Get Started with PySide6 Tutorial

Contents

Resources

- https://www.pythonguis.com/pyside6-tutorial/ (Excellent tutorials by Martin Fitzpatrick
- https://www.pythonguis.com/tutorials/pyside6-first-steps-qt-designer/ (How to create an app using QtDesigner.
- https://zetcode.com/gui/pysidetutorial/

Install PySide6

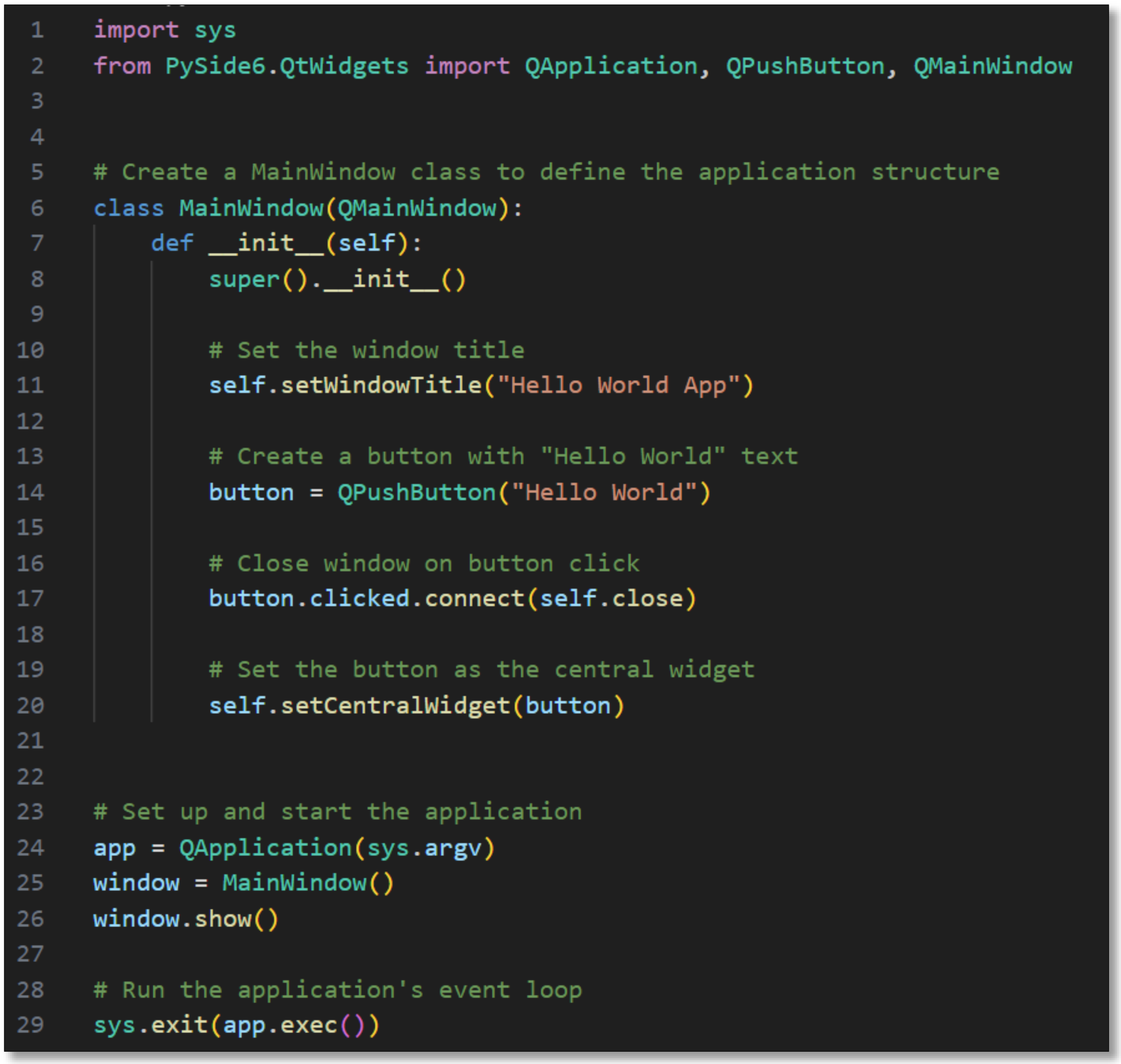
PySide6 is the official Python module from the Qt for Python project, which provides access to the complete Qt 6.0+ framework.

```
pip install pyside6
```

Tutorial 1: First Window

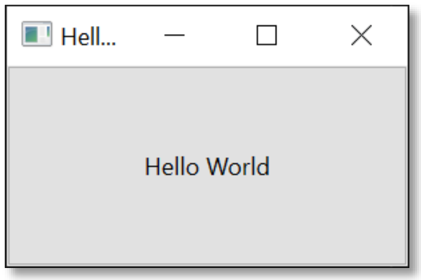
The first step in building any GUI is to create a Window. Of course, it displays Hello World!

This application is one big button. Not too cool . . . yet.



```python
import sys
from PySide6.QtWidgets import QApplication, QPushButton, QMainWindow


# Create a MainWindow class to define the application structure
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # Set the window title
        self.setWindowTitle("Hello World App")

        # Create a button with "Hello World" text
        button = QPushButton("Hello World")

        # Close window on button click
        button.clicked.connect(self.close)

        # Set the button as the central widget
        self.setCentralWidget(button)


# Set up and start the application
app = QApplication(sys.argv)
window = MainWindow()
window.show()

# Run the application's event loop
sys.exit(app.exec())
```

Example run:



Tutorial 2: First Qt Widgets Designer Program

Create a Simple UI with Qt Widgets Designer

1. **Open Qt Designer**:
   - Open a command prompt: pyside6-designer
     It will prompt you to create a new form.
2. **Create a New Form**:
   - Select **Main Window** from the templates.
   - Click **Create**.
3. **Add Widgets to the Main Window**:
   - **Drag and Drop Widgets**: You can drag widgets from the Widget Box (left panel) onto the main window.
   - For this example, drag a **Push Button** onto the window.
4. **Customize Widgets**:
   - Click the button to select it.
   - In the **Property Editor** (right panel), set the following properties:
     - **Object Name**: Change to **btn_push** (this will be the variable name).
     - **Text**: Change to "Click Me!" (this will be the button's label).
5. **Add a QLabel**:
   - From the **Widget Box** on the left, drag a **Label** onto the main window. (The Label widget is at the bottom of the Widget Box.)
   - Position it above or below the button.

    - Adjust the width a little wider to hold the text. You can come back and adjust this later if it doesn’t quite fit.

6. **Customize QLabel Properties**:
    - Select the QLabel, and in the **Property Editor** on the right, set the following properties:
    - **Object Name**: Change it to **lbl_display** (this will be the variable name).
    - **Text**: Set this to something like "Ready" or leave it empty if you want the label to start blank.

7. **Save the UI File**:
    - Save the design as a .ui file in the same folder as your program, for example, **hello_designer_window.ui**

Converting the UI File to Python Code

Convert the **.ui** file to a Python file using **pyside6-uic**. This generates code that can be imported and used in your Python program.

In your terminal, navigate to the directory where you saved **hello_designer_window.ui** and run:

```
# Create a batch file to run this command
pyside6-uic hello_designer_window.ui -o hello_designer_window.py
pause
```

This command will generate a **hello_designer.py** file.

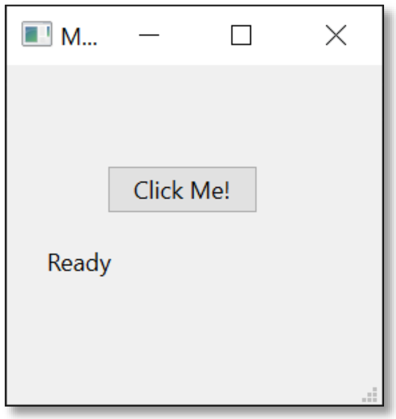
Create the Main Application Code

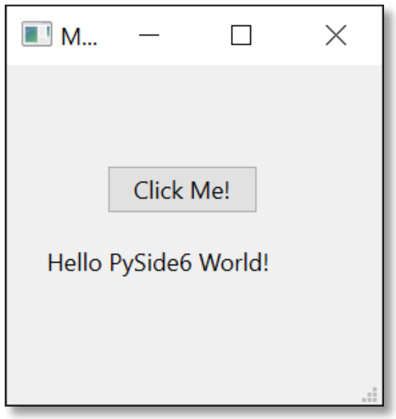
With the generated main_window.py file, create a new Python script to run the application.

Create a Python file named: **hello_designer_app.py**

```
# Filename: hello_designer_app.py
import sys
from PySide6.QtWidgets import QApplication, QMainWindow
# Import the generated UI class
from hello_designer_window import Ui_MainWindow


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()  # Create an instance of the UI
        self.ui.setupUi(self)  # Set up the UI in this MainWindow

        # Connect button signal to the function
        self.ui.btn_push.clicked.connect(self.update_label)

    def update_label(self):
        # Update the label text
        self.ui.lbl_display.setText("Hello PySide6 World!")


# Initialize the application
app = QApplication(sys.argv)
window = MainWindow()
window.show()

# Run the application's event loop
sys.exit(app.exec())
```

Example run:





While this program is simple and not very beautiful, it gives you a framework to build more complex UI's.

Tutorial 2: ToDo Application

Let's create a more useful, more extensive application.

Design the UI in Qt Designer

1. **Create a New Main Window**:
    - Select "Main Window" as the template in **Qt Designer**.
2. **Add Widgets**:
    - **List Widget**: Drag a QListWidget onto the window. This widget will display your tasks.
    - **Line Edit**: Below the QListWidget, add a QLineEdit for users to enter new tasks.
    - **Add Button** (QPushButton): Place an "Add" button next to the QLineEdit.
    - **Delete Button** (QPushButton): Add another button labeled "Delete" to remove tasks.
3. **Set Object Names**:
    - Set each widget's object name in **Qt Designer** for easy reference in code:
        - QListWidget – lst_task
        - QLineEdit – edt_task
        - Add button – btn_add
        - Delete button – btn_delete
4. **Save the UI**:
    - Save the file as **todo_window.ui** in your project folder.

Convert the UI File to Python Code

Convert the **.ui** file to a Python file using **pyside6-uic**. This generates code that can be imported and used in your Python program.

In your terminal, navigate to the directory where you saved **hello_designer_window.ui** and run:

```
# Create a batch file to run this command
pyside6-uic todo_window.ui -o todo_window.py
pause
```

This command will generate a **todo_window.py** file.

Create the Main Application Code

With the generated todo_window.py file, create a new Python script to run the application.

Create a Python file named: **todo_app.py**

```
"""
    File: todo_app.py
    Description: A simple To-Do list application using PySide6
"""

import sys
from PySide6.QtWidgets import QApplication, QMainWindow, QListWidgetItem
from PySide6.QtGui import QIcon
# Import the GUI layout created using PySide6's designer
from todo_window import Ui_MainWindow
# Import JSON library for saving/loading task data
import json
```

Setup main application.

```
# Define the main application class, inheriting from QMainWindow to create
the main app window
class TodoApp(QMainWindow):
    def __init__(self):
        # Call the parent constructor to initialize QMainWindow
        super().__init__()

        # Create an instance of the UI layout
        self.ui = Ui_MainWindow()

        # Set up the UI within the QMainWindow
        self.ui.setupUi(self)

        # Set the window icon for the application using a png file
        self.setWindowIcon(QIcon("todoicon.png"))

        # Load tasks from a file if it exists, display saved tasks on
startup
        self.load_tasks()

        # Connect button actions to respective functions
        self.ui.btn_add.clicked.connect(
            self.add_task)
        self.ui.btn_delete.clicked.connect(
            self.delete_task)
```

Add task Method

```
# ------------------------------ ADD TASK -------------------------------- #
    def add_task(self):
        # Get and trim text from input field
        task_text = self.ui.edt_task.text().strip()

        # Proceed only if the input is not empty
        if task_text:
            # Create a list item with the task text
            item = QListWidgetItem(task_text)

            # Add the new item to the task list widget
            self.ui.lst_task.addItem(item)

            # Clear the input field after adding the task
            self.ui.edt_task.clear()

            # Save the updated list of tasks to the file
            self.save_tasks()
```

Delete task method

```
# ------------------------------ DELETE TASK ----------------------------- #
    def delete_task(self):
        # Loop through each selected task item and
        # remove it from the list widget
        for item in self.ui.lst_task.selectedItems():
            # Remove item based on its row
            self.ui.lst_task.takeItem(self.ui.lst_task.row(item))

        # Save the updated list after deletion
        self.save_tasks()
```

Load tasks method.

```
# ------------------------------ LOAD TASK ------------------------------- #
    def load_tasks(self):
        """Load tasks from a JSON file and populate the task list widget"""
        try:
            # Open JSON file for reading
            with open("tasks.json", "r") as file:
                # Load tasks from JSON file
                tasks = json.load(file)

                # Iterate through each task in loaded data
                for task in tasks:
                    # Add task to list widget
                    self.ui.lst_task.addItem(QListWidgetItem(task))

        # Handle case where file does not exist
        except FileNotFoundError:
            # Do nothing if file not found
            pass
```

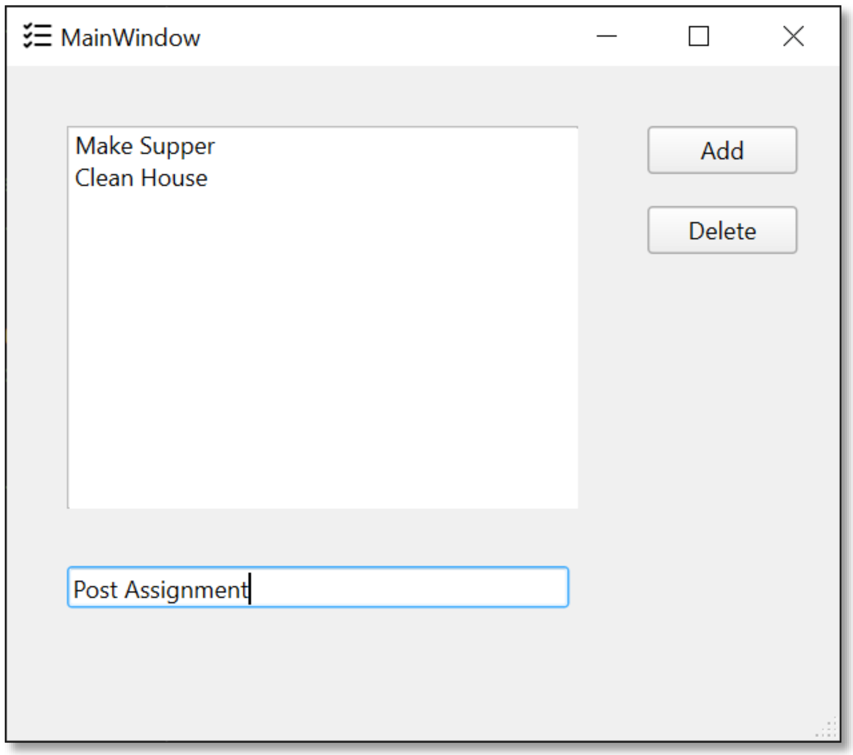
Save tasks method.

```
# ------------------------------ LOAD TASK ------------------------------- #
    def load_tasks(self):
        """Load tasks from a JSON file and populate the task list widget"""
        try:
            # Open JSON file for reading
            with open("tasks.json", "r") as file:
                # Load tasks from JSON file
                tasks = json.load(file)

                # Iterate through each task in loaded data
                for task in tasks:
                    # Add task to list widget
                    self.ui.lst_task.addItem(QListWidgetItem(task))

        # Handle case where file does not exist
        except FileNotFoundError:
            # Do nothing if file not found
            pass
```

Start the program.

```
# Main entry point for the application
if __name__ == "__main__":
    # Create the application instance with command-line arguments
    app = QApplication(sys.argv)

    # Set a more modern QT style
    app.setStyle('Fusion')

    # Create the main window instance
    window = TodoApp()

    # Display the main window
    window.show()

    # Execute the application’s main loop, setup clear exit of program
    sys.exit(app.exec())
```

Example run:



Make Supper
Clean House

Post Assignment