CALCULATOR APP

Putting your knowledge of QtWidgets to the test as we build an interactive Calculator App with PyQt

ZERO TO KNOWING

The Code Burger

ZERO TO KNOWING

1. All **Imports**
2. Main App Objects and **Settings**
3. **Create all Widgets** needed in App
4. **Design** your Layout, add your widgets to the screen
5. **Set the final layout** to the Main window
6. **Show** and **Execute** your app

Working with Layouts

ZERO TO KNOWING

Layouts you know:

- QVBoxLayout
- QHBoxLayout

QV -> V -> Vertical -> Column

QH -> H -> Horizontal -> Row

Working with Layouts

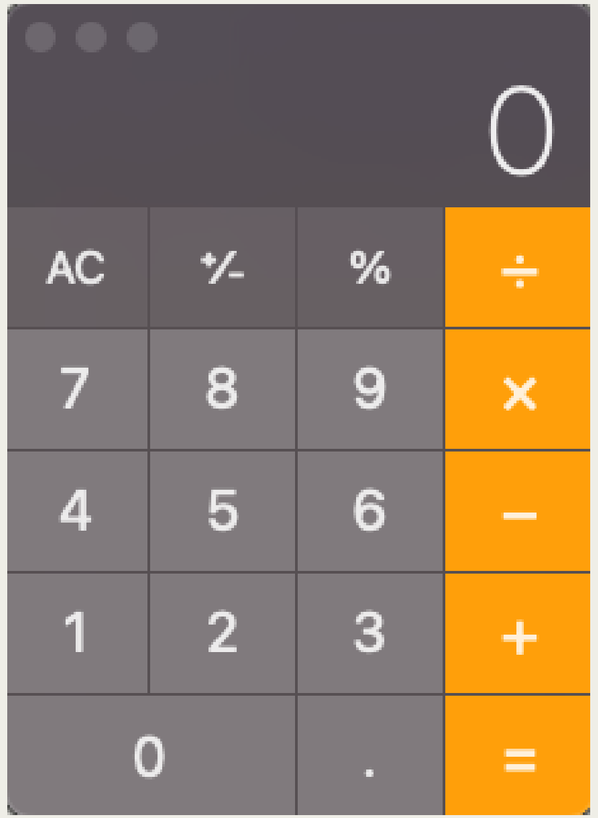
ZERO TO KNOWING

Layouts you know:

- QVBoxLayout
- QHBoxLayout

How would you design a calculator?

Rows and Columns?



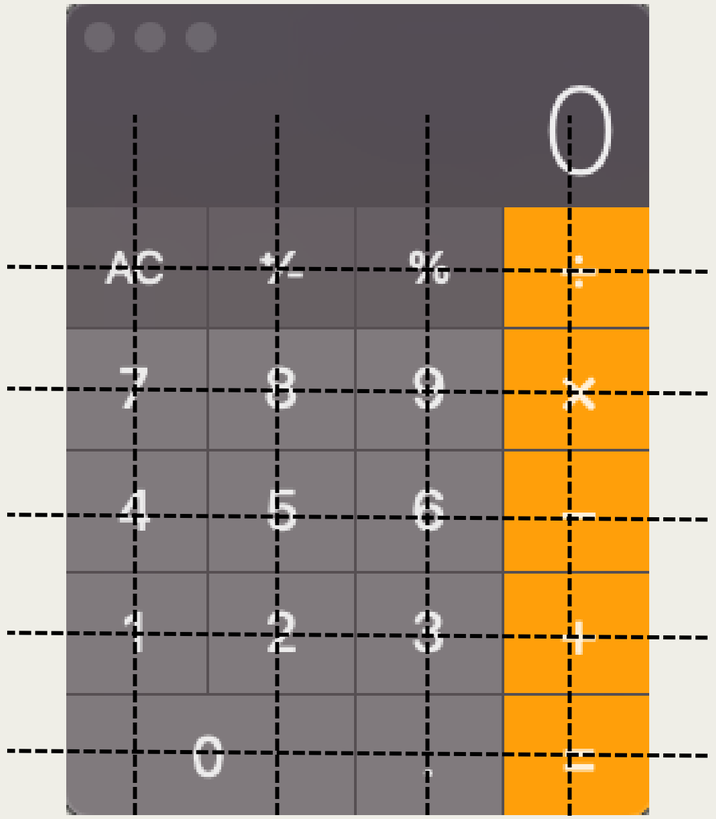
Working with Layouts

ZERO TO KNOWING

Layouts you know:

- QVBoxLayout
- QHBoxLayout
- QGridLayout

We have a Layout made for this, QGridLayout




```
grid = QGridLayout()
```

Working with Layouts

ZERO TO KNOWING

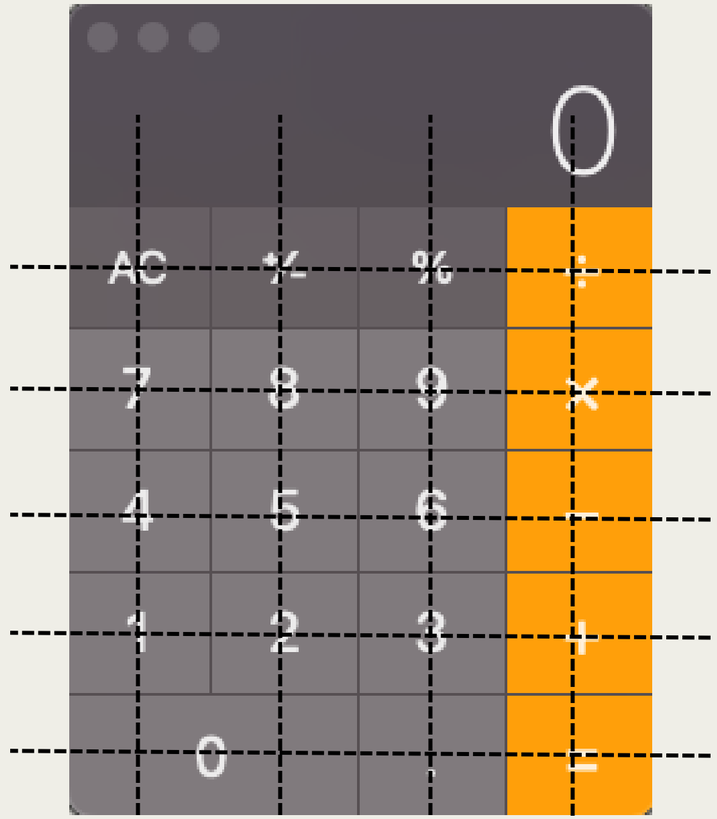
```
grid = QGridLayout()

grid.addWidget(button, row, col)
```

Layout Add This object to This row In this column

For this to work we need
a row and column index:

row = 2
col = 1

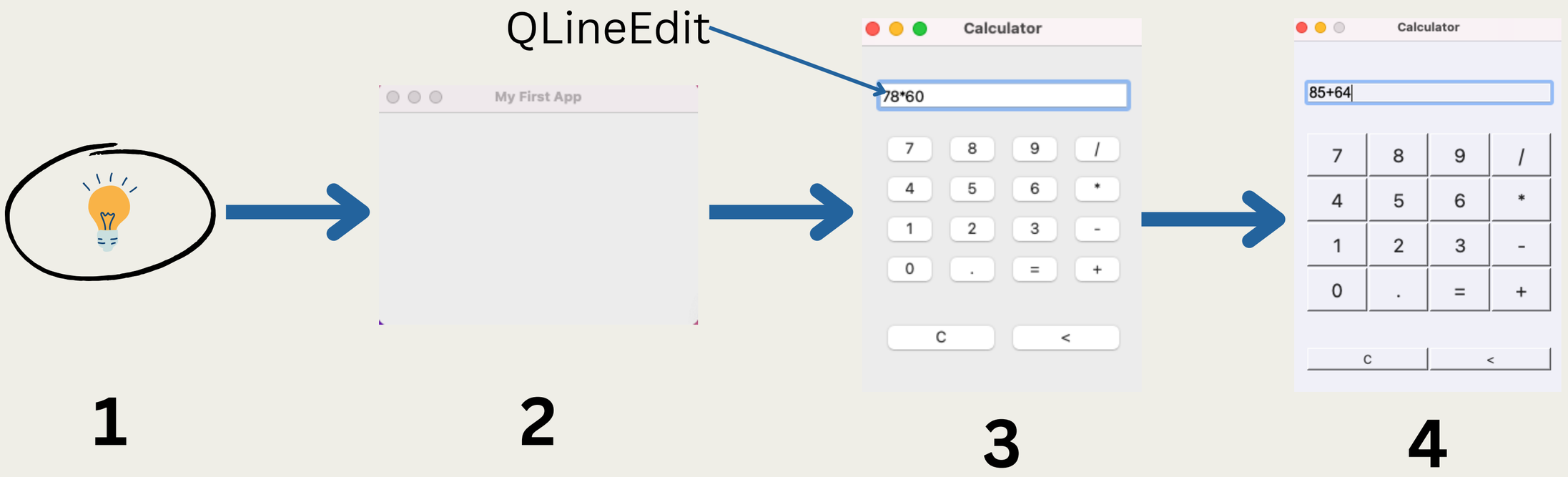


Inital App Design and Setup

ZERO TO KNOWING

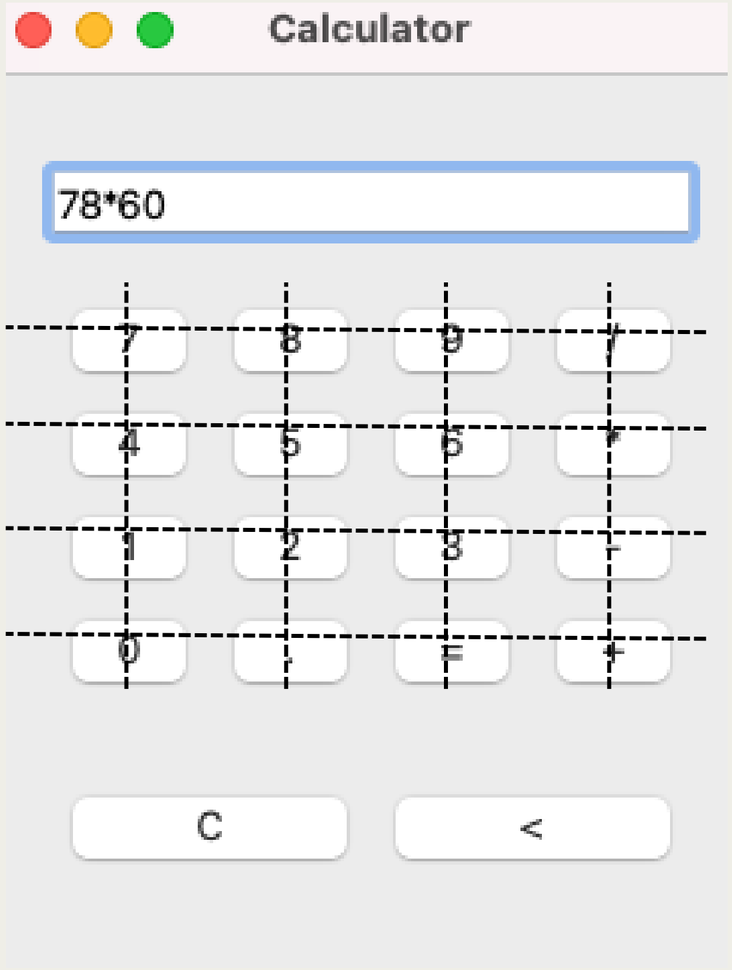
How can we go from One to Three ~

Only **focus on our Design**



Inital App Design and Setup

ZERO TO KNOWING

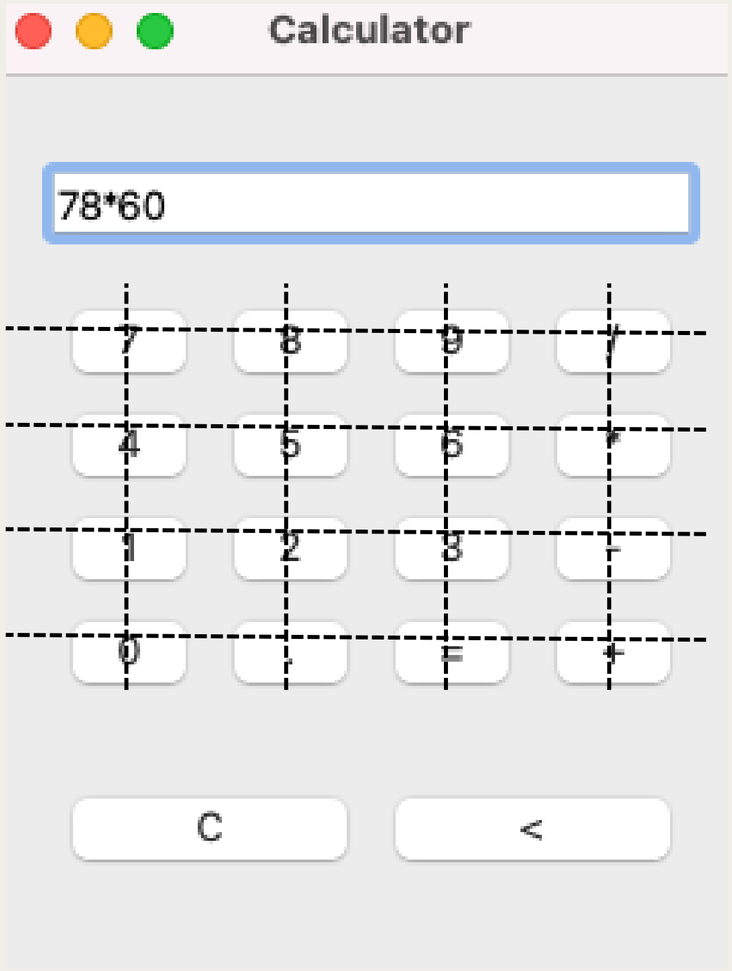


This whole grid thing seems like it will take a while

That's **17 Buttons**

Inital App Design and Setup

ZERO TO KNOWING



Let's repeat the task:

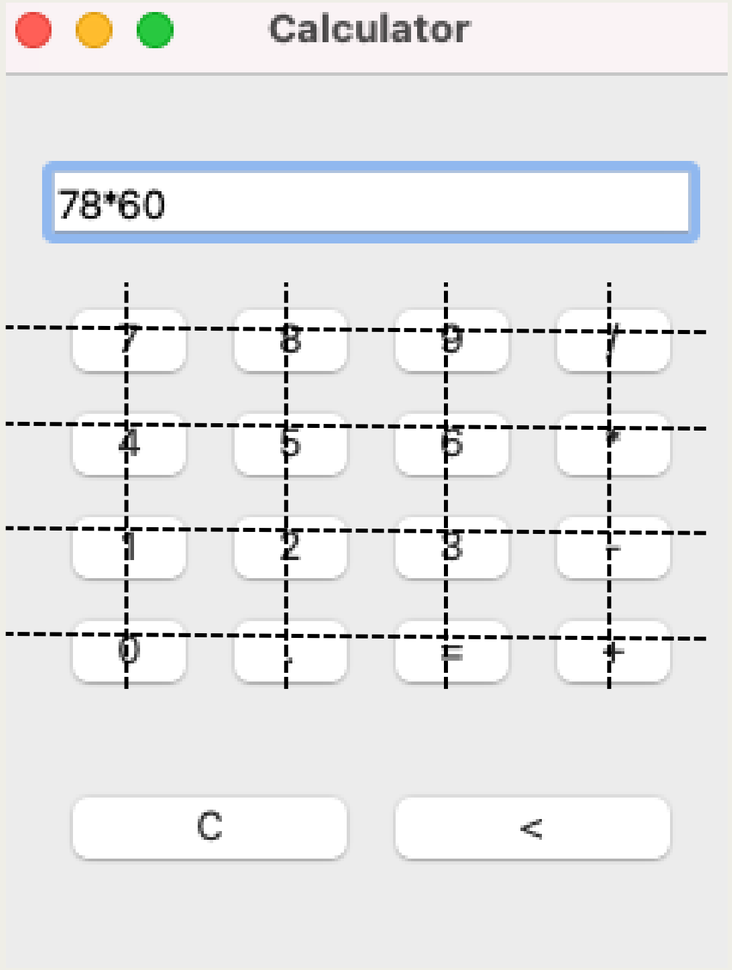
for button_text in a list of buttons:

1. **Create** a new **QPushButton**(button_text)
2. **Create** an **event** for that button
3. **Add** the **Button** to our **QGridLayout**
4. **Increase** row and column by 1

How can you achieve this? Think
Counter variables and Lists

Inital App Design and Setup

ZERO TO KNOWING




```
row = 0
col = 0

for button_text in buttons:
    button = QPushButton(button_text)
    button.clicked.connect(button_click)
    buttons_grid.addWidget(button, row, col)

    col += 1

    if col > 3:
        col = 0
        row += 1
```

Creating Functionality within our App

Learn how to evalute our expressions

ZERO TO KNOWING

Creating Functionality

```
def button_click():
    button = app.
    text = button.

    if text == "=":
        symbol = text_box.
        res =
        text_box.setText( str( res ))

    elif text == "C":
        text_box.

    elif text == "<":
        current_text = text_box.
        text_box.setText(
    else:
        current_text = text_box.
        text_box.setText( current_text + text )
```

This is the **only function required** to run our app

if the **text of the button** is "**=**", then we **set "="** as the **value** to the variable symbol

We the need to **evaluate our expression** and update our QLineEdit output

We need to **clear** ourQLineEdit (input)

We need to **cut the last element** off our current_text

How could we solve any of these?

Creating Functionality

```
def button_click():
    button = app.
    text = button.

    if text == "=":
        symbol = text_box.
        res =
        text_box.setText( str( res ))

    elif text == "C":
        text_box.

    elif text == "<":
        current_text = text_box.
        text_box.setText(
    else:
        current_text = text_box.
        text_box.setText( current_text + text )
```

Let's start with **getting the value of the button clicked**

More specifically the t**ext value** of the button clicked

Do you remember from the first app how we did this?

Creating Functionality

```
def button_click():
    button = app.[redacted])
    text = button.text()

    if text == "=":
       symbol = text_box.text()
        res = [redacted])
        text_box.setText( str( res ))

    elif text == "C":
        text_box.[redacted])

    elif text == "<":
       current_text = text_box.text()
       text_box.setText([redacted]])
    else:
       current_text = text_box.text()
       text_box.setText( current_text + text )
```

Now we need to **evaluate the expression** we **collected from the input field**

Any thoughts on how we can "evaluate"?

.text() method

Linked to an object and **gets the text value of that object**

Creating Functionality

ZERO TO KNOWING

```
def button_click():
    button = app.[redacted])
    text = button.text()

    if text == "=":
       symbol = text_box.text()
        res = eval( symbol )
        text_box.setText( str( res ))

    elif text == "C":
        text_box.[redacted])

    elif text == "<":
       current_text = text_box.text()
       text_box.setText([redacted]])
    else:
       current_text = text_box.text()
       text_box.setText( current_text + text )
```

If we **press the "C" button**, we want to **delete/clear** everything in the input field

Any thoughts on how we can achieve this ?

Python **eval**() function

Evaluates an expression. If the expression is valid/allowed, it'll run

Creating Functionality

ZERO TO KNOWING

```
def button_click():
    button = app.[redacted]
    text = button.text()

    if text == "=":
       symbol = text_box.text()
        res = eval( symbol )
        text_box.setText( str( res ))

    elif text == "C":
        text_box.clear()

    elif text == "<":
       current_text = text_box.text()
       text_box.setText([redacted])
    else:
       current_text = text_box.text()
       text_box.setText( current_text + text )
```

If we **press the "<" button**, we want to **delete the last item** entered

Any thoughts on how we can **cut** an element from a string?

PyQt **.clear**() method

Linked to an object, it'll **clear it's current value**

Creating Functionality

ZERO TO KNOWING

```
def button_click():
    button = app.[redacted]
    text = button.text()

    if text == "=":
       symbol = text_box.text()
        res = eval( symbol )
        text_box.setText( str( res ))

    elif text == "C":
        text_box.clear()

    elif text == "<":
       current_text = text_box.text()
       text_box.setText(current_text[ :-1 ])
    else:
       current_text = text_box.text()
       text_box.setText( current_text + text )
```

All the buttons have the same "Event Target".
We need a **way to tell them apart**

Luckily PyQt has us covered

We **take our current string value** of the input field and **use the brackets to index the last position** minus one

Creating Functionality

ZERO TO KNOWING

```
def button_click():
    button = app.sender()
    text = button.text()

    if text == "=":
       symbol = text_box.text()
        res = eval( symbol )
        text_box.setText( str( res ))

    elif text == "C":
        text_box.clear()

    elif text == "<":
       current_text = text_box.text()
       text_box.setText(current_text[ :-1 ])
    else:
       current_text = text_box.text()
       text_box.setText( current_text + text )
```

All the buttons have the same "Event Target".
We need a **way to tell them apart**

.sender() method

Used to facilitate differentiation of multiple event sources connected to the same event target

Creating Functionality

ZERO TO KNOWING

```
def button_click():
    button = app.sender()
    text = button.text()

    if text == "=":
       symbol = text_box.text()
       try:
            res = eval( symbol )
            text_box.setText( str( res ))

       except Exception as e:
            text_box.setText( "Error" )

    elif text == "C":
        text_box.clear()

    elif text == "<":
       current_text = text_box.text()
       text_box.setText(current_text[ :-1 ])
    else:
       current_text = text_box.text()
       text_box.setText( current_text + text )
```

As a final precaution we will throw a try/except statement in here to ensure we don't try to evalute an empty or wrong expression

Remaster Challenges

ZERO TO KNOWING

- Create a Class based application. **Build your own class** and **refactor your code so everything is within One Class**

- Add some design and styles to your app! Hint ->Check out the **module QFont** and how to use the **.setStyleSheet method**