Building Local GUI Applications Using PyQt5

Travis Phillips

PyJax

10/12/2021

So What is PyQT5?

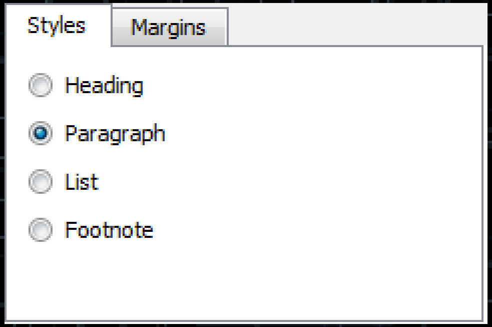
- In a nutshell, PyQT5 is the Python wrappers to the QT5 libraries.
- These can help build fairly nice GUI driven applications.

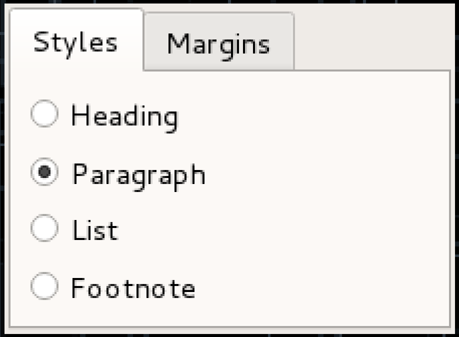
What is QT5?

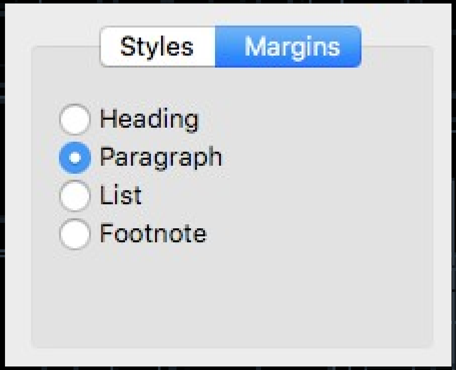
- Cross-platform C++ application framework
  - This is very important to understand
- Does so much more than just GUIs
  - Datetime, Databases, Bluetooth, Threading, Timers, Events, Processes, etc
  - Threading is worth noting, almost all GUIs require threading
  - IMO QT5's threading is better than Python's and should be used instead of Python's if you are building a PyQT5 application

What is QT5?

- But this talk will focus *MOSTLY* on the GUI part
- Will attempt to make the application’s GUI look native.





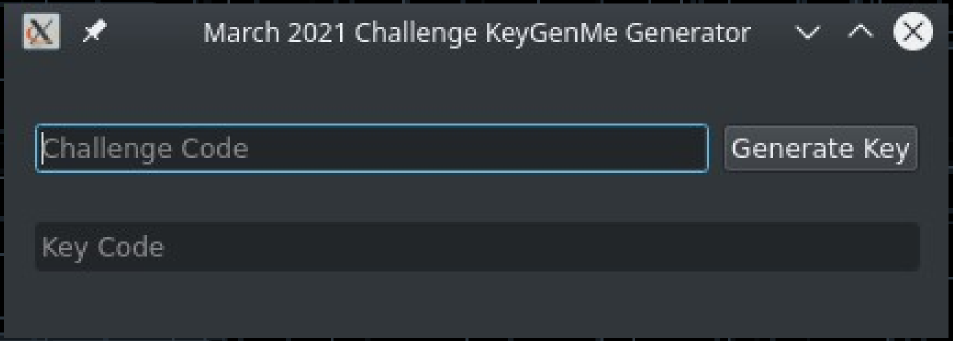


- Also supports a CSS-like formatting as well.

Assumed Knowledge of Intended Audience

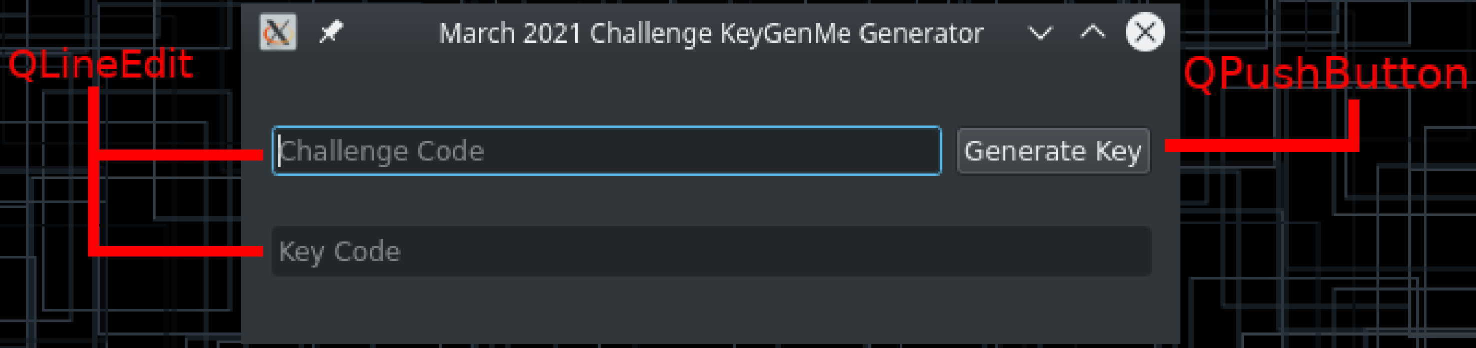
- You have a basic understanding of Python 3
- You have a basic understanding of OOP and classes
- C++ knowledge isn't required…
    - But the C++ documentation is MUCH better than the pyqt5 documentation

GUI App Basics: This is GUI



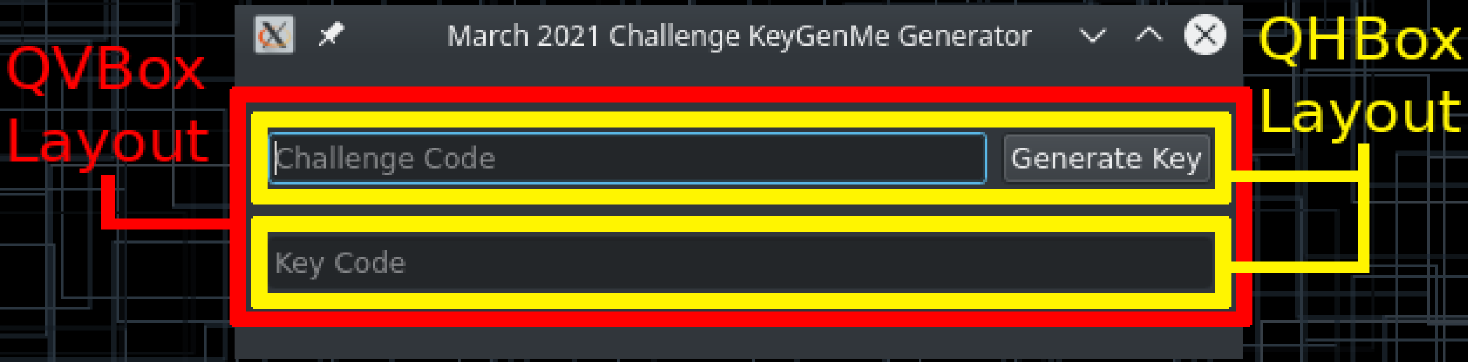
- You've probably seen one before, it's incredibly commonplace
  - But let's think about it for a minute
- GUIs use widgets & layouts
  - GUI apps are event driven & multi-threaded
    - **Heavy workloads SHOULD NOT occur on the main GUI thread**

GUI App Basics: Widgets



- Widgets are the GUI elements you'll use making GUI
- A QWidget can be used as a window.
  - Supports packing other widgets or layouts in it.
- Things to take input, trigger events, display data, etc

GUI App Basics: Layouts

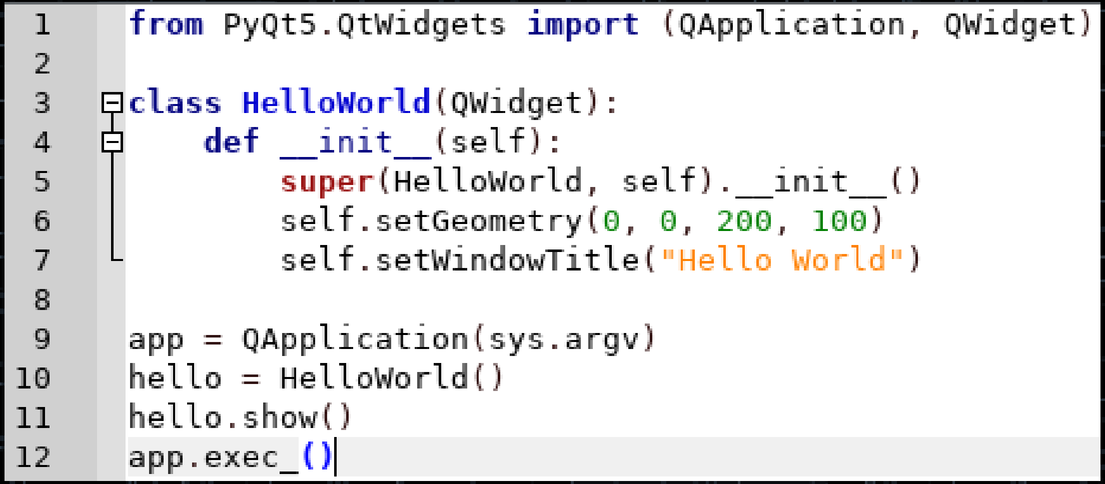


- Layouts provide a system for organizing the widgets
- You can do absolute positioning of widgets…
  - But layouts make life easier and scale automatically
- Can nest inside each other to make various configurations of layouts

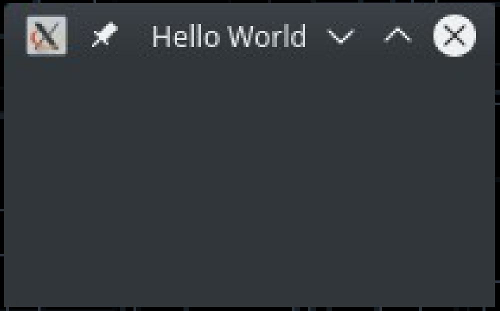
GUI App Basics: Events

- All GUIs usually wait for a user to do something
  - Click, type, mouse over, drag and drop, etc
- In QT5, such events trigger the widgets to emit signals
  - We can connect our code to these signals and hook callback functions to them

Basic Hello World!



```
from PyQt5.QtWidgets import (QApplication, QWidget)

class HelloWorld(QWidget):
    def __init__(self):
        super(HelloWorld, self).__init__()
        self.setGeometry(0, 0, 200, 100)
        self.setWindowTitle("Hello World")

app = QApplication(sys.argv)
hello = HelloWorld()
hello.show()
app.exec_()
```

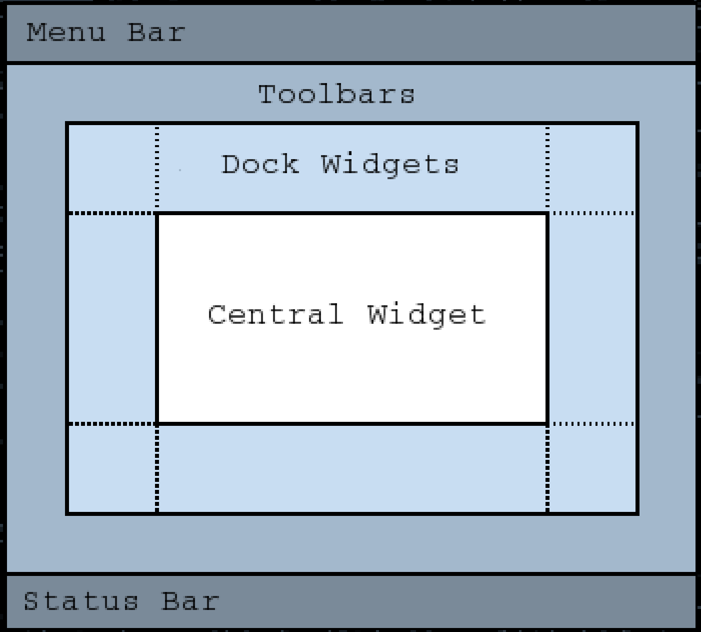


Widgets Showcase

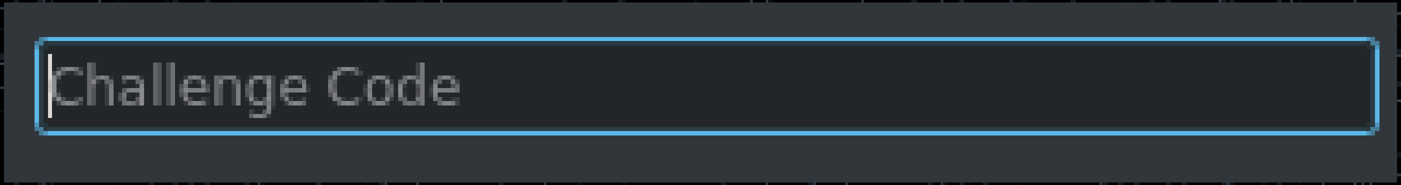
- Widgets make up visual elements of our applications
- QT5 provides several types
  - We cannot cover them all here
  - This is a highlights reel
- You can also make your own
- Let's go over a few

Widgets->Basic: QMainWindow

- Designed to provide a "Main Window" boilerplate.
- Provides central Widget, menu and status bar, docks and toolbars
- I usually use QWidget instead for my window instead, but this is here

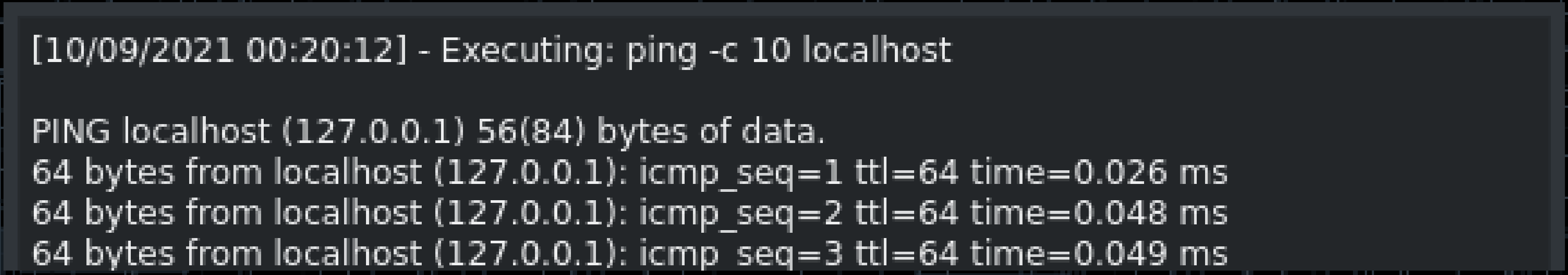


Widgets->Inputs: QLineEdit



- QLineEdit is designed to be a single line text entry field.
- Supports placeholder text and validators

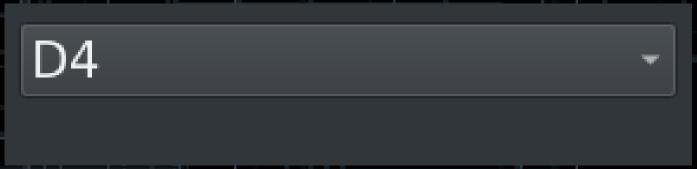
Widgets->Inputs: QTextEdit

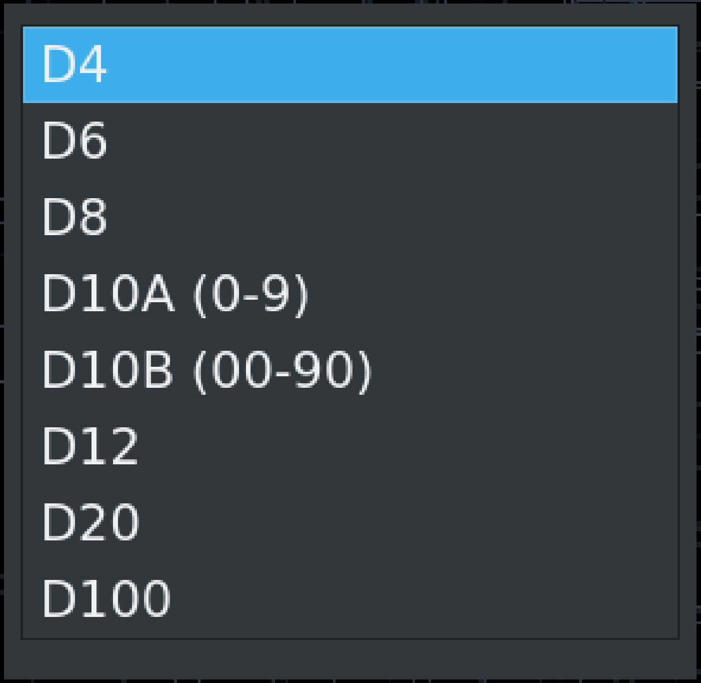


```
[10/09/2021 00:20:12] - Executing: ping -c 10 localhost

PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.026 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.049 ms
```

- QTextEdit is a large, multi-line text editor box
- Can be used for notes, or made read only for a console or log output stream.

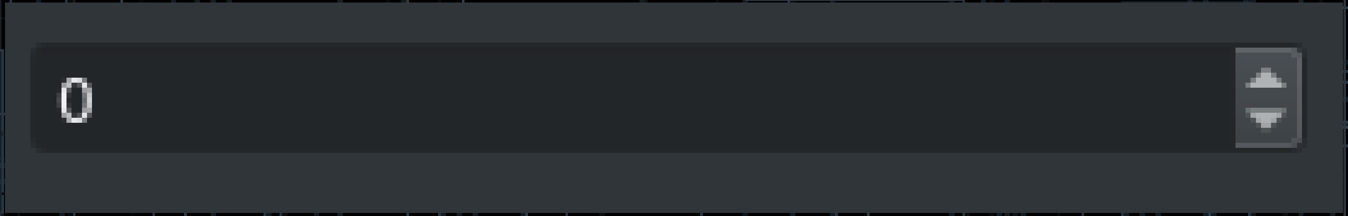
Widgets->Inputs: QComboBox





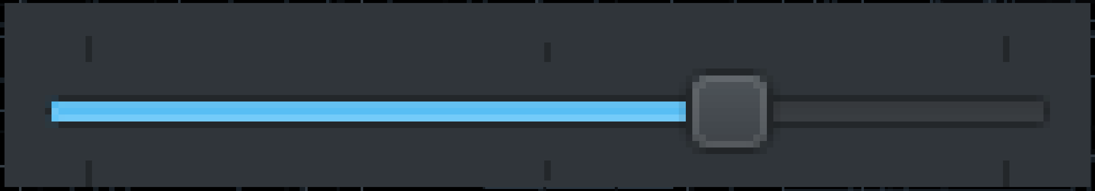
- QComboBox is a simple drop down selection box.
- Can also be made editable so it can be used for as a text input as well.

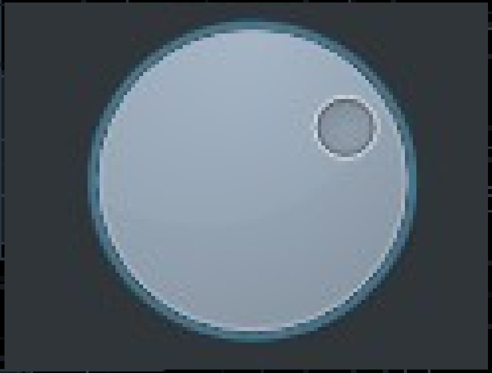
Widgets->Inputs: QSpinBox



- QSpinBox is usual used to increment a number up or down via up or arrows.

Widgets->Inputs: Qslider / QDial

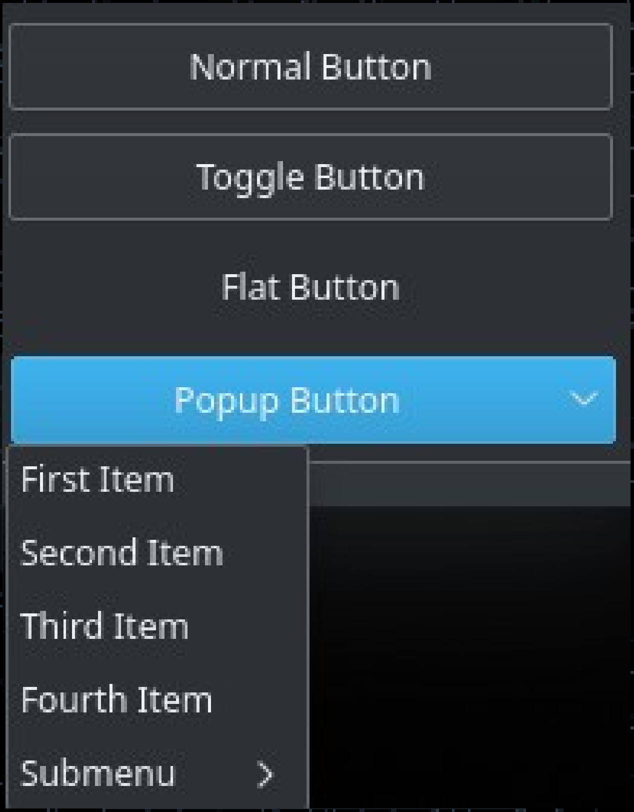




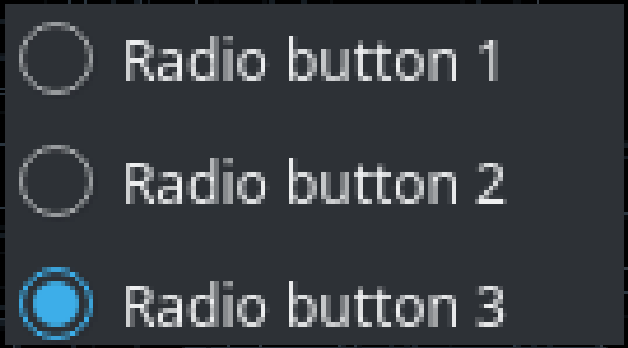
- Qslider & QDial are used to select a value in a range by dragging to the desired value.

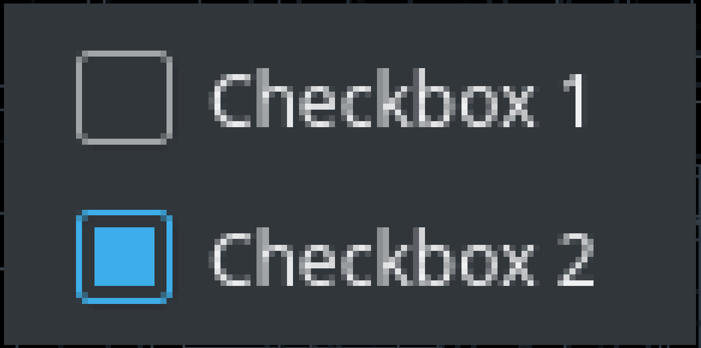
Widgets->Buttons: QPushButton

- QPushButton are the most common button type.
- .SetCheckable() can turn it into a toggle button as well.



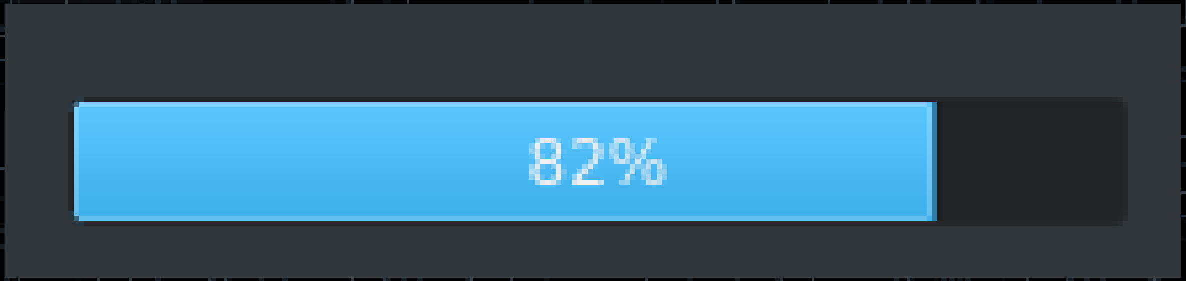
Widgets->Buttons: QRadioButton / QCheckBox





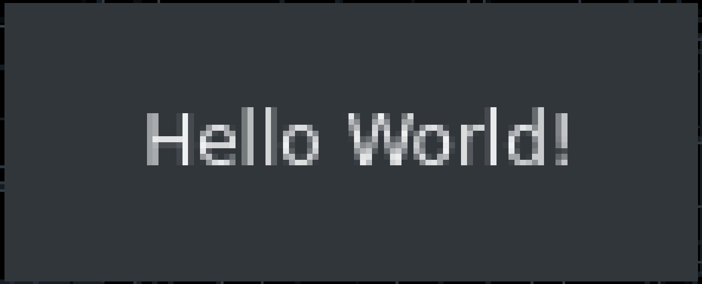
- Pretty standard radio and checkboxes

Widgets->Displays: QProgressBar



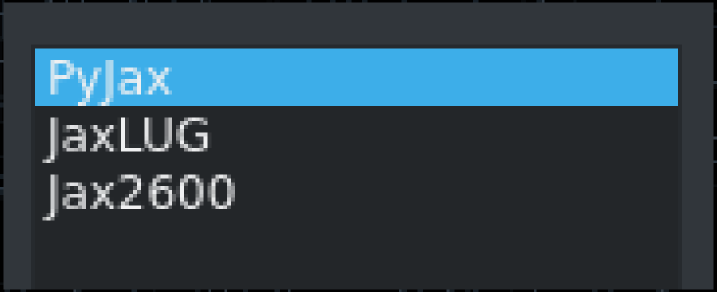
- A simple progress bar for displaying progress to the end user.

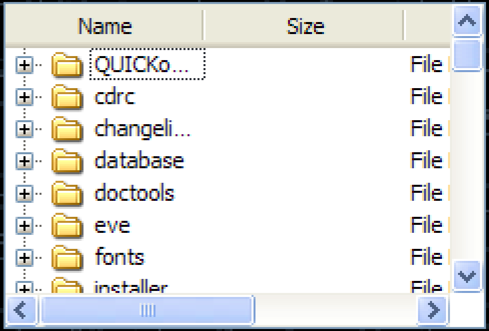
Widgets->Displays: QLabel

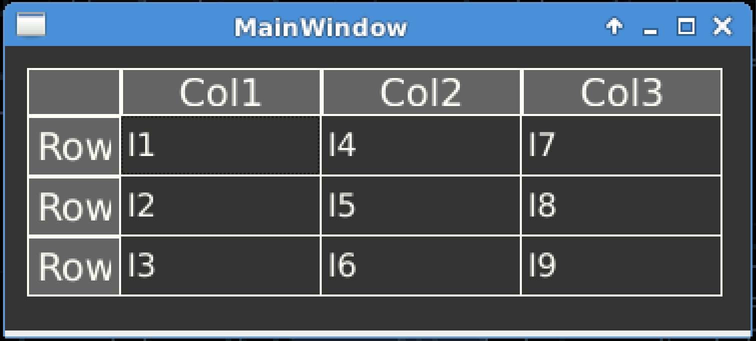


- A simple widget for labeling things or displaying read-only text.
- Supports style sheets in the the QT framework for styling.

Widgets->Displays: QListWidget / QTreeWidget / QTableWidget

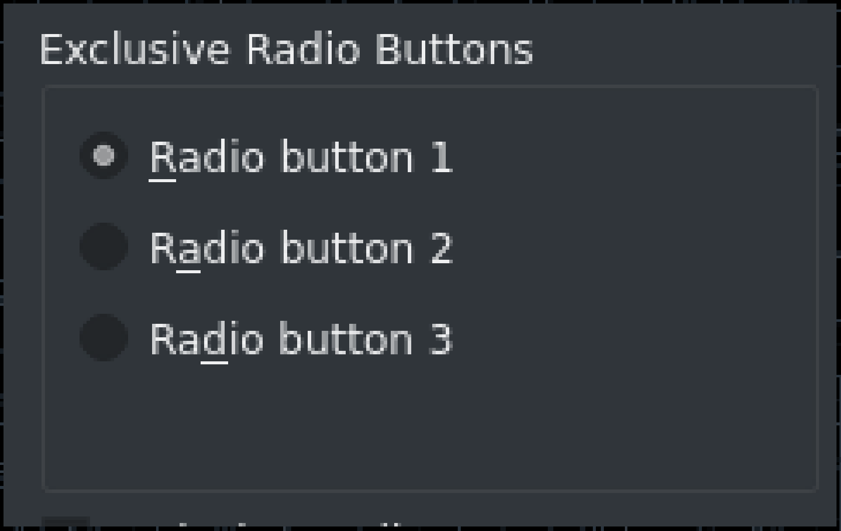





MainWindow

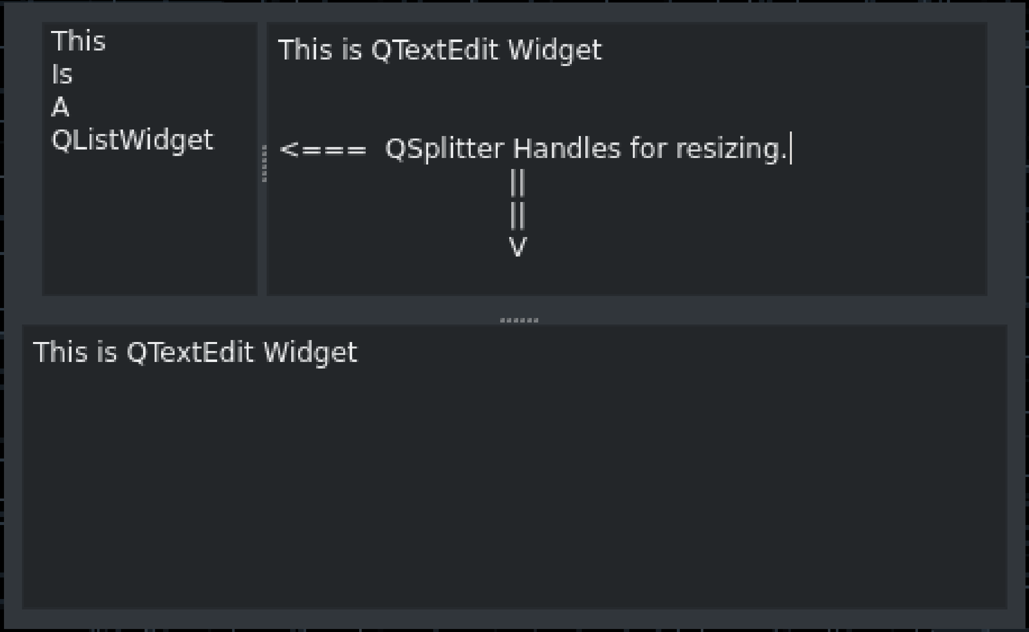
	Col1	Col2	Col3
Row	I1	I4	I7
Row	I2	I5	I8
Row	I3	I6	I9

Widgets->Organize: QGroupBox



- A way to Group widgets in a labeled box.
- Personally, I use this a lot for sub-classing a group of the UI together.

Widgets->Organize: QSplitter


This
Is
A
QListWidget

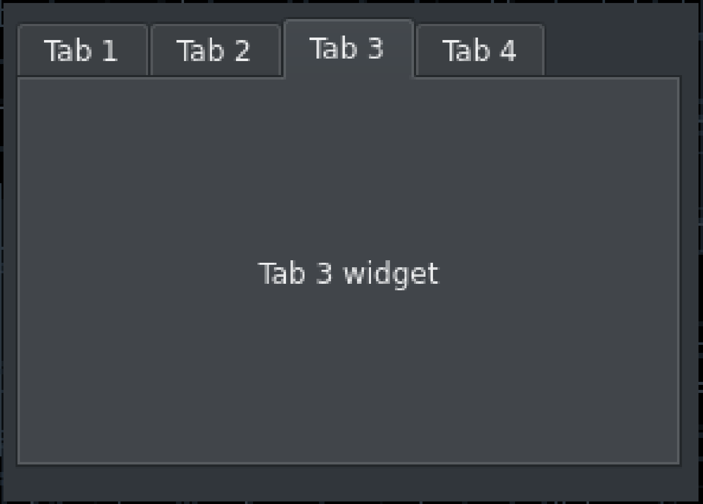
This is QTextEdit Widget

<=== QSplitter Handles for resizing.
||
||
V

This is QTextEdit Widget

- A splitter between widgets.
- Can be vertical or horizontal.
- Has handles for resizing widgets.

Widgets->Organize: QTabWidget

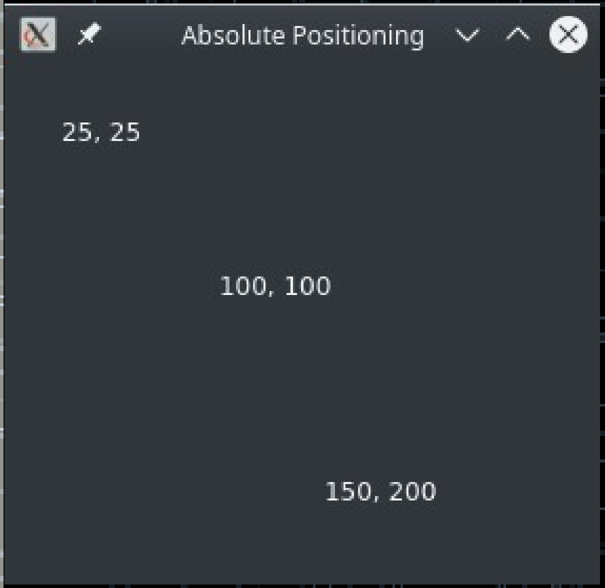


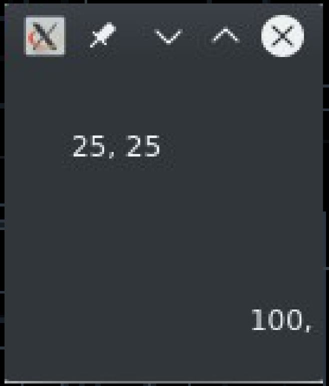
- Provides tabs for packing various widgets in a tabbed view.

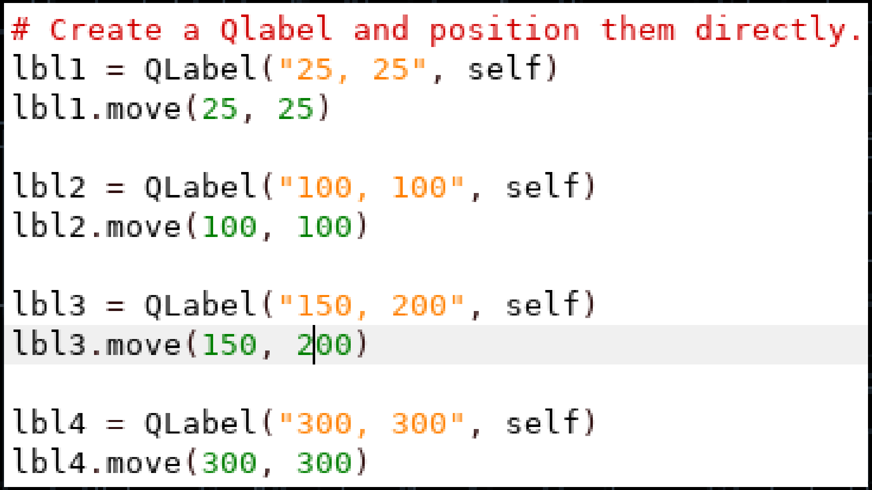
Layouts

- Now that we covered Widgets, let’s cover layouts
- Layouts offer a way to manage placing widgets in your UI.
- Offers may styles and handles a lot of the overhead devs usually don’t want to handle (resizing events, etc)
- Layouts can be nested to get the desired effects.

Layout Management: Absolute



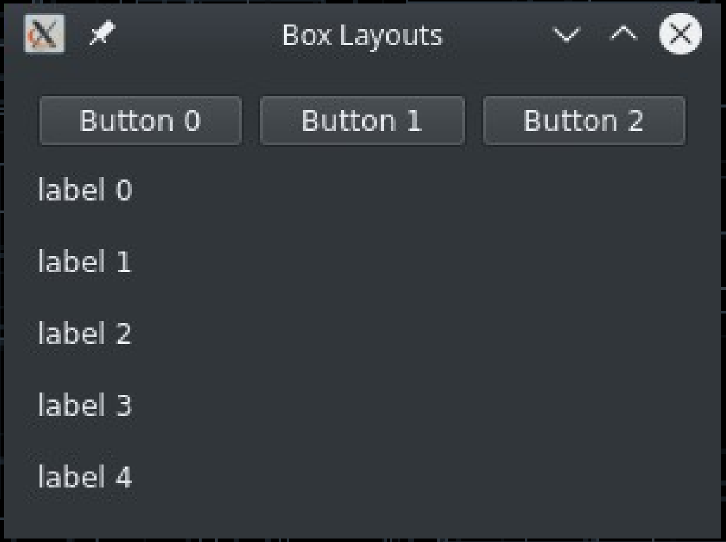


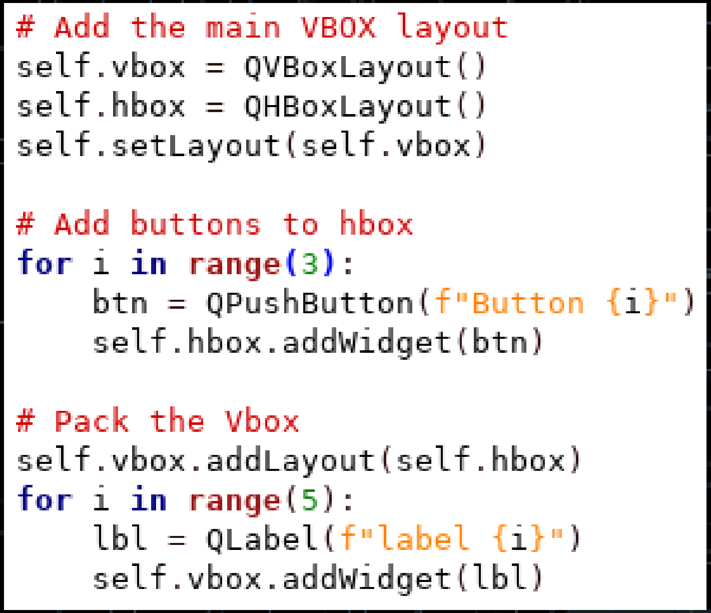


```
# Create a Qlabel and position them directly.
lbl1 = QLabel("25, 25", self)
lbl1.move(25, 25)

lbl2 = QLabel("100, 100", self)
lbl2.move(100, 100)

lbl3 = QLabel("150, 200", self)
lbl3.move(150, 200)

lbl4 = QLabel("300, 300", self)
lbl4.move(300, 300)
```

- Not a layout itself, but worth mentioning here
- Provided by widgets is the .move(x, y)
- Not a great approach since its absolute.
  - Resizing is now your problem

Layout Management: QHBoxLayout / QVBoxLayout

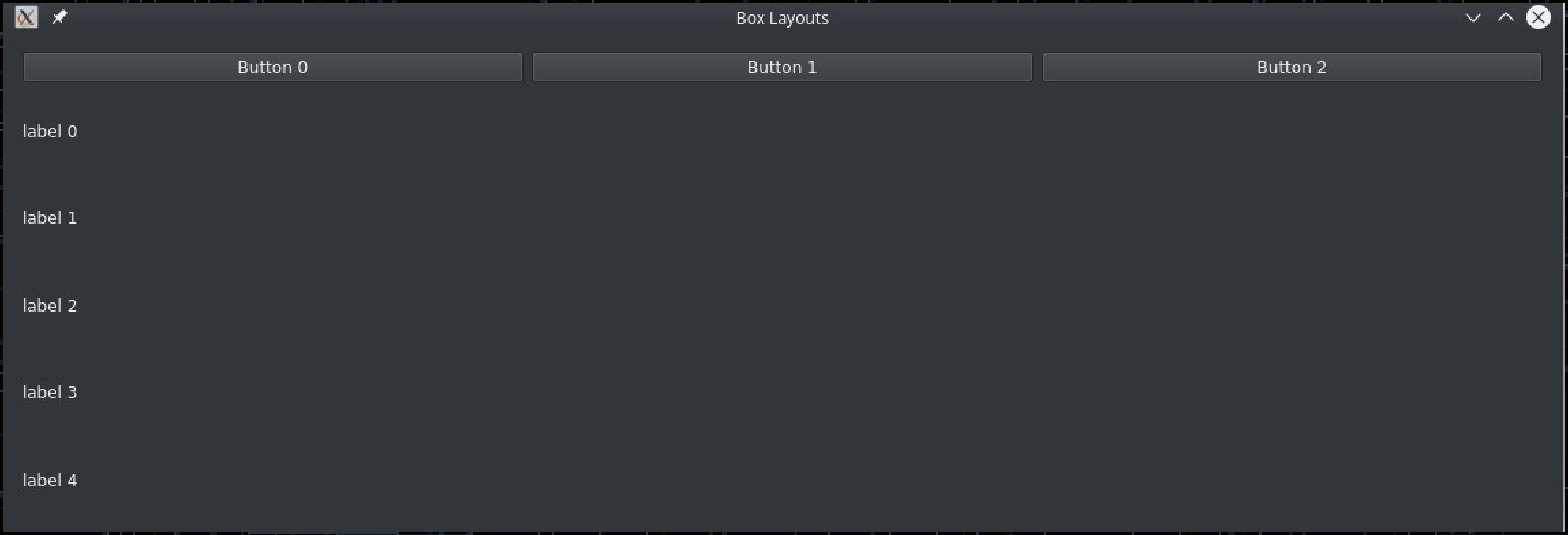




```python
# Add the main VBOX layout
self.vbox = QVBoxLayout()
self.hbox = QHBoxLayout()
self.setLayout(self.vbox)

# Add buttons to hbox
for i in range(3):
    btn = QPushButton(f"Button {i}")
    self.hbox.addWidget(btn)

# Pack the Vbox
self.vbox.addLayout(self.hbox)
for i in range(5):
    lbl = QLabel(f"label {i}")
    self.vbox.addWidget(lbl)
```

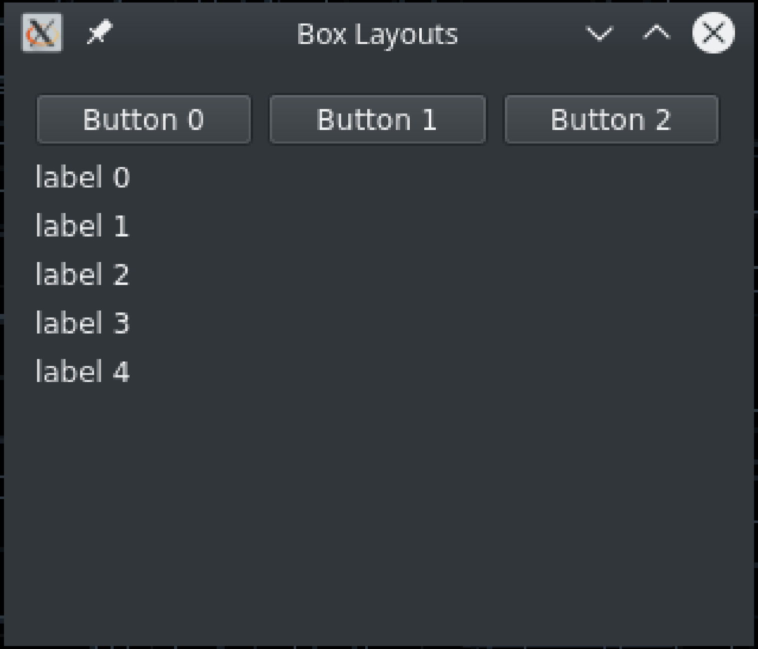
- The most common layouts, the horizontal and vertical box
- Just add widgets and layouts and they will be stacked accordingly.

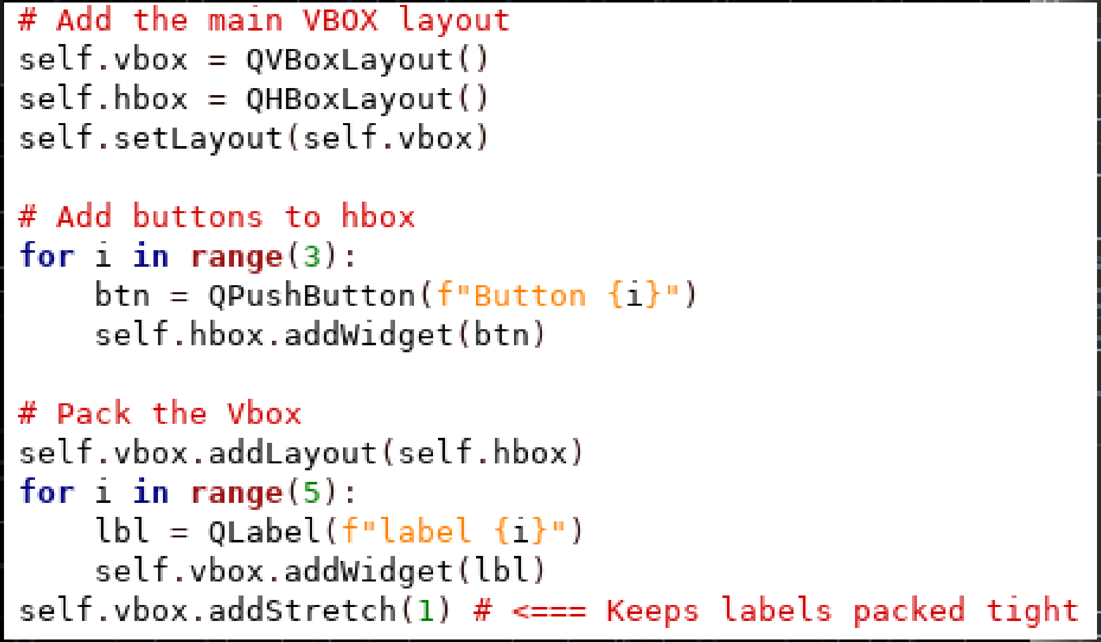
Layout Management: QHBoxLayout / QVBoxLayout



- Auto Resize
- Label spacing might not be preferred

Layout Management: QHBoxLayout / QVBoxLayout



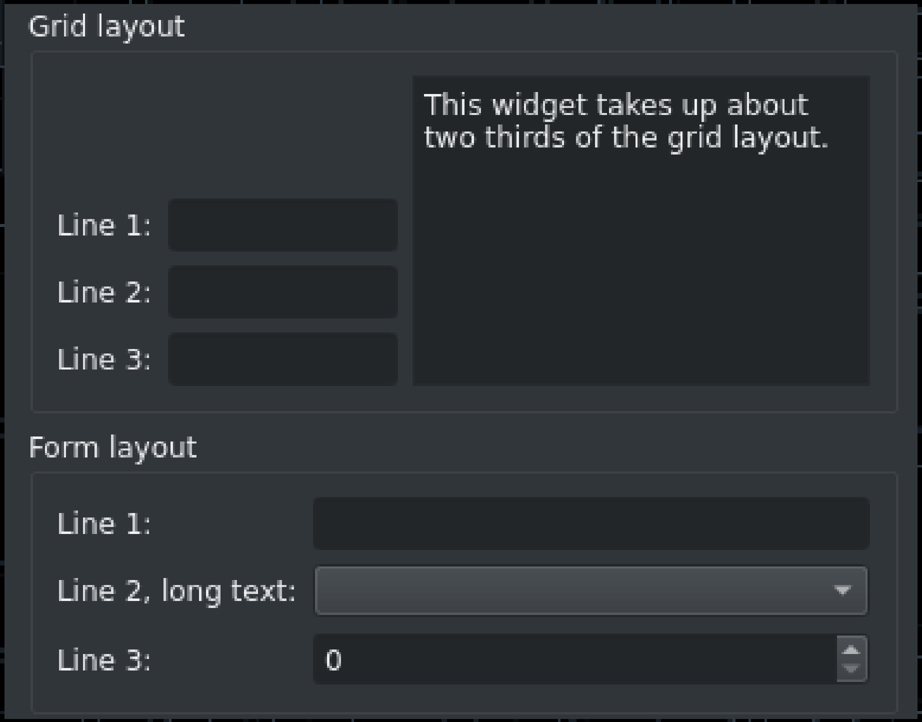


```
# Add the main VBOX layout
self.vbox = QVBoxLayout()
self.hbox = QHBoxLayout()
self.setLayout(self.vbox)

# Add buttons to hbox
for i in range(3):
    btn = QPushButton(f"Button {i}")
    self.hbox.addWidget(btn)

# Pack the Vbox
self.vbox.addLayout(self.hbox)
for i in range(5):
    lbl = QLabel(f"label {i}")
    self.vbox.addWidget(lbl)
self.vbox.addStretch(1) # <=== Keeps labels packed tight
```

- There is an .addStretch(int) that can help pack things via a stretch factor.

Layout Management: QGridLayout / QFormLayout

- Grid provides a column and rows system for adding widgets in a UI.
- Form uses 2 column rows for a label/input style of form.


Grid layout

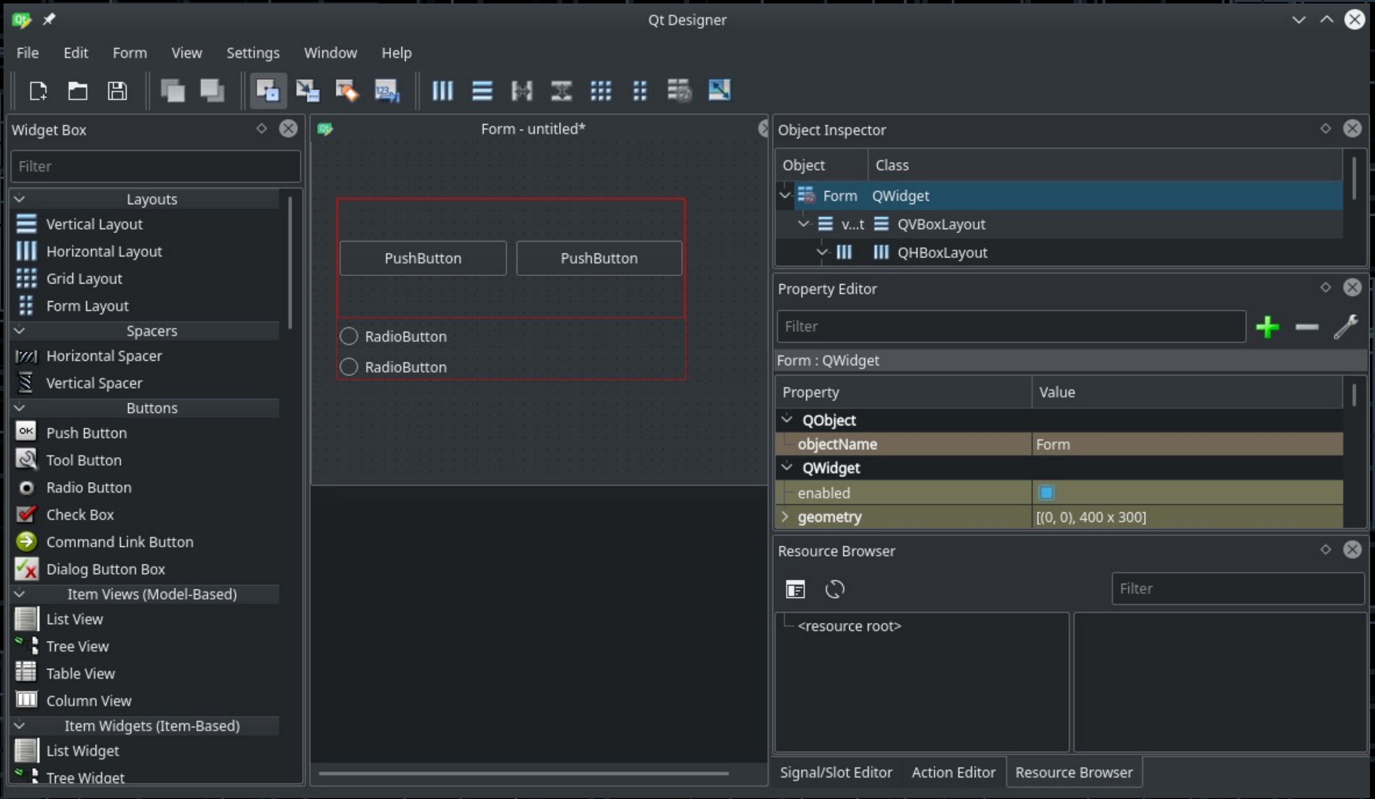
This widget takes up about two thirds of the grid layout.

Line 1:
Line 2:
Line 3:

Form layout

Line 1:
Line 2, long text:
Line 3: 0

QT Designer



Object	Class
Form	QWidget
v...t	QVBoxLayout
	QHBoxLayout

Property Editor

Form : QWidget

Property	Value
QObject	
objectName	Form
QWidget	
enabled	✓
geometry	[(0, 0), 400 x 300]

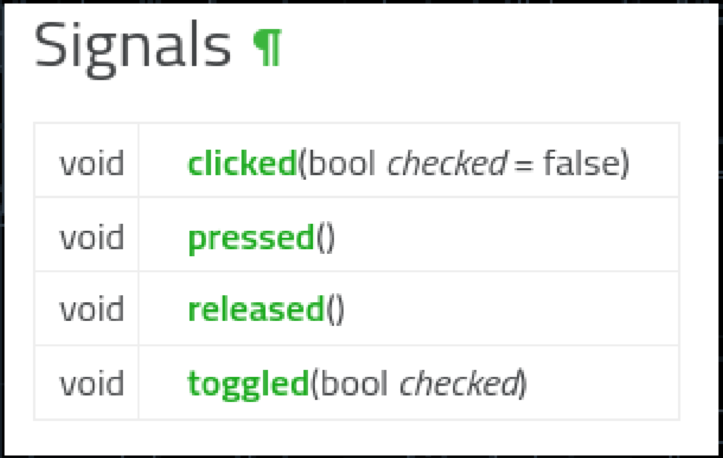
- Creates XML based *.ui files.
- Can load with uic.loadUI in PyQT5

Signals and Slots

- Signal and Slots are used for communication between objects in QT.
  - Signals are emitted from an object on events.
  - Slots can be connected to signals.
- For this talk, we will focus on signals mostly
- This is how we connect code to UI events

Signals and Widgets

- Documentation is your friend here
- Let’s look at the signals that a QAbstractButton (the parent class of QPushButton) can emit



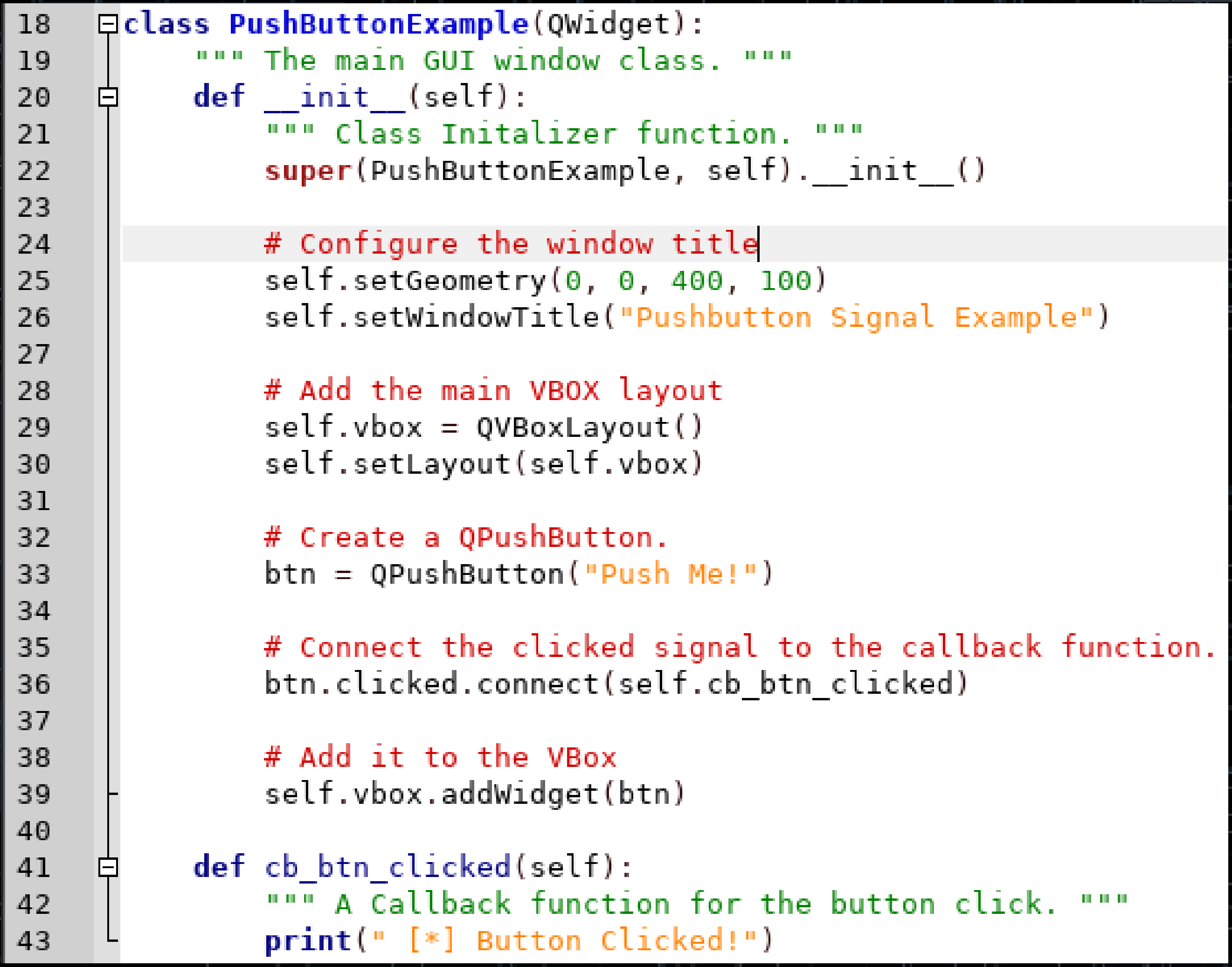
Signals ¶

void	clicked(bool *checked* = false)
void	pressed()
void	released()
void	toggled(bool *checked*)

Connecting to Signals

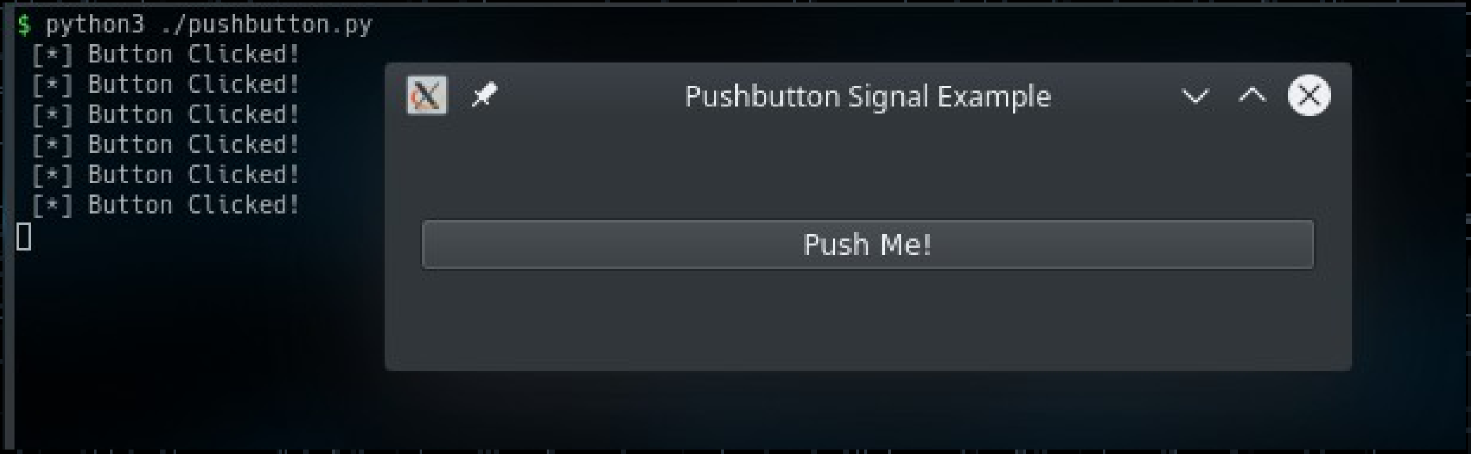
- Say we had a QPushButton and want to program it to do something.
- The steps would be
  - Create the button
  - Add it to your UI
  - Create a callback function
  - Connect the clicked signal to our callback function.

Connecting to Signals



```python
class PushButtonExample(QWidget):
    """ The main GUI window class. """
    def __init__(self):
        """ Class Initalizer function. """
        super(PushButtonExample, self).__init__()

        # Configure the window title
        self.setGeometry(0, 0, 400, 100)
        self.setWindowTitle("Pushbutton Signal Example")

        # Add the main VBOX layout
        self.vbox = QVBoxLayout()
        self.setLayout(self.vbox)

        # Create a QPushButton.
        btn = QPushButton("Push Me!")

        # Connect the clicked signal to the callback function.
        btn.clicked.connect(self.cb_btn_clicked)

        # Add it to the VBox
        self.vbox.addWidget(btn)

    def cb_btn_clicked(self):
        """ A Callback function for the button click. """
        print(" [*] Button Clicked!")
```

Connecting to Signals



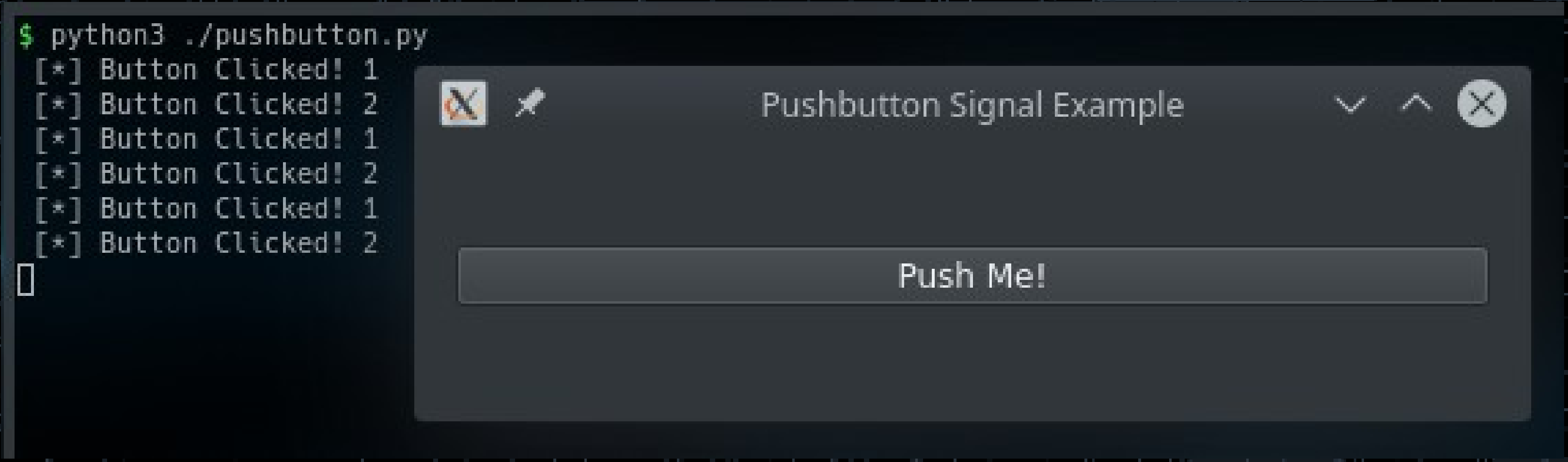
```
$ python3 ./pushbutton.py
 [*] Button Clicked!
 [*] Button Clicked!
 [*] Button Clicked!
 [*] Button Clicked!
 [*] Button Clicked!
 [*] Button Clicked!
```

Connecting to Signals

- You can also connect more than one callback.

```python
# Connect the clicked signal to the callback function.
btn.clicked.connect(self.cb_btn_clicked_1)
btn.clicked.connect(self.cb_btn_clicked_2)
```

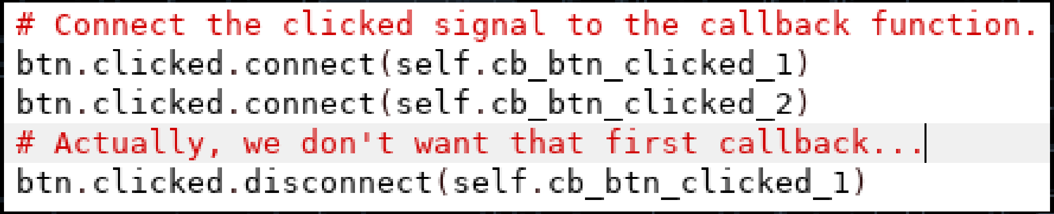
```python
def cb_btn_clicked_1(self):
    """ A Callback function for the button click. """
    print(" [*] Button Clicked! 1")

def cb_btn_clicked_2(self):
    """ A Callback function for the button click. """
    print(" [*] Button Clicked! 2")
```



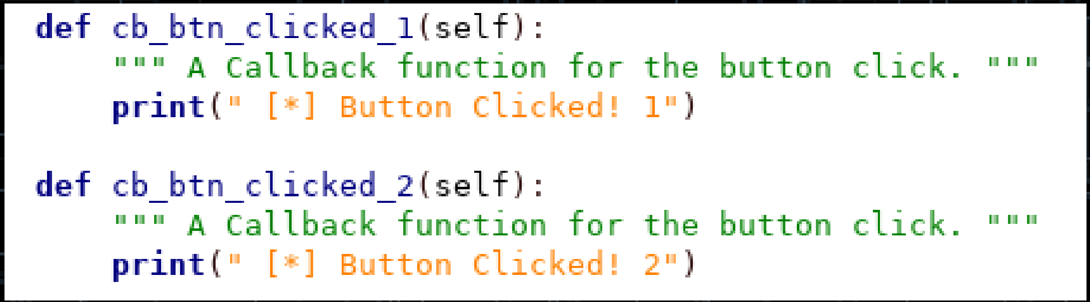
```
$ python3 ./pushbutton.py
 [*] Button Clicked! 1
 [*] Button Clicked! 2
 [*] Button Clicked! 1
 [*] Button Clicked! 2
 [*] Button Clicked! 1
 [*] Button Clicked! 2
```

Disconnecting to Signals

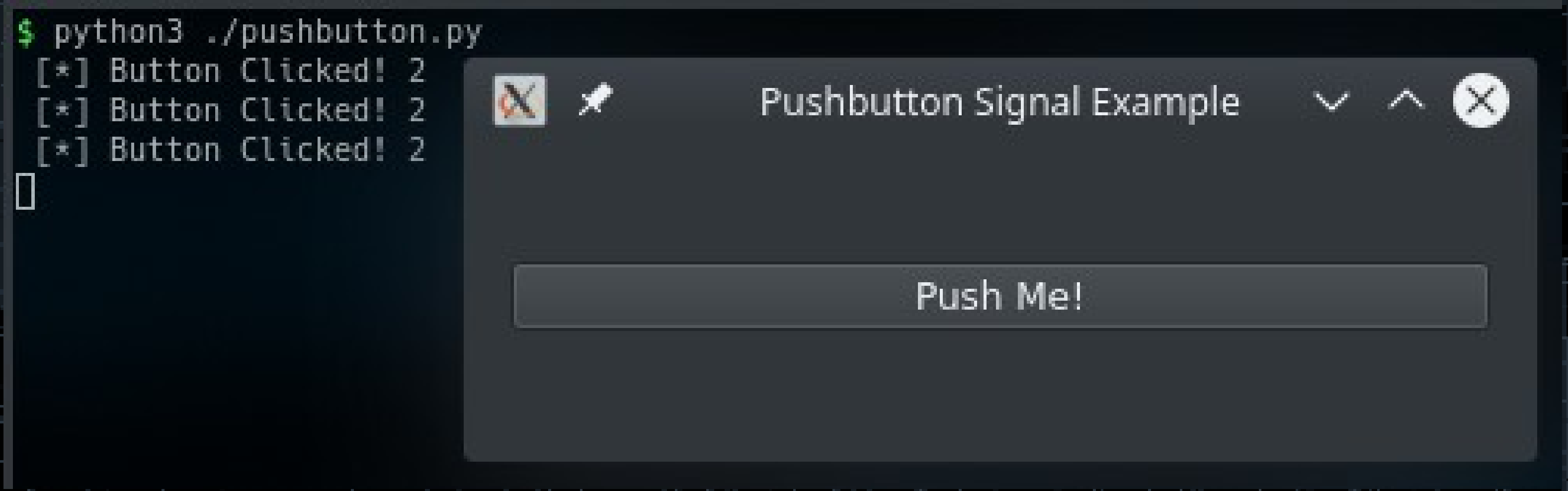
- You can also disconnect connected callbacks.



```python
# Connect the clicked signal to the callback function.
btn.clicked.connect(self.cb_btn_clicked_1)
btn.clicked.connect(self.cb_btn_clicked_2)
# Actually, we don't want that first callback...
btn.clicked.disconnect(self.cb_btn_clicked_1)
```



```python
def cb_btn_clicked_1(self):
    """ A Callback function for the button click. """
    print(" [*] Button Clicked! 1")

def cb_btn_clicked_2(self):
    """ A Callback function for the button click. """
    print(" [*] Button Clicked! 2")
```



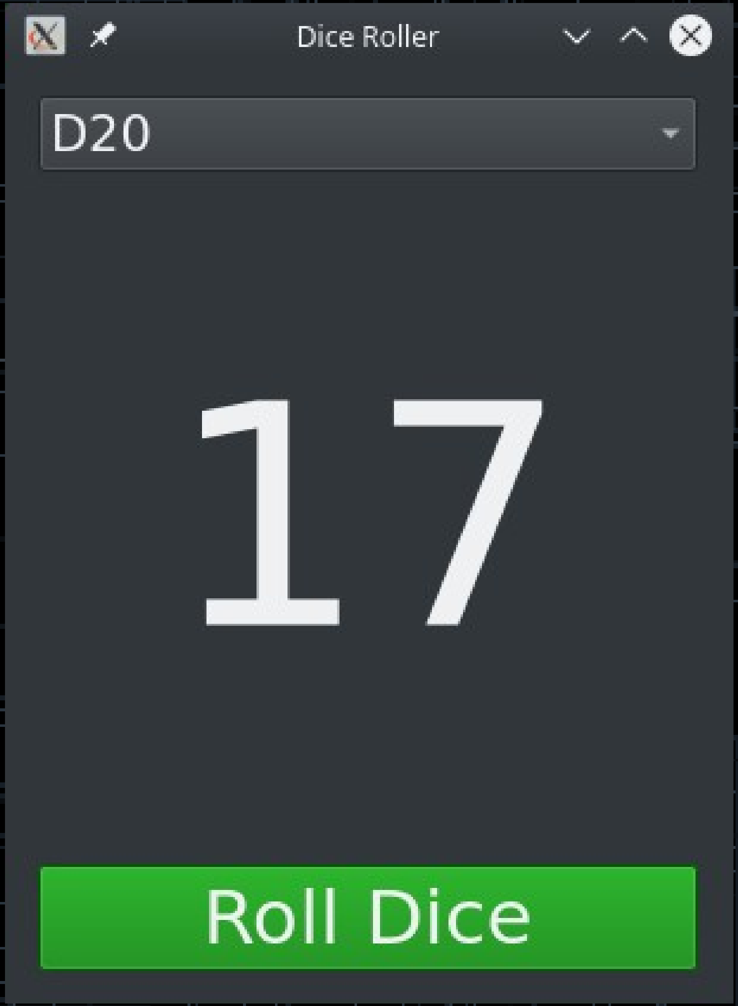
```
$ python3 ./pushbutton.py
 [*] Button Clicked! 2
 [*] Button Clicked! 2
 [*] Button Clicked! 2
```

Creating Your Own Signals

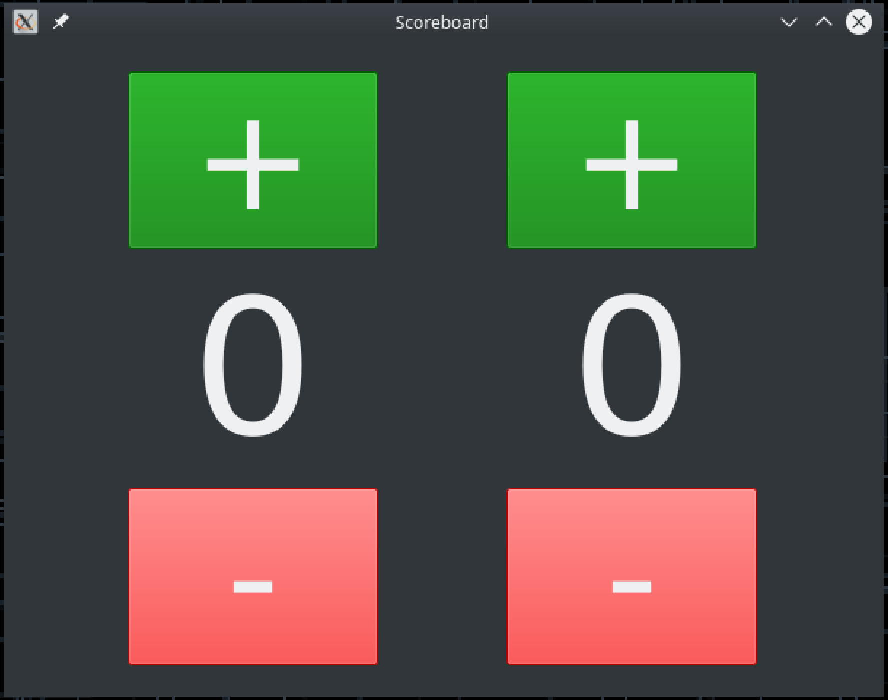
- You can create your own signals.
  - Useful for forms or custom widgets.
- For this you need to:
  - From PyQt5.QtCore import pyqtSignal
  - Give your class a signal
    - text_changed = pyqtSignal(bool, name="text_changed")
  - At some point, emit the signal
    - self.text_changed.emit(self.is_ready())

Examples: Dice

- Cryptographically secure dice roller
- Teaches:
  - QPushButton
  - QLabel
  - QComboBox
  - Connecting Signals
  - QRandomGenerator
  - Stylesheets

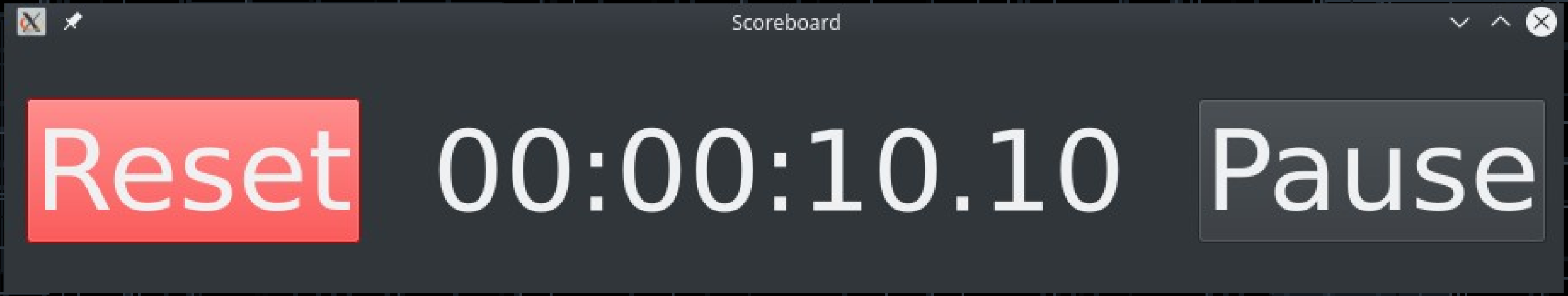


Examples: Scoreboard



- A Simple Scoreboard
- Teaches:
  - Classing widgets for easier reuse and management

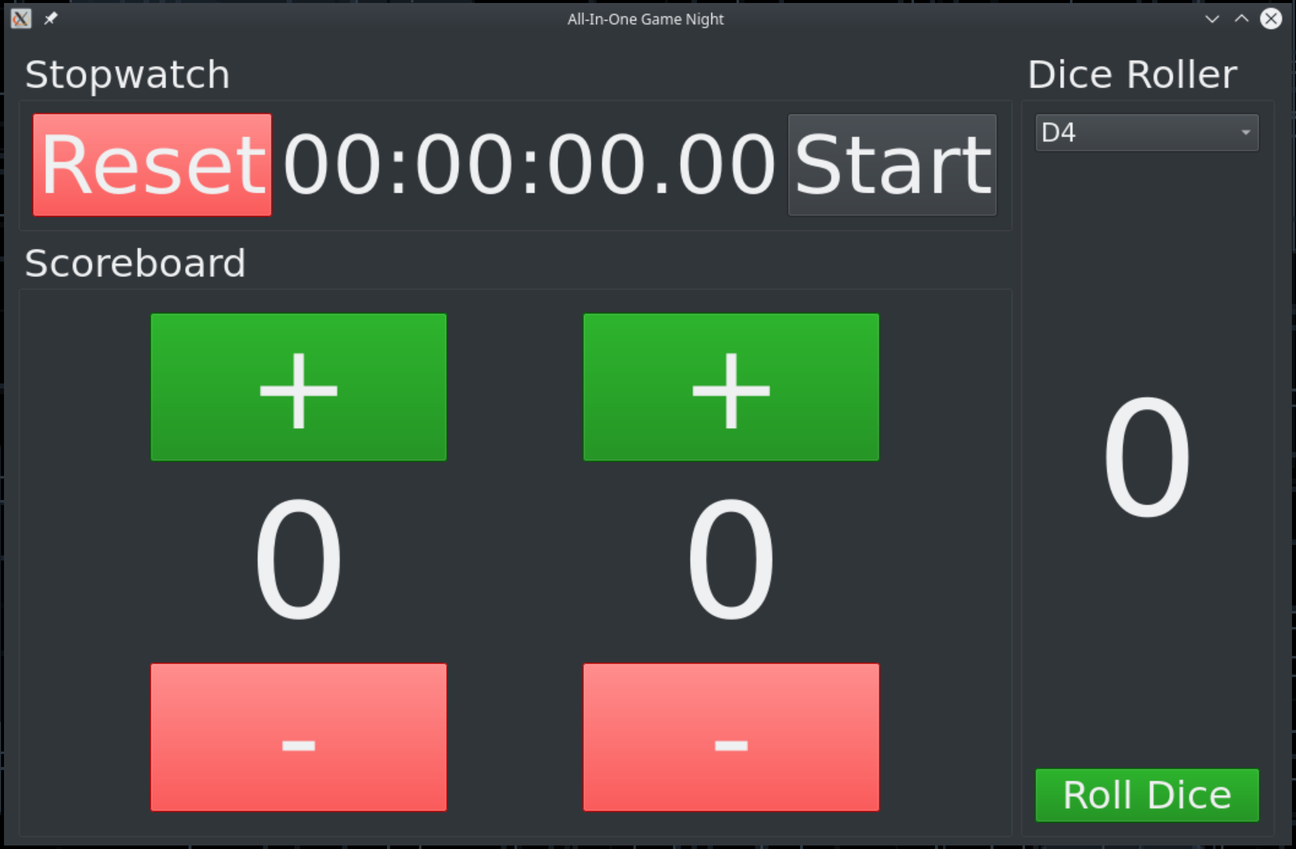
Examples: Stopwatch



- A Simple Scoreboard
- Teaches:
  - QDateTime
    - Formatting and DateTime Math
  - QTimer
    - Starting, Stopping, and Catching Timeouts

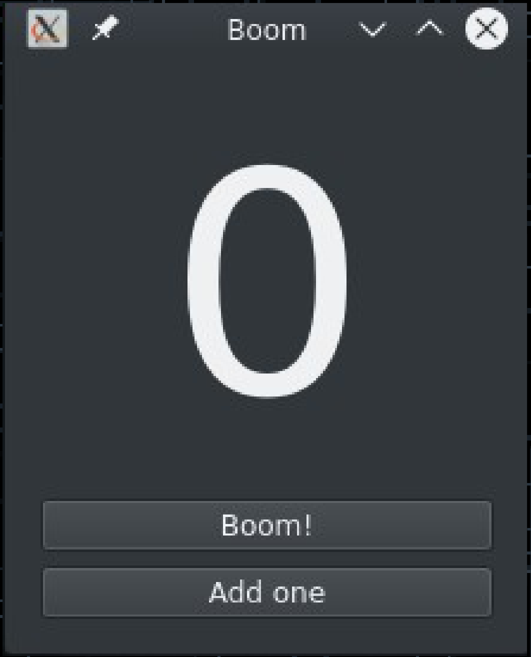
Examples: All-In-One

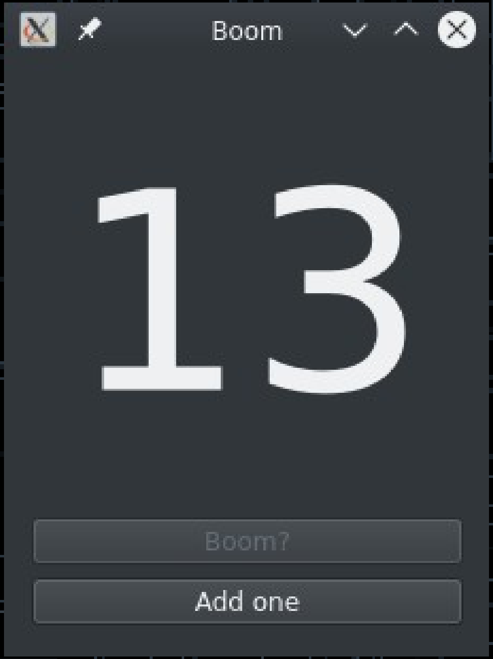
- Tie it all together with only slight mods.



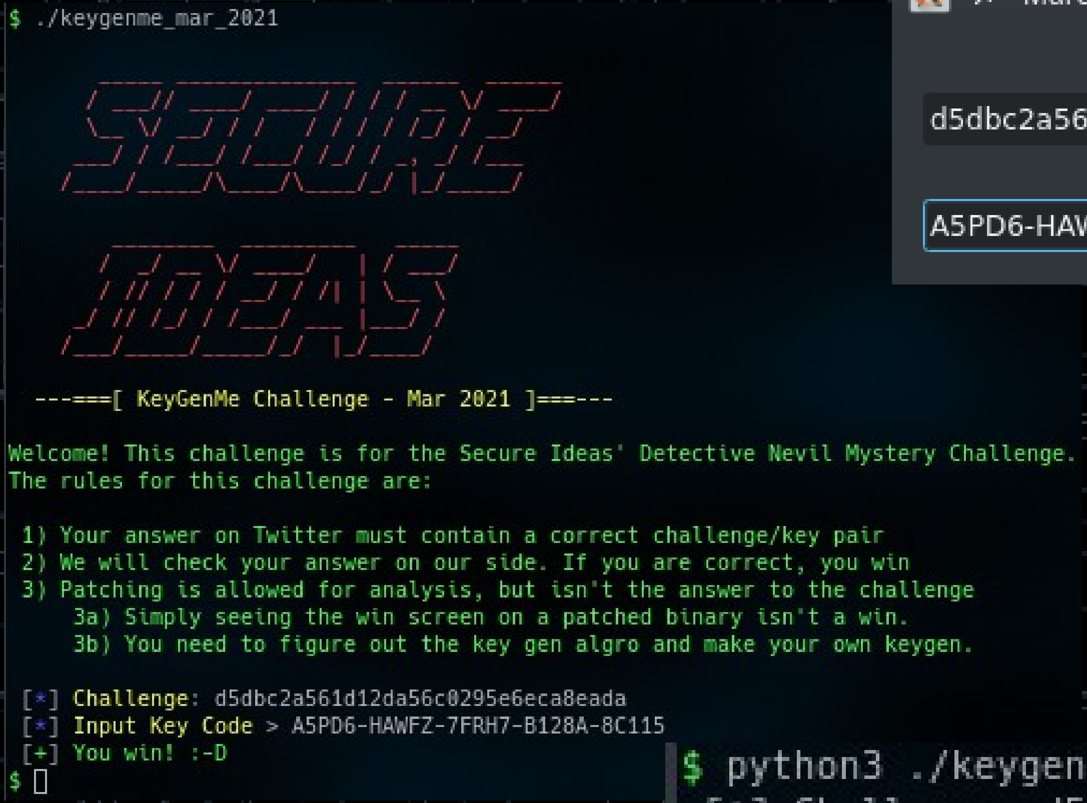
Examples: Boom and Boom_Fixed

- An example of why you don’t block the main GUI thread with sleep() vs QTimer

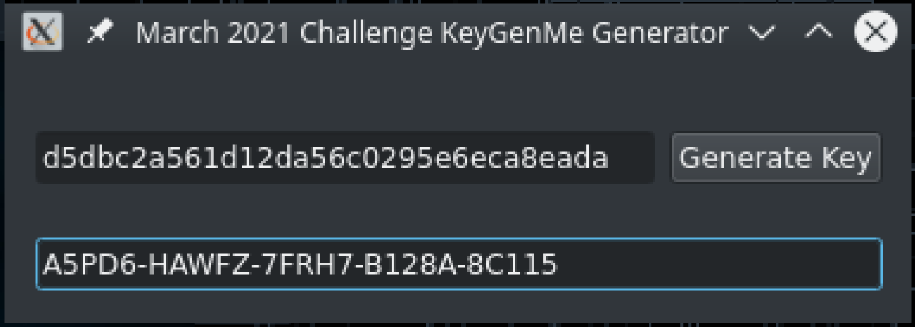




Examples: KeyGen Solution

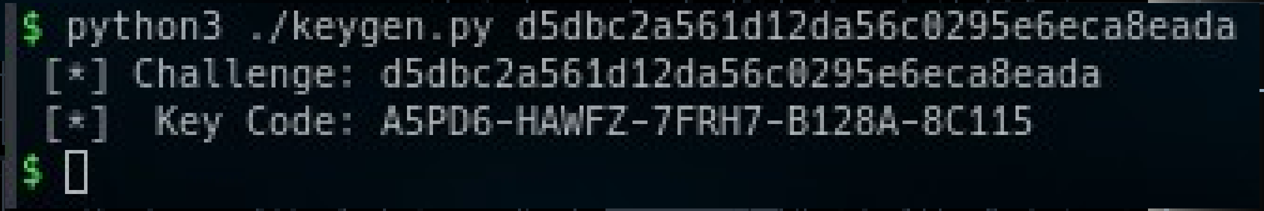


```
$ ./keygenme_mar_2021

  ---===[ KeyGenMe Challenge - Mar 2021 ]===---

Welcome! This challenge is for the Secure Ideas' Detective Nevil Mystery Challenge.
The rules for this challenge are:

 1) Your answer on Twitter must contain a correct challenge/key pair
 2) We will check your answer on our side. If you are correct, you win
 3) Patching is allowed for analysis, but isn't the answer to the challenge
     3a) Simply seeing the win screen on a patched binary isn't a win.
     3b) You need to figure out the key gen algro and make your own keygen.

 [*] Challenge: d5dbc2a561d12da56c0295e6eca8eada
 [*] Input Key Code > A5PD6-HAWFZ-7FRH7-B128A-8C115
 [+] You win! :-D
$ 
```



d5dbc2a561d12da56c0295e6eca8eada

A5PD6-HAWFZ-7FRH7-B128A-8C115



```
$ python3 ./keygen.py d5dbc2a561d12da56c0295e6eca8eada
 [*] Challenge: d5dbc2a561d12da56c0295e6eca8eada
 [*]  Key Code: A5PD6-HAWFZ-7FRH7-B128A-8C115
$ 
```

- GUI and CLI tool

Best Practices

- Try to keep Code and UI Separated the best you can
- Offload heavy workloads to other threads or use QT build in functions
  - QProcess, Qtimer, etc
- I Usually suggest *NOT* customizing UI skins to much…
  - If you do COVER EVERY ASPECT
    - font color, font style, font size, background colors, etc.

Q&A

- **GitHub:**
  - https://github.com/jaxhax-travis
- **Documentation:**
  - https://doc.qt.io/qtforpython-5/api.html
- **Examples:**
  - sudo apt install pyqt5-examples
    - /usr/share/doc/pyqt5-examples/