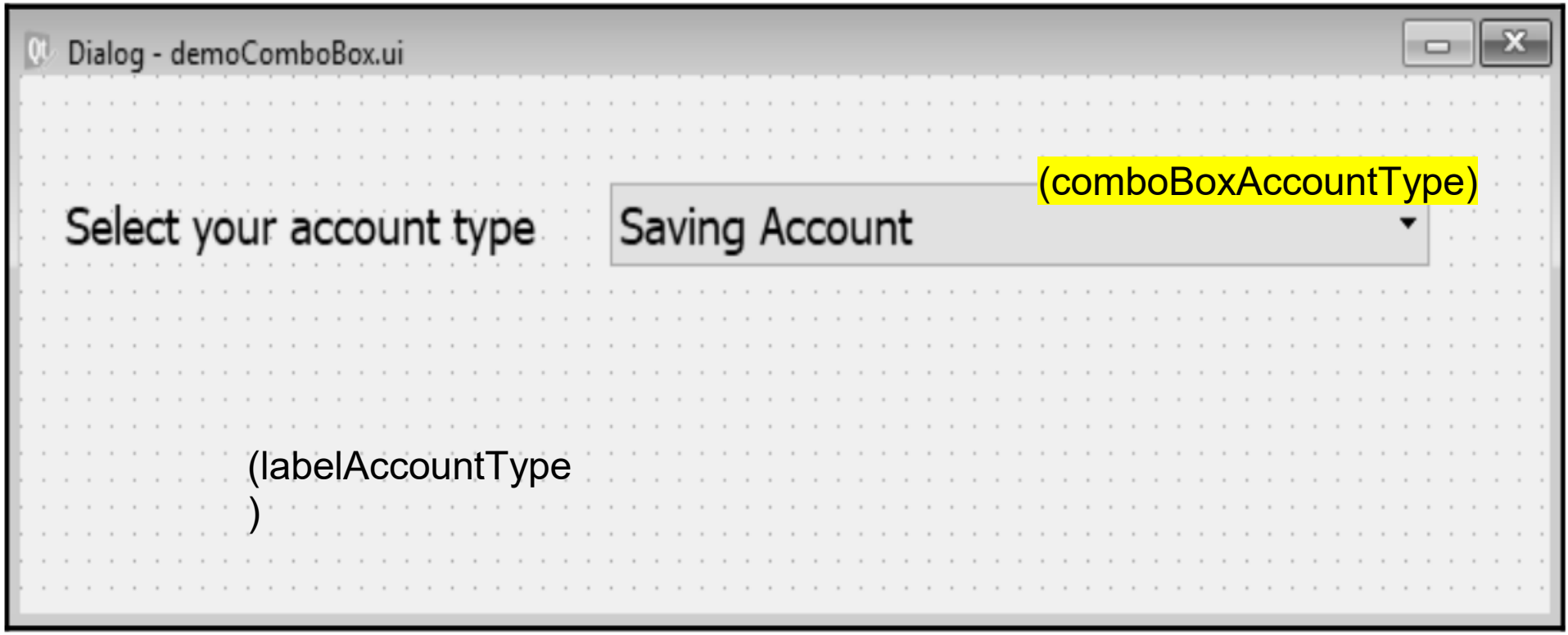
- **ComboBox** dùng để nhận đầu vào, cho phép **chọn 1 mục duy nhất**.
- **Tiết kiệm không gian** hơn List widget.
- Dùng lớp **QComboBox**, có thể hiển thị **văn bản hoặc hình ảnh (pixmap)**.
- Các hàm thường dùng: **addItem(s), removeItem, clear, currentText, currentIndex, count, setEditable,…**

Signal được tạo bởi combo box

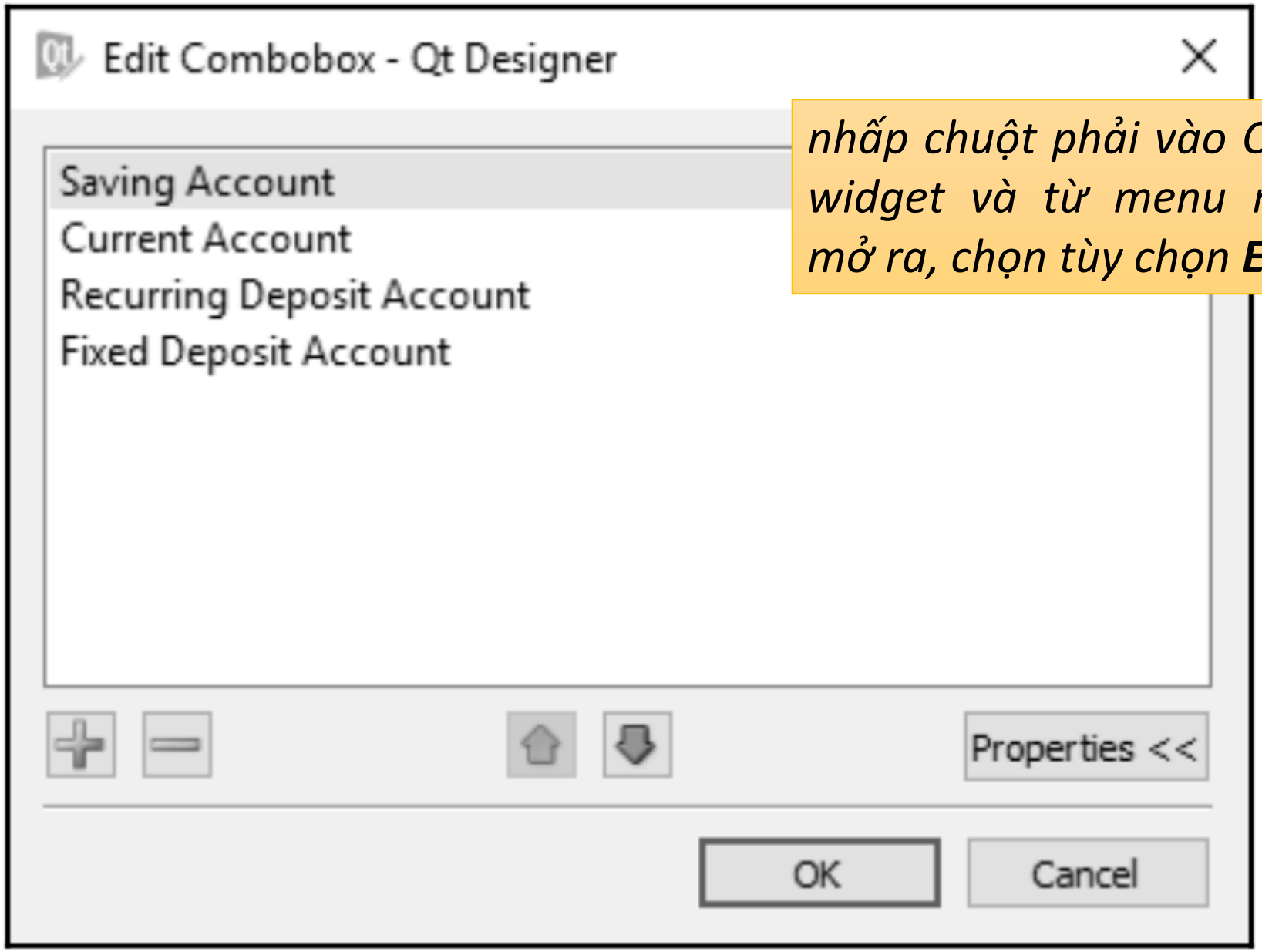
currentIndexChanged()	Phát ra khi chỉ số của combo box được thay đổi, nghĩa là người dùng chọn một số mục mới trong combo box.
activated()	Phát ra khi chỉ số được thay đổi bởi người dùng.
highlighted()	Phát ra khi người dùng đánh dấu một mục trong hộp tổ hợp
editTextChanged()	Phát ra khi văn bản của hộp tổ hợp có thể chỉnh sửa được thay đổi

Áp dụng sử dụng Combo Box

Xây dựng chương trình hiển thị một số loại tài khoản ngân hàng nhất định thông qua combo box và sẽ nhắc người dùng chọn loại tài khoản ngân hàng mà họ muốn mở. Loại tài khoản ngân hàng được chọn từ combo box sẽ được hiển thị trên màn hình thông qua Label widget.



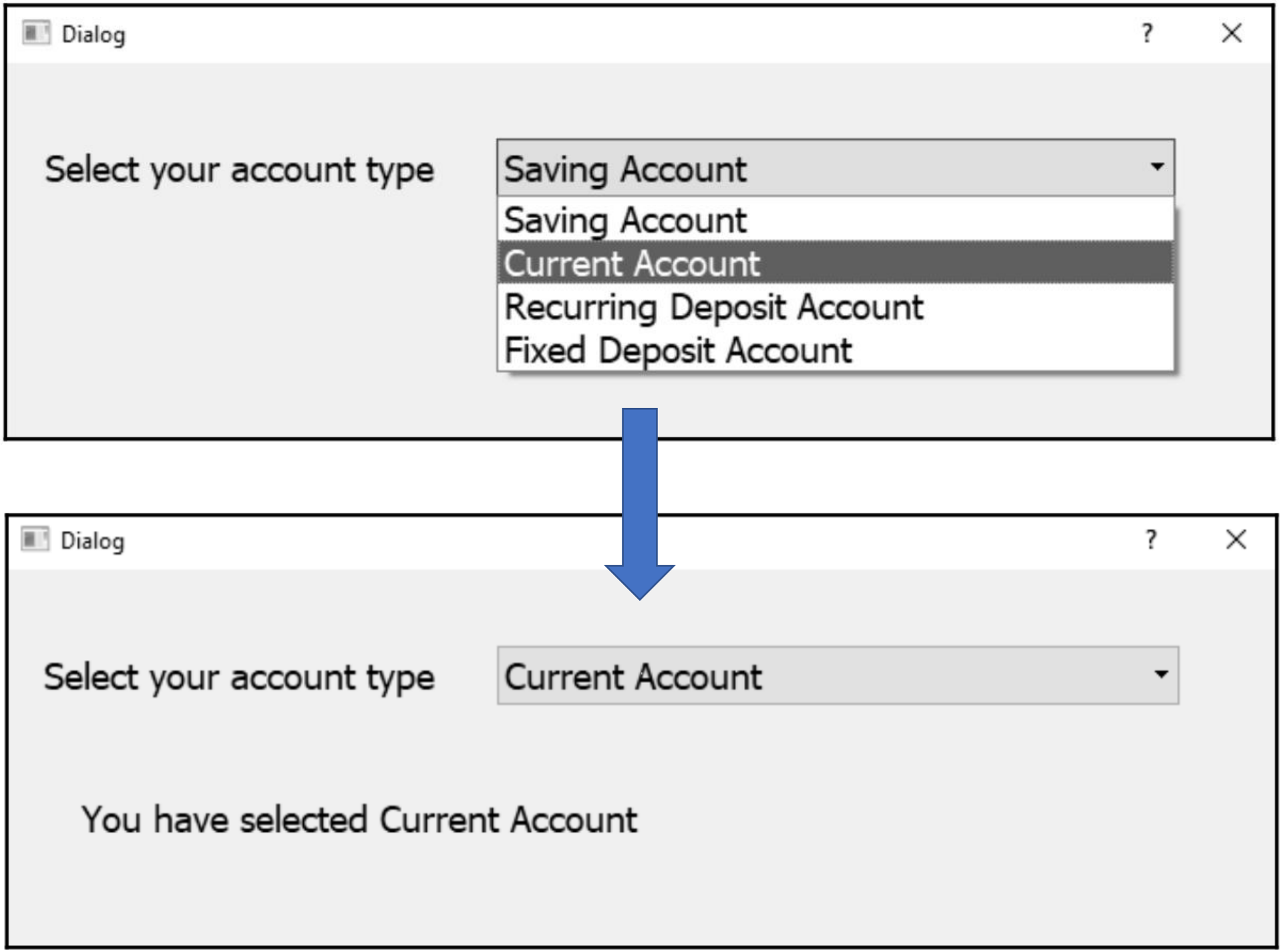
Thêm một số item vào Combo Box



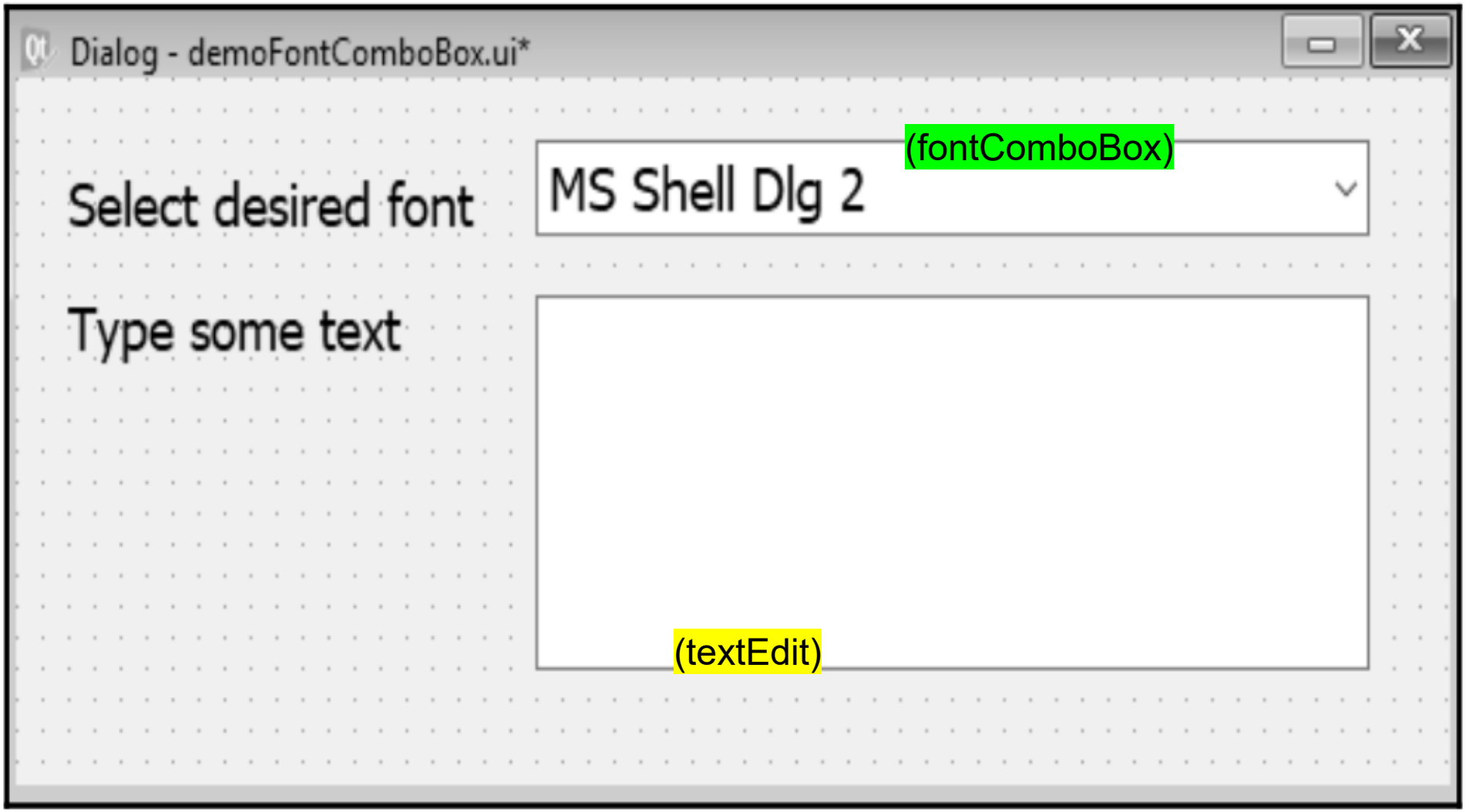
nhấp chuột phải vào Combobox widget và từ menu ngữ cảnh mở ra, chọn tùy chọn ***Edit Items***

Gợi ý code thao tác trên combo box

```
class MyForm(QDialog):
  def __init__(self):
    super().__init__()
    self.ui = Ui_Dialog()
    self.ui.setupUi(self)
    self.ui.comboBoxAccountType.currentIndexChanged.connect(
                                                  self.dispAccountType)
    self.show()

  def dispAccountType(self):
    self.ui.labelAccountType.setText("You have selected " +
                    self.ui.comboBoxAccountType.itemText(
                                        self.ui.comboBoxAccountType.
                                                    currentIndex()))
```

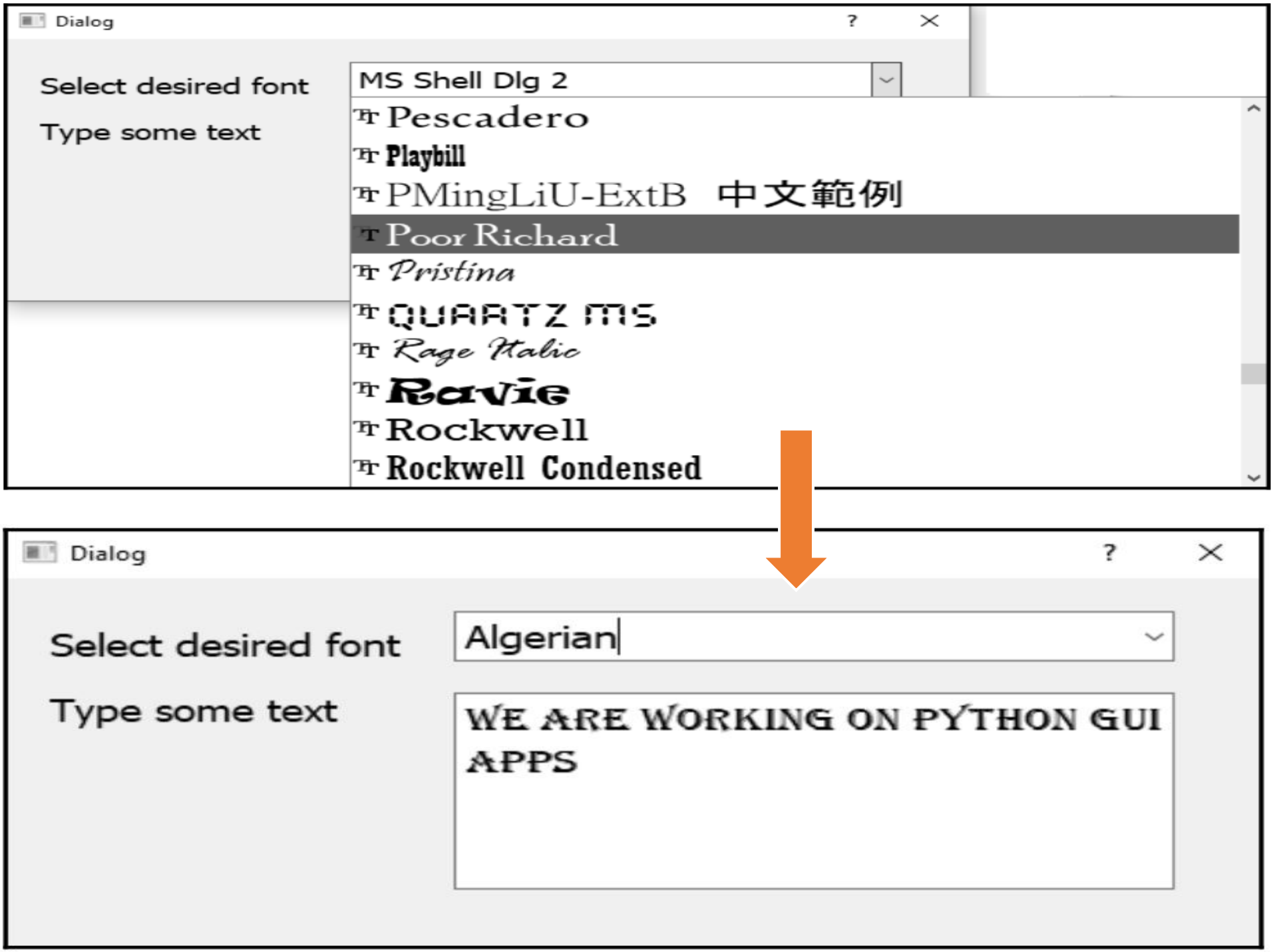

Dialog
Select your account type
Saving Account
Saving Account
Current Account
Recurring Deposit Account
Fixed Deposit Account
Dialog
Select your account type
Current Account
You have selected Current Account

Bài tập: tạo Combo box chọn Font



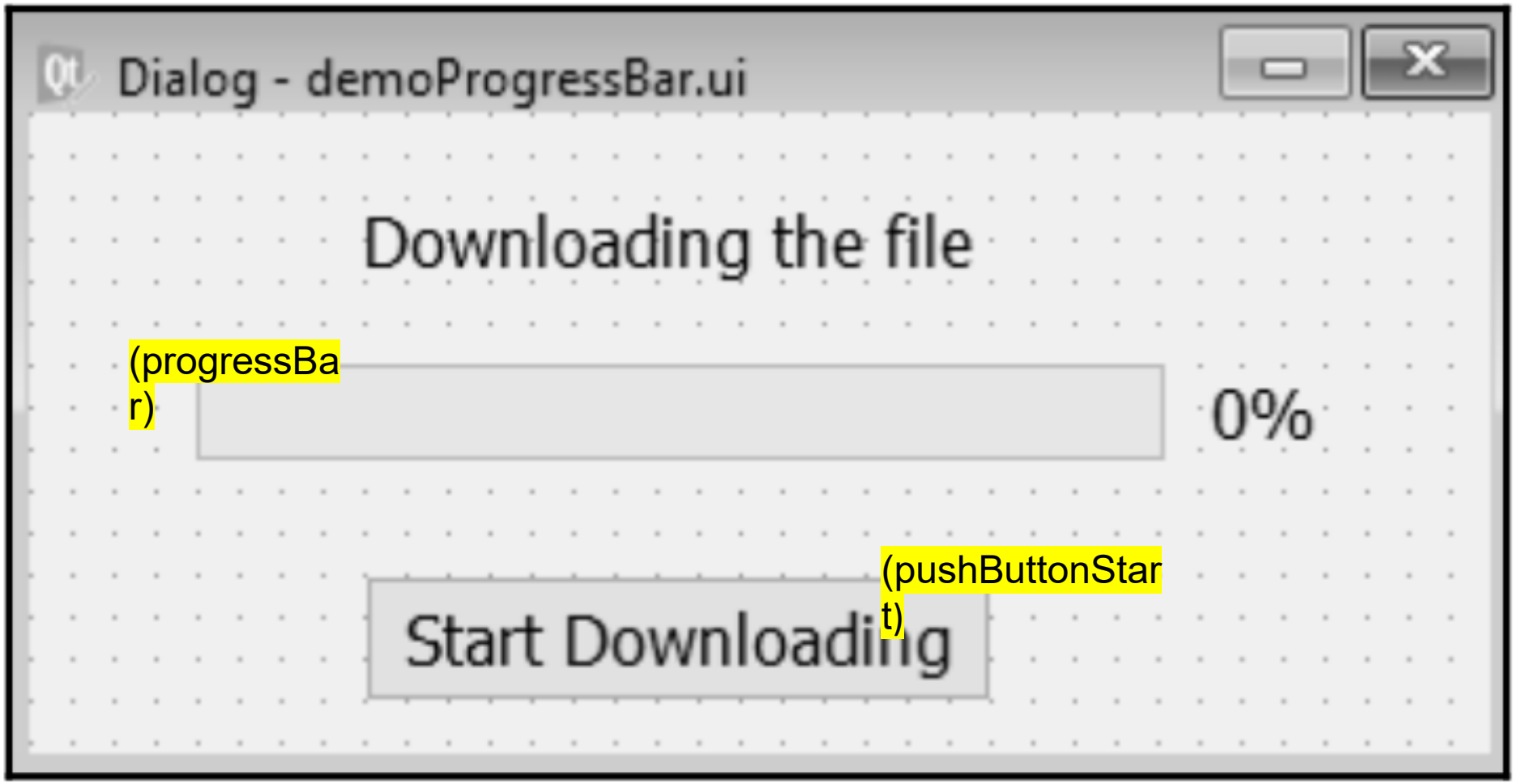
Người dùng nhập nội dung mong muốn trong Text Edit, sau đó người dùng chọn bất kỳ kiểu phông chữ từ Font Combo Box, thì font được chọn sẽ được áp dụng cho các nội dung đã gõ vào Text Edit.

```
class MyForm(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        myFont = QtGui.QFont(
            self.ui.fontComboBox.itemText(self.
             ui.fontComboBox.currentIndex()), 15)
        self.ui.textEdit.setFont(myFont)
        self.ui.fontComboBox.currentFontChanged
                            .connect(self.changeFont)
        self.show()

    def changeFont(self):
        myFont = QtGui.QFont(
            self.ui.fontComboBox.itemText(self.
             ui.fontComboBox.currentIndex()), 15)
        self.ui.textEdit.setFont(myFont)
```

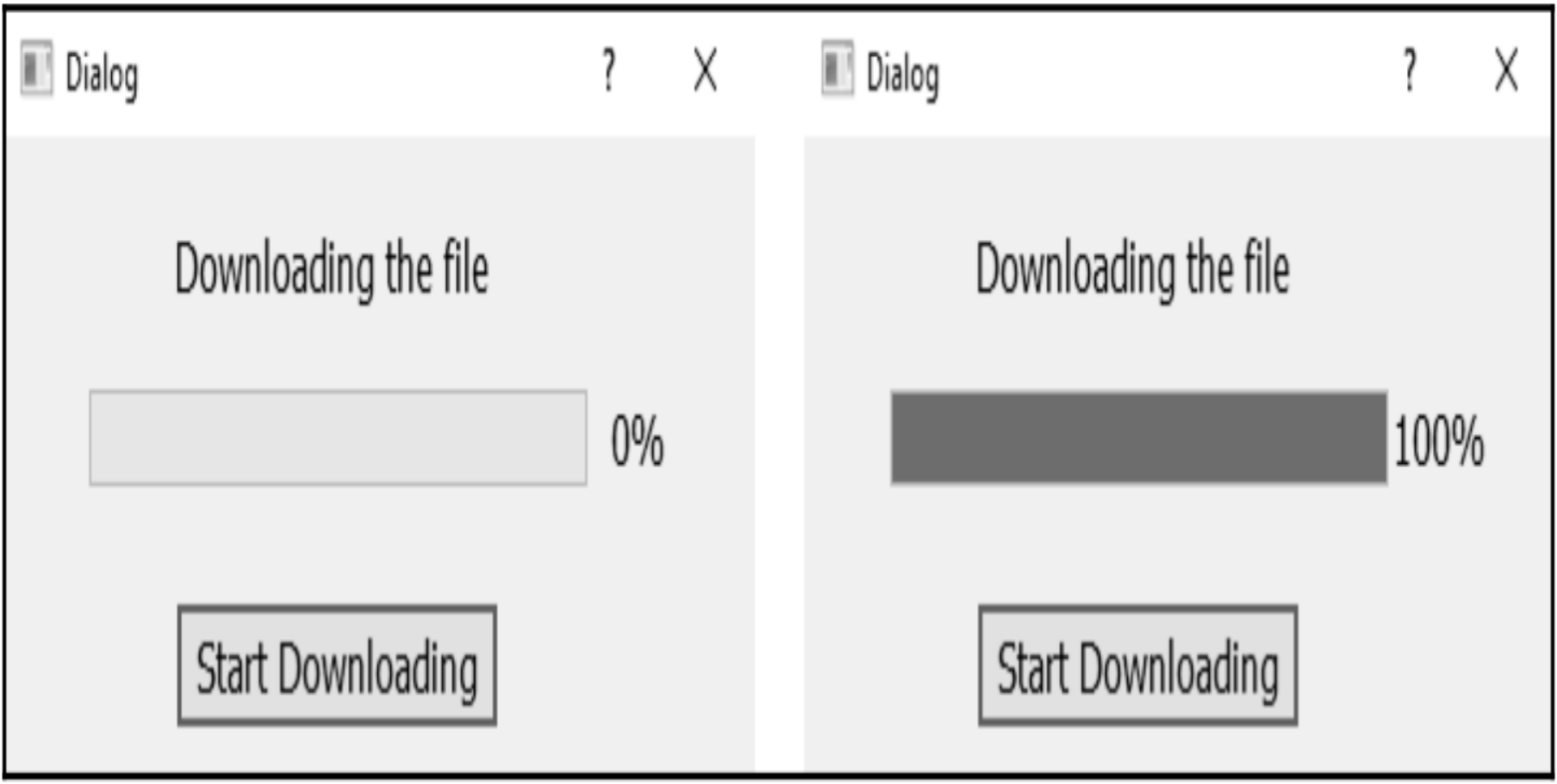

Dialog
Select desired font
MS Shell Dlg 2
Type some text
Pescadero
Playbill
PMingLiU-ExtB 中文範例
Poor Richard
Pristina
QUARTZ MS
Rage Italic
Ravie
Rockwell
Rockwell Condensed
Dialog
Select desired font
Algerian
Type some text
WE ARE WORKING ON PYTHON GUI APPS

Sử dụng **Progress Bar** widget

Progress Bar (thanh tiến trình) dùng để hiển thị mức độ hoàn thành của một tác vụ. Khi tải xuống tệp, thanh tiến trình cập nhật từ 0% đến 100% theo quá trình tải và hiển thị 100% khi tác vụ hoàn tất, giúp người dùng dễ theo dõi tiến độ.


Dialog - demoProgressBar.ui
Downloading the file
(progressBa
r)
0%
(pushButtonStar
t)
Start Downloading

```
class MyForm(QDialog):
  def __init__(self):
    super().__init__()
    self.ui = Ui_Dialog()
    self.ui.setupUi(self)
    self.ui.pushButtonStart.clicked.
                        connect(self.updateBar)
    self.show()

  def updateBar(self):
    x = 0
    while x < 100:
      x += 0.0001
      self.ui.progressBar.setValue(x)
```


Dialog
?
X
Downloading the file
0%
Start Downloading
Dialog
?
X
Downloading the file
100%
Start Downloading