**Exercise 1: "Hello Qt!" - Your First Window**

**Goal:** Create a basic Qt Widgets application that displays a simple window with a text label.

**Steps:**

1. **Create a New Project:**
   * Open Qt Creator.
   * Go to File > New File or Project....
   * Select Application > Qt Widgets Application.
   * Choose a project name (e.g., HelloQt) and a location.
   * Keep the default options for Build System (qmake) and Base class (QWidget).
2. **Design the UI (using Designer):**
   * In the Projects pane, double-click on mainwindow.ui (or widget.ui if you chose QWidget as base class). This will open Qt Designer.
   * From the Widget Box (usually on the left), drag a Label widget onto your main window.
   * Double-click the label and change its text to "Hello, Qt!".
   * You can resize the window and the label as needed.
   * Save the mainwindow.ui file (Ctrl+S or Cmd+S).
3. **Run the Application:**
   * Click the green "Run" button (or Ctrl+R / Cmd+R).
   * Observe your simple "Hello, Qt!" window.

**Learning Points:**

* Navigating Qt Creator's interface.
* Creating a new Qt Widgets project.
* Using Qt Designer to add and modify widgets.
* Compiling and running a basic application.

**Exercise 2: Simple Counter Application**

**Goal:** Create an application with a label to display a number, and two buttons to increment and decrement that number.

**Steps:**

1. **Create a New Project:**
   * Start a new Qt Widgets Application project (e.g., SimpleCounter).
   * Choose QWidget as the base class for simplicity.
2. **Design the UI:**
   * Open widget.ui in Qt Designer.
   * Drag a Label widget onto the window. Name it (in the Property Editor) countLabel. Set its initial text to "0". You might want to make the font size larger for visibility (find font in Property Editor).
   * Drag two Push Button widgets onto the window.
   * Change the text of one button to "Increment" and its object name to incrementButton.
   * Change the text of the other button to "Decrement" and its object name to decrementButton.
   * Arrange them neatly (e.g., label at top, buttons below).
   * Save widget.ui.
3. **Implement Logic (Connecting Signals and Slots):**
   * Right-click on the incrementButton in Qt Designer and select Go to Slot....
   * Choose clicked(). This will automatically generate a slot function in your widget.cpp file.
   * Repeat for the decrementButton, choosing clicked().
   * In widget.h, declare a private integer member variable to store the count: int m\_count;.
   * In widget.cpp:
     + In the constructor Widget::Widget(QWidget \*parent):
       - Initialize m\_count = 0;.
       - Update the label: ui->countLabel->setText(QString::number(m\_count));
     + In the on\_incrementButton\_clicked() slot:
       - m\_count++;
       - ui->countLabel->setText(QString::number(m\_count));
     + In the on\_decrementButton\_clicked() slot:
       - m\_count--;
       - ui->countLabel->setText(QString::number(m\_count));
4. **Run the Application:**
   * Test your counter.

**Learning Points:**

* Understanding the concept of "signals and slots" for event handling.
* Connecting buttons to custom code.
* Modifying widget properties (text, font size) programmatically.
* Using QString::number() for converting integers to strings.

**Exercise 3: Basic Calculator Layout**

**Goal:** Create the UI layout for a simple calculator, without implementing the actual calculation logic. Focus on using layout managers.

**Steps:**

1. **Create a New Project:**
   * Start a new Qt Widgets Application project (e.g., CalculatorLayout).
   * Choose QWidget as the base class.
2. **Design the UI (Focus on Layouts):**
   * Open widget.ui.
   * Drag a Line Edit widget onto the window (this will be for displaying the input/result). Name it displayLineEdit.
   * Drag a Grid Layout (from Layouts in the Widget Box) onto the main window.
   * Drag a Vertical Layout (from Layouts) onto the main window.
   * **Crucially:** Select the main Widget (the top-level window) in the Object Inspector (usually top-right). Then, click one of the layout buttons in the top toolbar of Designer (e.g., Lay Out Vertically or Lay Out in a Grid) to apply a top-level layout to your main window. For a calculator, a Vertical Layout for the overall structure (display + buttons) usually works well, with a Grid Layout *inside* it for the number/operator buttons.
   * **Populating the Grid:** Inside the Grid Layout, drag Push Button widgets to represent numbers (0-9) and basic operators (+, -, \*, /). You'll need about 16 buttons for a basic 4x4 layout.
   * Set the text for each button (e.g., "7", "8", "9", "+", "4", "5", "6", "-", etc.).
   * Experiment with stretching and spacing properties of the layouts.
   * Save widget.ui.
3. **Run the Application:**
   * Observe how your buttons and display resize correctly when you resize the window, thanks to the layout managers.

**Learning Points:**

* Mastering Qt's layout managers (QVBoxLayout, QHBoxLayout, QGridLayout).
* Creating complex UI arrangements efficiently.
* Understanding how widgets scale and position themselves within layouts.
* The importance of Lay Out Vertically/Horizontally/in a Grid buttons in Designer.

**Exercise 4: Simple To-Do List Application**

**Goal:** Create a basic To-Do list application where users can add and remove items.

**Steps:**

1. **Create a New Project:**
   * New Qt Widgets Application project (e.g., ToDoListApp).
   * Choose QWidget as the base class.
2. **Design the UI:**
   * Open widget.ui.
   * Drag a Line Edit widget (for new tasks input). Name it taskLineEdit.
   * Drag a Push Button widget (for adding tasks). Name it addButton, text "Add Task".
   * Drag a List Widget (from Item Views in Widget Box) onto the window. Name it taskListWidget.
   * Drag another Push Button (for removing tasks). Name it removeButton, text "Remove Selected".
   * Arrange these widgets using appropriate layouts (e.g., QVBoxLayout for the overall window, QHBoxLayout for the input field and add button).
   * Save widget.ui.
3. **Implement Logic:**
   * **Add Task:**
     + Right-click addButton in Designer, Go to Slot... > clicked().
     + In on\_addButton\_clicked():
       - Get text from taskLineEdit: QString task = ui->taskLineEdit->text();
       - Check if text is not empty: if (!task.isEmpty()) { ... }
       - Add item to taskListWidget: ui->taskListWidget->addItem(task);
       - Clear taskLineEdit: ui->taskLineEdit->clear();
   * **Remove Task:**
     + Right-click removeButton in Designer, Go to Slot... > clicked().
     + In on\_removeButton\_clicked():
       - Get current selected row: int currentRow = ui->taskListWidget->currentRow();
       - Check if an item is selected: if (currentRow != -1) { ... }
       - Remove item: ui->taskListWidget->takeItem(currentRow); (This takes ownership of the item, so if you don't store it, it will be deleted by Qt's memory management).
4. **Run the Application:**
   * Test adding and removing tasks.

**Learning Points:**

* Working with QLineEdit for user input.
* Adding and removing items from a QListWidget.
* Handling user selections in list-based widgets.
* Basic input validation (checking for empty text).