Let's dive into detailed, step-by-step instructions for each of those Qt Creator exercises. This will walk you through everything from project creation to running your application.

**Before You Start (Important!):**

* **Install Qt Creator and Qt:** Make sure you have Qt Creator installed, along with a Qt version (e.g., Qt 6.x or Qt 5.x) and a compatible compiler (like MinGW for Windows, GCC for Linux, or Clang for macOS). If you haven't, download the Qt Online Installer from the official Qt website (<https://www.qt.io/download>).

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

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

**Difficulty:** Easy

**Detailed Steps:**

1. **Open Qt Creator:**
   * Find the Qt Creator icon on your desktop or in your applications folder and launch it.
2. **Create a New Project:**
   * In Qt Creator, go to the menu bar at the top and select File > New File or Project....
   * In the "New File or Project" wizard:
     + Under Projects, select Application.
     + Then, choose Qt Widgets Application.
     + Click the Choose... button.
3. **Configure Project Details:**
   * **Name:** Enter HelloQt (or any other descriptive name) in the "Name" field.
   * **Create in:** Click the Browse... button to select a folder on your computer where you want to save your project. Choose a clear, accessible location (e.g., Documents/QtProjects).
   * **Build system:** Leave this as qmake (the default and common build system for Qt projects).
   * Click Next.
4. **Define Class Information:**
   * **Class name:** Leave MainWindow as the default.
   * **Base class:** Ensure QWidget is selected. This tells Qt that your main window will inherit from the basic QWidget class. (If you wanted menus and toolbars, you'd choose QMainWindow, but QWidget is simpler for this exercise).
   * Leave "Generate form" checked. This will create the .ui file for visual design.
   * Click Next.
5. **Select Translation File (Optional, Skip for now):**
   * Just click Next on this step. We won't be localizing the app yet.
6. **Kit Selection:**
   * Ensure that at least one "Kit" is selected. A Kit represents a Qt version + compiler combination. If you only have one installed, it will likely be pre-selected. If you have multiple, choose one that corresponds to your installed Qt version (e.g., "Desktop Qt 6.x.x MinGW 64-bit").
   * Click Next.
7. **Project Management (Optional):**
   * If you're using a version control system like Git, you can initialize a repository here. For this exercise, leave "None" selected.
   * Click Finish.
8. **Explore the Project Structure:**
   * Qt Creator will now open your new project. In the "Projects" pane (usually on the left side), you'll see files like:
     + HelloQt.pro: The project file (qmake configuration).
     + main.cpp: The entry point of your application.
     + mainwindow.h: Header file for your MainWindow class.
     + mainwindow.cpp: Source file for your MainWindow class.
     + mainwindow.ui: The UI definition file (what we'll be editing visually).
9. **Design the UI (using Qt Designer):**
   * In the "Projects" pane, double-click on mainwindow.ui. This will open the Qt Designer interface within Qt Creator.
   * **Add a Label:**
     + On the left side, you'll see the "Widget Box" (a list of available widgets).
     + Scroll down or type "Label" in the search bar.
     + Click and drag a Label widget from the "Widget Box" onto your main window (the grey rectangle in the central area).
   * **Change Label Text:**
     + Double-click on the Label widget you just placed. A text cursor will appear.
     + Type Hello, Qt!
     + Press Enter or click outside the label.
   * **Adjust Size (Optional):**
     + You can click and drag the corners of the main window or the label to resize them.
     + To ensure the label text is fully visible, make the window a bit larger.
   * **Save the UI:**
     + Go to File > Save All (or Ctrl+S / Cmd+S). This saves the changes to mainwindow.ui.
10. **Run the Application:**
    * Look for the green "Run" button in the bottom-left of Qt Creator (it looks like a play arrow).
    * Click the "Run" button (or press Ctrl+R / Cmd+R).
    * Qt Creator will compile your project. If there are no errors, a new window will pop up displaying "Hello, Qt!".

**Congratulations! You've successfully created and run your first Qt Widgets 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.

**Difficulty:** Easy

**Detailed Steps:**

1. **Create a New Project:**
   * File > New File or Project... > Application > Qt Widgets Application > Choose...
   * **Name:** SimpleCounter
   * **Create in:** Choose your projects folder.
   * Click Next.
   * **Class name:** Widget (default)
   * **Base class:** QWidget (default)
   * Leave "Generate form" checked.
   * Click Next.
   * Skip translation file: Next.
   * Select your Kit: Next.
   * Finish.
2. **Design the UI (using Qt Designer):**
   * Double-click on widget.ui in the "Projects" pane.
   * **Add a Label for the Count:**
     + Drag a Label from the "Widget Box" onto the form.
     + In the "Property Editor" (usually on the right side), find the objectName property and change it to countLabel.
     + Double-click the label on the form and change its initial text to 0.
     + **Make text larger (Optional but recommended):** In the "Property Editor" for countLabel, find the font property (it has a + next to it). Click the ... button next to Font. In the dialog, choose a larger Point Size (e.g., 24 or 36) and maybe Bold. Click OK.
   * **Add "Increment" Button:**
     + Drag a Push Button onto the form.
     + In "Property Editor", change its objectName to incrementButton.
     + Double-click the button on the form and change its text to Increment.
   * **Add "Decrement" Button:**
     + Drag another Push Button onto the form.
     + In "Property Editor", change its objectName to decrementButton.
     + Double-click the button on the form and change its text to Decrement.
   * **Arrange Widgets (Optional, but good practice):**
     + You can drag the widgets around to arrange them nicely (e.g., label at the top, buttons below). For now, simple manual arrangement is fine.
   * **Save the UI:** File > Save All (Ctrl+S / Cmd+S).
3. **Implement Logic (Connecting Signals and Slots):**
   * **Open widget.h:**
     + In the "Projects" pane, double-click on widget.h.
     + Inside the private: section of the Widget class, add a new member variable to store the count:

C++

private:

Ui::Widget \*ui;

int m\_count; // Our counter variable

* + - Save widget.h.
  + **Open widget.cpp:**
    - Double-click on widget.cpp in the "Projects" pane.
    - **Initialize m\_count in the constructor:**
      * Inside the Widget::Widget(QWidget \*parent) constructor, *after* ui->setupUi(this);, add:

C++

Widget::Widget(QWidget \*parent)

: QWidget(parent)

, ui(new Ui::Widget)

{

ui->setupUi(this);

m\_count = 0; // Initialize the counter

ui->countLabel->setText(QString::number(m\_count)); // Display initial count

}

* + - * *Self-correction:* Make sure you include <QString> for QString::number() if it's not already included. It usually is, but good to check.
    - **Create the on\_incrementButton\_clicked() slot:**
      * Go back to widget.ui (the Designer).
      * Right-click on the "Increment" button (incrementButton).
      * Select Go to Slot....
      * In the "Go to Slot" dialog, select clicked() and click OK.
      * Qt Creator will automatically generate a new empty function on\_incrementButton\_clicked() in your widget.cpp file and add its declaration to widget.h.
      * Implement the logic inside this new function:

C++

void Widget::on\_incrementButton\_clicked()

{

m\_count++; // Increment the counter

ui->countLabel->setText(QString::number(m\_count)); // Update the label

}

* + - **Create the on\_decrementButton\_clicked() slot:**
      * Go back to widget.ui (the Designer).
      * Right-click on the "Decrement" button (decrementButton).
      * Select Go to Slot....
      * In the "Go to Slot" dialog, select clicked() and click OK.
      * Implement the logic inside this new function:

C++

void Widget::on\_decrementButton\_clicked()

{

m\_count--; // Decrement the counter

ui->countLabel->setText(QString::number(m\_count)); // Update the label

}

* + - Save widget.cpp.

1. **Run the Application:**
   * Click the green "Run" button (or Ctrl+R / Cmd+R).
   * Your counter application will appear. Click "Increment" and "Decrement" to test it.

**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.

**Difficulty:** Medium

**Detailed Steps:**

1. **Create a New Project:**
   * File > New File or Project... > Application > Qt Widgets Application > Choose...
   * **Name:** CalculatorLayout
   * **Create in:** Choose your projects folder.
   * Click Next.
   * **Class name:** Widget (default)
   * **Base class:** QWidget (default)
   * Leave "Generate form" checked.
   * Click Next.
   * Skip translation file: Next.
   * Select your Kit: Next.
   * Finish.
2. **Design the UI (Focus on Layouts):**
   * Double-click on widget.ui.
   * **Add a Display Line Edit:**
     + Drag a Line Edit from the "Widget Box" onto the form.
     + In the "Property Editor", change its objectName to displayLineEdit.
     + **Set Placeholder Text (Optional):** In "Property Editor", find placeholderText and enter 0 or Enter numbers....
     + **Align Text (Optional):** In "Property Editor", find alignment and choose AlignRight.
     + **Make Read-Only (Optional):** In "Property Editor", set readOnly to true. This prevents users from typing directly into it, as it's meant to be output.
   * **Add a Grid Layout for Buttons:**
     + From the "Widget Box", find Layouts and drag a Grid Layout onto your form. Make it large enough to hold about 16 buttons.
   * **Populate the Grid with Buttons (4x4 layout):**
     + Drag Push Button widgets into the Grid Layout. As you drag them, a blue outline will show you where they'll be placed within the grid.
     + Place 16 buttons to form a 4x4 grid.
     + **Set Button Text:** Double-click each button and set its text according to a typical calculator layout:
       - Row 1: 7, 8, 9, /
       - Row 2: 4, 5, 6, \*
       - Row 3: 1, 2, 3, -
       - Row 4: 0, ., =, +
     + **Name Buttons (Optional but good practice for later logic):** Give each button a descriptive objectName (e.g., button7, buttonPlus, buttonEquals).
   * **Apply a Top-Level Layout to the Main Window:**
     + This is the most crucial step for responsive design!
     + In the "Object Inspector" (usually top-right, shows a tree of your widgets), click on the top-most item, which will be named Widget (your main window).
     + Now, look at the toolbar at the top of Qt Designer (above your form). You'll see icons for layouts: "Lay Out Horizontally", "Lay Out Vertically", "Lay Out in a Grid", etc.
     + Click the "Lay Out Vertically" icon (it looks like a stack of horizontal lines). This will arrange your displayLineEdit and your Grid Layout vertically.
     + You'll see the displayLineEdit at the top, and the Grid Layout below it, both now managed by a top-level QVBoxLayout.
   * **Adjust Spacing and Stretching (Optional):**
     + You can select the main Widget in the Object Inspector and then examine the layout properties in the "Property Editor" to adjust spacing.
     + You can also select individual widgets within the layout and change their sizePolicy or stretch properties if you want them to expand differently. For now, the default behavior should be fine.
   * **Save the UI:** File > Save All (Ctrl+S / Cmd+S).
3. **Run the Application:**
   * Click the green "Run" button (or Ctrl+R / Cmd+R).
   * Observe your calculator layout. Try resizing the window – notice how the buttons and display adjust their size and position automatically due to the layout managers.

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

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

**Difficulty:** Medium

**Detailed Steps:**

1. **Create a New Project:**
   * File > New File or Project... > Application > Qt Widgets Application > Choose...
   * **Name:** ToDoListApp
   * **Create in:** Choose your projects folder.
   * Click Next.
   * **Class name:** Widget (default)
   * **Base class:** QWidget (default)
   * Leave "Generate form" checked.
   * Click Next.
   * Skip translation file: Next.
   * Select your Kit: Next.
   * Finish.
2. **Design the UI (using Qt Designer):**
   * Double-click on widget.ui.
   * **Add a Line Edit for input:**
     + Drag a Line Edit onto the form.
     + Set its objectName to taskLineEdit.
     + Set its placeholderText to Enter a new task....
   * **Add an "Add Task" Push Button:**
     + Drag a Push Button onto the form.
     + Set its objectName to addButton.
     + Set its text to Add Task.
   * **Group input and add button horizontally:**
     + Select both the taskLineEdit and the addButton (click one, then Ctrl or Cmd click the other).
     + Right-click on one of the selected widgets, then Layout > Lay Out Horizontally. This will put them inside an QHBoxLayout.
   * **Add a List Widget:**
     + From the "Widget Box", find Item Views and drag a List Widget onto the form.
     + Set its objectName to taskListWidget.
   * **Add a "Remove Selected" Push Button:**
     + Drag another Push Button onto the form.
     + Set its objectName to removeButton.
     + Set its text to Remove Selected.
   * **Apply a Top-Level Vertical Layout:**
     + In the "Object Inspector", select the main Widget.
     + Click the "Lay Out Vertically" icon in the Designer toolbar. This will arrange the horizontal layout (containing line edit and add button), the list widget, and the remove button vertically.
   * **Save the UI:** File > Save All (Ctrl+S / Cmd+S).
3. **Implement Logic:**
   * **Open widget.cpp:**
     + Double-click on widget.cpp.
   * **Connect "Add Task" Button:**
     + Go back to widget.ui.
     + Right-click on the addButton (Add Task button).
     + Select Go to Slot... > clicked().
     + Implement the slot in widget.cpp:

C++

void Widget::on\_addButton\_clicked()

{

QString task = ui->taskLineEdit->text().trimmed(); // Get text, remove leading/trailing whitespace

if (!task.isEmpty()) { // Only add if text is not empty

ui->taskListWidget->addItem(task); // Add the task to the list

ui->taskLineEdit->clear(); // Clear the input field

ui->taskLineEdit->setFocus(); // Put cursor back in the input field

}

}

* + **Connect "Remove Selected" Button:**
    - Go back to widget.ui.
    - Right-click on the removeButton (Remove Selected button).
    - Select Go to Slot... > clicked().
    - Implement the slot in widget.cpp:

C++

void Widget::on\_removeButton\_clicked()

{

// Get the currently selected item's row index

int currentRow = ui->taskListWidget->currentRow();

// Check if an item is actually selected (currentRow will be -1 if nothing is selected)

if (currentRow != -1) {

// Remove the item from the list widget

// QListWidget::takeItem takes ownership of the item and removes it from the list.

// If you don't store the returned QListWidgetItem\*, it will be automatically deleted

// when it goes out of scope (Qt handles this cleanup).

delete ui->taskListWidget->takeItem(currentRow);

}

}

* + - Save widget.cpp.

1. **Run the Application:**
   * Click the green "Run" button (or Ctrl+R / Cmd+R).
   * Type a task in the input field and click "Add Task".
   * Select an item in the list and click "Remove Selected". Test both functionalities.