

PyQt Graphical User Interface

- There are multiple GUI modules for Python. These lessons will use PyQt which is Qt with a python wrapper. Qt is a framework for developing graphical user interfaces. It is cross platform which means the same code works on Windows, Mac OS, and Linux.
- Qt is written in C++. PyQt provides a python interface or wrapper that lets you use the Qt software in Python.
- This lesson will show you how to create a GUI with PyQt and a framework or program structure to use.
- First, install PyQt. We will use PyQt version 6.
- Open a terminal and enter
 - **`pip3 install pyqt6`**

Program Structure

- A PyQt program has three basic parts.
 - Part 1
 - The top imports all the needed parts of PyQt. For now, we import everything. You can import select parts of PyQt but this is easier. The statement from tells Python to import only parts of the specified module. An asterisk indicates everything in the module.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        b = QPushButton('Button')
        grid = QGridLayout()
        grid.addWidget(b,1,0)
        self.setLayout(grid)
        b.pressed.connect(self.button_pressed)

    def button_pressed(self):
        print("Button Pressed")

app = QApplication(sys.argv)
win = Button()
win.show()
sys.exit(app.exec())
```

Program Structure

- A PyQt program has three basic parts.
 - Part 2
 - The second part is the GUI itself which is defined as a class. We make our GUI a widget. The argument in the class is QWidget. This allows the class to be included in other GUIs. Writing the GUI this way allows you to create complex widgets that can be incorporated in other widgets.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        b = QPushButton('Button')
        grid = QGridLayout()
        grid.addWidget(b, 1, 0)
        self.setLayout(grid)
        b.pressed.connect(self.button_pressed)
        b.released.connect(self.button_released)

    def button_pressed(self):
        print("Button Pressed")

    def button_released(self):
        print("Button Released")

app = QApplication(sys.argv)
win = Button()
win.show()
sys.exit(app.exec())
```

Program Structure

- In the class called Button, the `__init__()` function must be included and use this specific name. Every time the class is instantiated, the `__init__()` function is called. It is used to initialize the widget.
- Notice the `super().__init__()` function call is the first line in the `__init__()` function. This is used by the system so this class inherits everything from the parent.
- Next, a pushbutton is instantiated or created and called `b`. The argument is the text that will be displayed in the button.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        b = QPushButton('Button')
        grid = QGridLayout()
        grid.addWidget(b,1,0)
        self.setLayout(grid)
        b.pressed.connect(self.button_pressed)
        b.released.connect(self.button_released)

    def button_pressed(self):
        print("Button Pressed")

    def button_released(self):
        print("Button Released")

app = QApplication(sys.argv)
win = Button()
win.show()
sys.exit(app.exec())
```

Program Structure

- How the button is displayed and positioned in the widget or window is by defining a layout. We will use a grid layout. A grid layout lets you position widgets in rows and columns. The window or widget gets divided up into a grid of rows and columns.
- To add widgets to the grid layout, the **addWidget()** function is used. The arguments are the widget, then the row position and the column position. Since there is only one widget, the position is 0,0.
- After all widgets are added, the **setLayout()** function is called and the name of the layout is the argument. Notice it is **self.setLayout()** since it is for this widget or window.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        b = QPushButton('Button')
        grid = QGridLayout()
        grid.addWidget(b,1,0)
        self.setLayout(grid)
        b.pressed.connect(self.button_pressed)
        b.released.connect(self.button_released)

    def button_pressed(self):
        print("Button Pressed")

    def button_released(self):
        print("Button Released")

app = QApplication(sys.argv)
win = Button()
win.show()
sys.exit(app.exec())
```

Program Structure

- To make the push button do something, a signal needs to be connected to the button.
- There are two signals we will use; pressed and released.
- The **connect()** function is used. The argument is the function to call when the action is performed.
- The functions for the push button action must be included in the same class.
- When ever the button is pressed, the **button_pressed()** function will be called.
- When ever the button is released, the **button_release()** function will be called.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        b = QPushButton('Button')
        grid = QGridLayout()
        grid.addWidget(b,1,0)
        self.setLayout(grid)
        b.pressed.connect(self.button_pressed)
        b.released.connect(self.button_released)

    def button_pressed(self):
        print("Button Pressed")

    def button_released(self):
        print("Button Released")

app = QApplication(sys.argv)
win = Button()
win.show()
sys.exit(app.exec())
```

Program Structure

- The third part is running the program.
- We instantiate QApplication which is the code that runs and handles all the operations in the GUI.
- Next, the class is instantiated and called win. **win.show()** makes the window visible.
- Finally, **app.exec()** is executed which runs the GUI. Notice it is in **sys.exit()**. **sys.exit()** is called which executes **app.exec()** as an argument. When you close the window, **app.exec()** quits and returns a value indicating if there was an error. Its use is for when other programs call this program and the function passes the value to the calling program.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        b = QPushButton('Button')
        grid = QGridLayout()
        grid.addWidget(b,1,0)
        self.setLayout(grid)
        b.pressed.connect(self.button_pressed)
        b.released.connect(self.button_released)

    def button_pressed(self):
        print("Button Pressed")

    def button_released(self):
        print("Button Released")

app = QApplication(sys.argv)
win = Button()
win.show()
sys.exit(app.exec())
```

Program Structure

- Save the program as **button.py**
- Run the program. A small window will appear with the button. Click on it and look at the messages in the IDLE console.
- Close the program.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        b = QPushButton('Button')
        grid = QGridLayout()
        grid.addWidget(b,1,0)
        self.setLayout(grid)
        b.pressed.connect(self.button_pressed)
        b.released.connect(self.button_released)

    def button_pressed(self):
        print("Button Pressed")

    def button_released(self):
        print("Button Released")

app = QApplication(sys.argv)
win = Button()
win.show()
sys.exit(app.exec())
```

Push Buttons

- Let's add a second button.
- First, in the `__init__()` function, the second button is added called `b2`.
- It is added to the grid one row below the first button.
- The pressed signal is connected to a new function.
- A new function is added for the second button.
- Run the program and see the second button added.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        b = QPushButton('Button')
        b2 = QPushButton('Hello')
        grid = QGridLayout()
        grid.addWidget(b,1,0)
        grid.addWidget(b2,2,0)
        self.setLayout(grid)
        b.pressed.connect(self.button_pressed)
        b.released.connect(self.button_released)
        b2.pressed.connect(self.button2_pressed)

    def button_pressed(self):
        print("Button Pressed")

    def button_released(self):
        print("Button Released")

    def button2_pressed(self):
        print("Hello")

app = QApplication(sys.argv)
win = Button()
win.show()
sys.exit(app.exec())
```

Push Buttons

- Let's modify the size of the button and change the font of the text in the button.
- The size of the button can be adjusted to a fixed size by specifying the number of pixels wide and high. Without this, the button will be sized to the length of the text in the button.
- The font is changed with the **setFont()** function and **QFont()** is used to specify the type of font and the size of the text.
- No other changes need to be made.
- Try it out and try repositioning the buttons.
- Add another button and a function for it.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        b = QPushButton('Button')
        b.setFixedSize(120,100)
        b.setFont(QFont('Arial',18))
        b2 = QPushButton('Hello')
        b2.setFixedSize(120,100)
        b2.setFont(QFont('Arial',18))
        grid = QGridLayout()
        grid.addWidget(b,1,0)
        grid.addWidget(b2,2,0)
        self.setLayout(grid)
        b.pressed.connect(self.button_pressed)
        b.released.connect(self.button_released)
        b2.pressed.connect(self.button2_pressed)
```

Push Buttons

- Let's have one of the button functions change the color of the button. In order to do that, the button needs to be made available to the function. The way to do that is to add **self.** to the front of the button name. This makes the button name available to all functions in the class. It is like a global variable in the class.
- Add **self.** to all the lines highlighted.
- Add a variable **self.b_state** and set it to the value of zero. This will be used in the function.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        self.b = QPushButton('Button')
        self.b.setFixedSize(120,100)
        self.b.setFont(QFont('Arial',18))
        b2 = QPushButton('Hello')
        b2.setFixedSize(120,100)
        b2.setFont(QFont('Arial',18))
        grid = QGridLayout()
        grid.addWidget(self.b,1,0)
        grid.addWidget(b2,2,0)
        self.setLayout(grid)
        self.b.pressed.connect(self.button_pressed)
        self.b.released.connect(self.button_released)
        b2.pressed.connect(self.button2_pressed)
        self.b_state = 0
```

Push Buttons

- Let's modify the function **button_pressed()** to toggle the color of the button.
- First, the **b_state** variable is changed. The equation will toggle the variable between 0 and 1.
- Next, the value is checked. If **b_state** is 1, the button is turned red.
- If **b_state** is 0, the default background is set on the button.
- Try it out.
- Update the program to make the other buttons to toggle different colors.

```
def button_pressed(self):  
    print("Button Pressed")  
    self.b_state = 1 - self.b_state  
    if self.b_state == 1:  
        self.b.setStyleSheet('background-color: red;')  
    else:  
        self.b.setStyleSheet('')
```

Modifying the Window

- Here are a couple useful functions to control the Window. The first is setting the window title. This puts the text in the window title.
- The second one sets the geometry of the window. The first two arguments position the window at the pixel coordinates of the display. The second two arguments specify the width and height of the window.
- Try it out and change it to see how the window behaves.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Buttons")
        self.setGeometry(100, 100, 400, 400)
        self.b = QPushButton('Button')
        self.b.setFixedSize(120, 100)
        self.b.setFont(QFont('Arial', 18))
        b2 = QPushButton('Hello')
        b2.setFixedSize(120, 100)
        b2.setFont(QFont('Arial', 18))
        grid = QGridLayout()
        grid.addWidget(self.b, 1, 0)
        grid.addWidget(b2, 2, 0)
        self.setLayout(grid)
        self.b.pressed.connect(self.button_pressed)
        self.b.released.connect(self.button_released)
        b2.pressed.connect(self.button2_pressed)
        self.b_state = 0
```

Label Widget

- Labels are where text can be placed in a window or widget.
- Add the label highlighted. The label will appear above the two buttons.
- Run the program and the label should appear.
- Change the font size of the label. It is the same as changing the font of a button.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Buttons")
        self.setGeometry(100,100,400,400)
        self.L1 = QLabel("Label")
        self.L1.setFont(QFont('Arial',18))
        self.b = QPushButton('Button')
        self.b.setFixedSize(120,100)
        self.b.setFont(QFont('Arial',18))
        b2 = QPushButton('Hello')
        b2.setFixedSize(120,100)
        b2.setFont(QFont('Arial',18))
        grid = QGridLayout()
        grid.addWidget(self.b,1,0)
        grid.addWidget(b2,2,0)
        grid.addWidget(self.L1,0,0)
        self.setLayout(grid)
        self.b.pressed.connect(self.button_pressed)
        self.b.released.connect(self.button_released)
        b2.pressed.connect(self.button2_pressed)
        self.b_state = 0
```

Label Widget

- The text in the Label can be changed with one function.
- Modify the **button2_pressed()** function and replace the print statement.
- Run the program again and press the second button. The label should change.

```
def button2_pressed(self):  
    self.L1.setText("New Label")
```

Line Edit Widget

- The next widget is called **QLineEdit**. It allows one line of text to be entered.
- The simplest way to create a line edit widget is shown in the first highlighted line. The widget is then placed below the second button at the bottom of the window.
- **editingFinished** is a signal that triggers the function when the Enter key is pressed after entering text.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Buttons")
        self.setGeometry(100, 100, 400, 400)
        self.L1 = QLabel("Label")
        self.L1.setFont(QFont('Arial', 18))
        self.b = QPushButton('Button')
        self.b.setFixedSize(120, 100)
        self.b.setFont(QFont('Arial', 18))
        b2 = QPushButton('Hello')
        b2.setFixedSize(120, 100)
        b2.setFont(QFont('Arial', 18))
        self.LE = QLineEdit(self)
        grid = QGridLayout()
        grid.addWidget(self.b, 1, 0)
        grid.addWidget(b2, 2, 0)
        grid.addWidget(self.L1, 0, 0)
        grid.addWidget(self.LE, 3, 0)
        self.setLayout(grid)
        self.b.pressed.connect(self.button_pressed)
        self.b.released.connect(self.button_released)
        b2.pressed.connect(self.button2_pressed)
        self.LE.editingFinished.connect(self.edit)
        self.b_state = 0
```

Line Edit Widget

- Add the **edit()** function to the class.
- The edit function gets the text entered in the line edit widget and updates the label widget.
- Try it out. Remember to press the Enter key after entering some text.

```
def button2_pressed(self):  
    self.L1.setText("New Label")  
  
def edit(self):  
    text = self.LE.text()  
    self.L1.setText(text)
```

Line Edit Widget

- Change **editingFinished** to **editedText** in the connect function.
- Run the program again.
- You should see the label get updated as you type text into the line edit widget.
- You do not need to change anything in the **edit()** function.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Buttons")
        self.setGeometry(100,100,400,400)
        self.L1 = QLabel("Label")
        self.L1.setFont(QFont('Arial',18))
        self.b = QPushButton('Button')
        self.b.setFixedSize(120,100)
        self.b.setFont(QFont('Arial',18))
        b2 = QPushButton('Hello')
        b2.setFixedSize(120,100)
        b2.setFont(QFont('Arial',18))
        self.LE = QLineEdit(self)
        grid = QGridLayout()
        grid.addWidget(self.b,1,0)
        grid.addWidget(b2,2,0)
        grid.addWidget(self.L1,0,0)
        grid.addWidget(self.LE,3,0)
        self.setLayout(grid)
        self.b.pressed.connect(self.button_pressed)
        self.b.released.connect(self.button_released)
        b2.pressed.connect(self.button2_pressed)
        self.LE.editedText.connect(self.edit)
        self.b_state = 0
```

Slider Widget

- Next, we'll add a slider.
- The default state of the slider is to be vertical. The slider will be positioned to the right side of all the other widgets.
- Notice in the `addWidget()` function there are two extra numbers. The third number is the number of rows to span and the fourth number is the number of columns to span. The slider will be stretched over the four rows to the right of the four other widgets.
- Lastly, the `valueChanged` signal is connected to the function `self.slider()`. Any time the slider is moved, the function is called.

```
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *

class Button(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Buttons")
        self.setGeometry(100, 100, 400, 400)
        self.L1 = QLabel("Label")
        self.L1.setFont(QFont('Arial', 18))
        self.b = QPushButton('Button')
        self.b.setFixedSize(120, 100)
        self.b.setFont(QFont('Arial', 18))
        b2 = QPushButton('Hello')
        b2.setFixedSize(120, 100)
        b2.setFont(QFont('Arial', 18))
        self.LE = QLineEdit(self)
        self.slide = QSlider(self)
        self.slide.setRange(0, 100)
        grid = QGridLayout()
        grid.addWidget(self.b, 1, 0)
        grid.addWidget(b2, 2, 0)
        grid.addWidget(self.L1, 0, 0)
        grid.addWidget(self.LE, 3, 0)
        grid.addWidget(self.slide, 0, 1, 4, 1)
        self.setLayout(grid)
        self.b.pressed.connect(self.button_pressed)
        self.b.released.connect(self.button_released)
        b2.pressed.connect(self.button2_pressed)
        self.slide.valueChanged.connect(self.slider)
        self.LE.editedText.connect(self.edit)
        self.b_state = 0
```

Slider Widget

- Add the `slider()` function to the class.
- The function will get the new value of the slider and update the label.
- The value is an integer so it needs to be changed to a string type for the label widget.
- Run the program and play with the slider.
- There are more widgets and ways to customize the widgets. To find out more, check out:
<https://www.pythonguis.com/tutorials/pyqt6-widgets/>

```
def button2_pressed(self):  
    self.L1.setText("New Label")  
  
def edit(self):  
    text = self.LE.text()  
    self.L1.setText(text)  
  
def slider(self):  
    val = self.slide.value()  
    self.L1.setText(str(val))
```

