pyside

Live de Python #208

Roteiro

1. QT

Um pouco da história e uma sopa de letrinhas

2. pyside

Um básico sobre a biblioteca, Gui, Core e Widgets

3. Qt design

Arrastando e soltando interfaces

4. QT quick e QML

Uma linguagem descritiva para interfaces

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Acássio Anjos, Ademar Peixoto, Alexandre Harano, Alexandre Souza, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Ana Padovan, André Rocha, Aquiles Coutinho, Arnaldo Turque, Aurelio Costa, Bloquearsites Farewall, Bruno Barcellos, Bruno Freitas, Bruno Guizi, Bruno Oliveira, Bruno Ramos, Caio Nascimento, Carlos Alipio, Christiano Morais, Clara Battesini, Daniel Freitas, Daniel Haas, Danilo Segura, Dartz Dartz, David Kwast, Delton Porfiro, Dhyeives Rodovalho, Diego Farias, Diego Guimarães, Dilenon Delfino, Dino Aguilar, Diogo Paschoal, Douglas Bastos, Douglas Braga, Douglas Zickuhr, Eli Júnior, Emerson Rafael, Érico Andrei, Eugenio Mazzini, Euripedes Borges, Evandro Avellar, Everton Silva, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Felipe Rodrigues, Fernanda Prado, Fernando Rozas, Fernando Sousa, Flávio Meira, Flavkaze Flavkaze, Franklin Silva, Gabriel Barbosa, Gabriel Simonetto, Geab Duarte, Geandreson Costa, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Gustavo Dettenborn, Gustavo Suto, Heitor Fernandes, Henrique Junqueira, Igor Taconi, Ismael Ventura, Israel Gomes, Italo Silva, Jair Andrade, Jairo Lenfers, Janael Pinheiro, João Lugão, João Rodrigues, Joelson Sartori, Johnny Tardin, Jonatas Leon, Jonatas Oliveira, Jônatas Silva, Jose Mazolini, Juan Gutierrez, Juliana Machado, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Botassio, Leandro Miranda, Leonardo Mello, Leonardo Nazareth, Lucas Adorno, Lucas Mello, Lucas Mendes, Lucas Oliveira, Lucas Polo, Lucas Teixeira, Lucas Valino, Luciano Silva, Luciano Teixeira, Luiz Junior, Luiz Lima, Luiz Paula, Luiz Perciliano, Maiquel Leonel, Marcelino Pinheiro, Marcelo Matte, Márcio Martignoni, Marco Mello, Marcos Gomes, Marco Yamada, Maria Clara, Marina Passos, Mario Deus, Mateus Lisboa, Matheus Silva, Matheus Vian, Mírian Batista, Murilo Andrade, Murilo Cunha, Murilo Viana, Natan Cervinski, Nicolas Teodosio, Osvaldo Neto, Otávio Barradas, Patricia Minamizawa, Patrick Felipe, Paulo Braga, Paulo Tadei, Pedro Duarte, Pedro Henrique, Pedro Pereira, Peterson Santos, P Muniz, Priscila Santos, Rafael Lopes, Rafael Rodrigues, Rafael Romão, Ramayana Menezes, Reinaldo Silva, Renato Veirich, Ricardo Silva, Riverfount Riverfount, Robson Maciel, Rodrigo Alves, Rodrigo Brandao, Rodrigo Ferreira, Rodrigo Freire, Rodrigo Vaccari, Rodrigo Vieira, Rogério Sousa, Ronaldo Silva, Rui Jr, Samanta Cicilia, Sara Selis, Thalysson Bogéa, Thiago Araujo, Thiago Borges, Thiago Bueno, Thiago Curvelo, Thiago Moraes, Thiago Oliveira, Thiago S, Thiago Souza, Tiago Minuzzi, Tony Dias, Valdir Tegon, Victor Wildner, Vinícius Bastos, Vítor Gomes, Vitor Luz, Vlademir Souza, Vladimir Lemos, Walter Reis, Wellington Abreu, Wesley Mendes, William Alves, Willian Lopes, Wilson Neto, Yury Barros

Obrigado você

A sopa de letrinhas

QT

Um pouco de história

- Nascimento em 1991, criado pela Trolltech
- Escrito originalmente em C++
- Originalmente desenvolvido para Windows e Linux
  - Licença aberta somente para Linux
- Se pronuncia *cute*. Que é uma em inglês que significa "bonitinho", "fofo"
- Foi chamado QT porque a letra Q ficava bonita no emacs
- Portado para o MacOS em 2001 (QT 3.0 - GPL)
- 2008 foi comprado pela Nokia
  - Software Livre para todas as plataformas
- Atualmente na versão 6.3, lançada em 2022

https://doc.qt.io/qtforpython-6/modules.html

ABIR ESSA PÁGINA, NÃO ESQUECER!!!

Portes para o Python

Existem dois portes famosos para o python da biblioteca QT:

- PyQT: Criado em 1998
  - Licença GPL / Dual
  - Desenvolvido por Riverbank Computing
- PySide: Criado em 2009
  - Licença LGPL
  - Desenvolvido pela QT Company

Basicamente, se você faz software livre ou de código aberto, você pode escolher qualquer uma das duas.

Se você desenvolve software comercial, o PyQT sob GPL, pede que seu código seja compartilhado com os clientes ou que você compre uma licença de 550 dólares por pessoa envolvida no desenvolvimento.

Popularidade dos portes

Embora os dois portes sejam muito populares, em 2009, após a aquisição da trolltech pela Nokia. O QT foi lançado sob LGPL e houveram desentendimento entre a Riverbank e a QT company sobre o licenciamento do PyQT. Fazendo a QT company lançar seu próprio porte sob licença LPGL, o PySide.

Em 2009 os projetos era bem parecidos, porém, com o lançamento do QT 5, em 2013 a velocidade de adaptação dos portes foi MUITO diferente

- PyQT 5 foi lançado em 2016, três anos após o QT 5
- PySide 2, com suporte ao QT 5 foi lançado em 2018. Cinco anos após o lançamento do QT 5

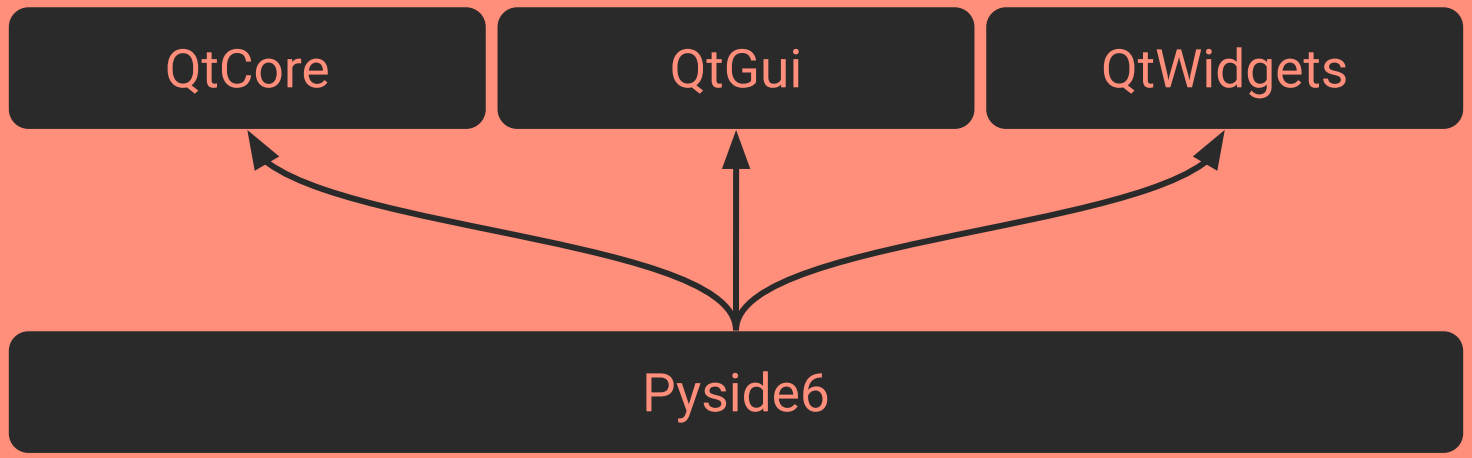
PySide

Um entendimento básico

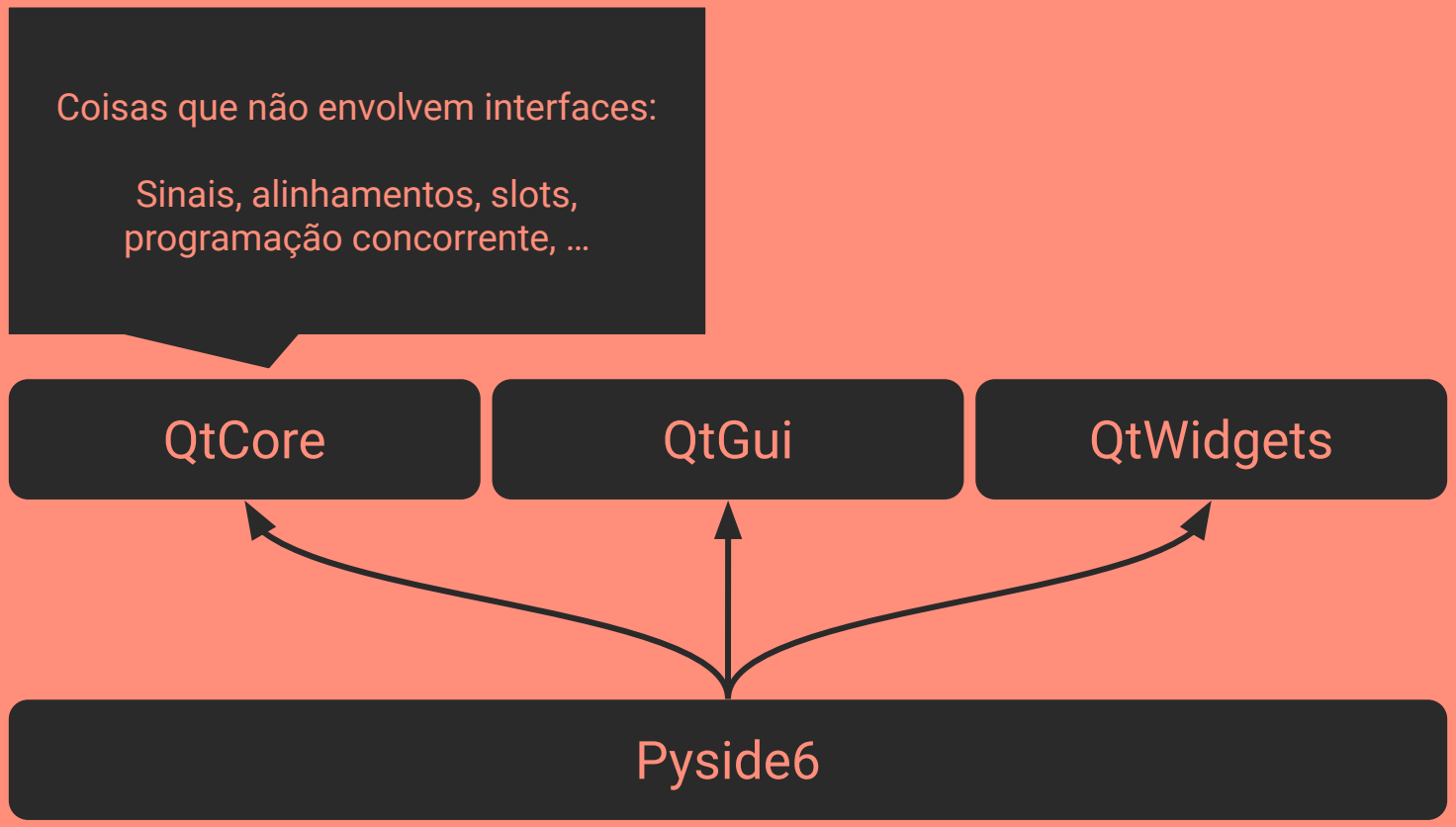
pip install pyside6

Instalando...

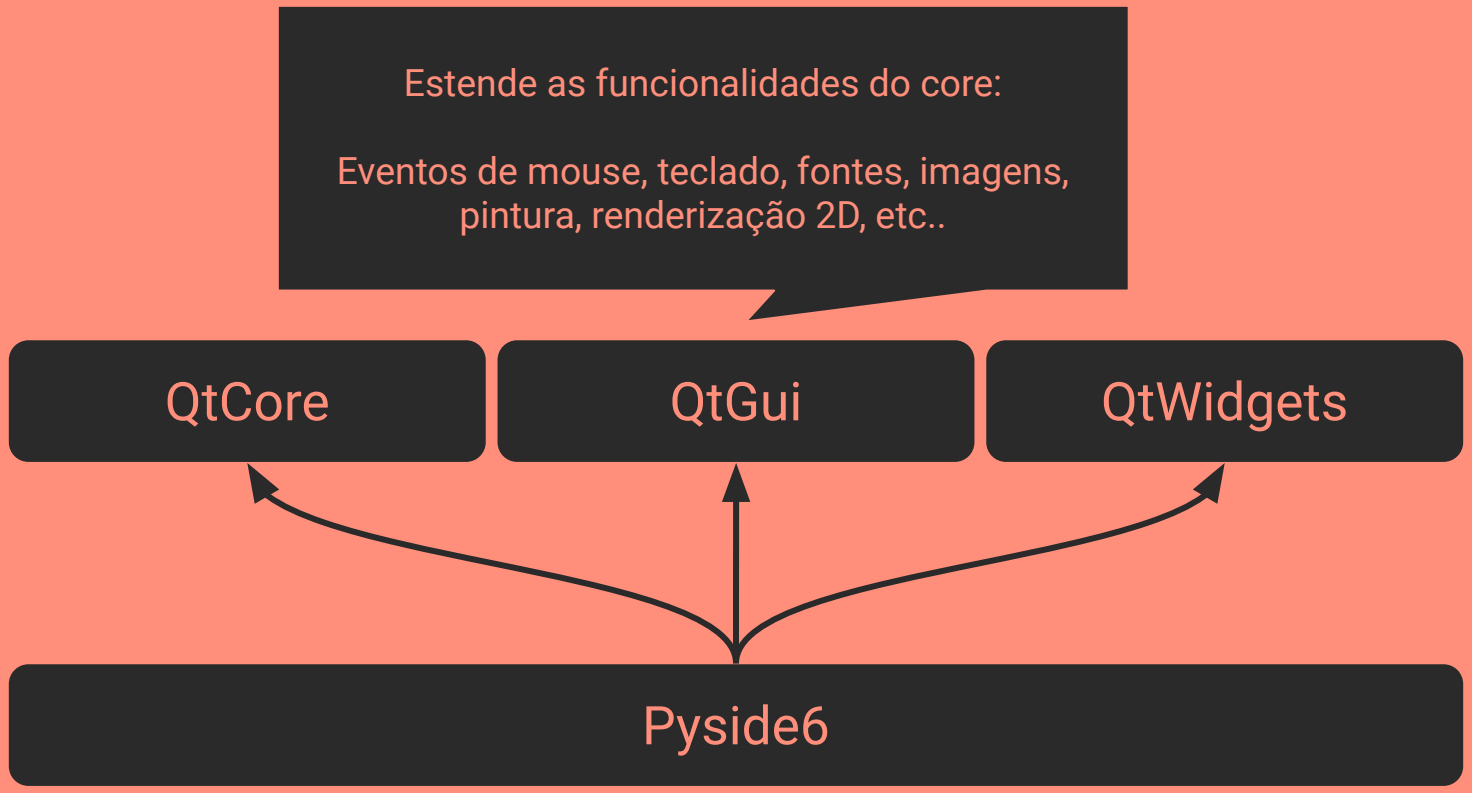
Os três componentes principais do pyside



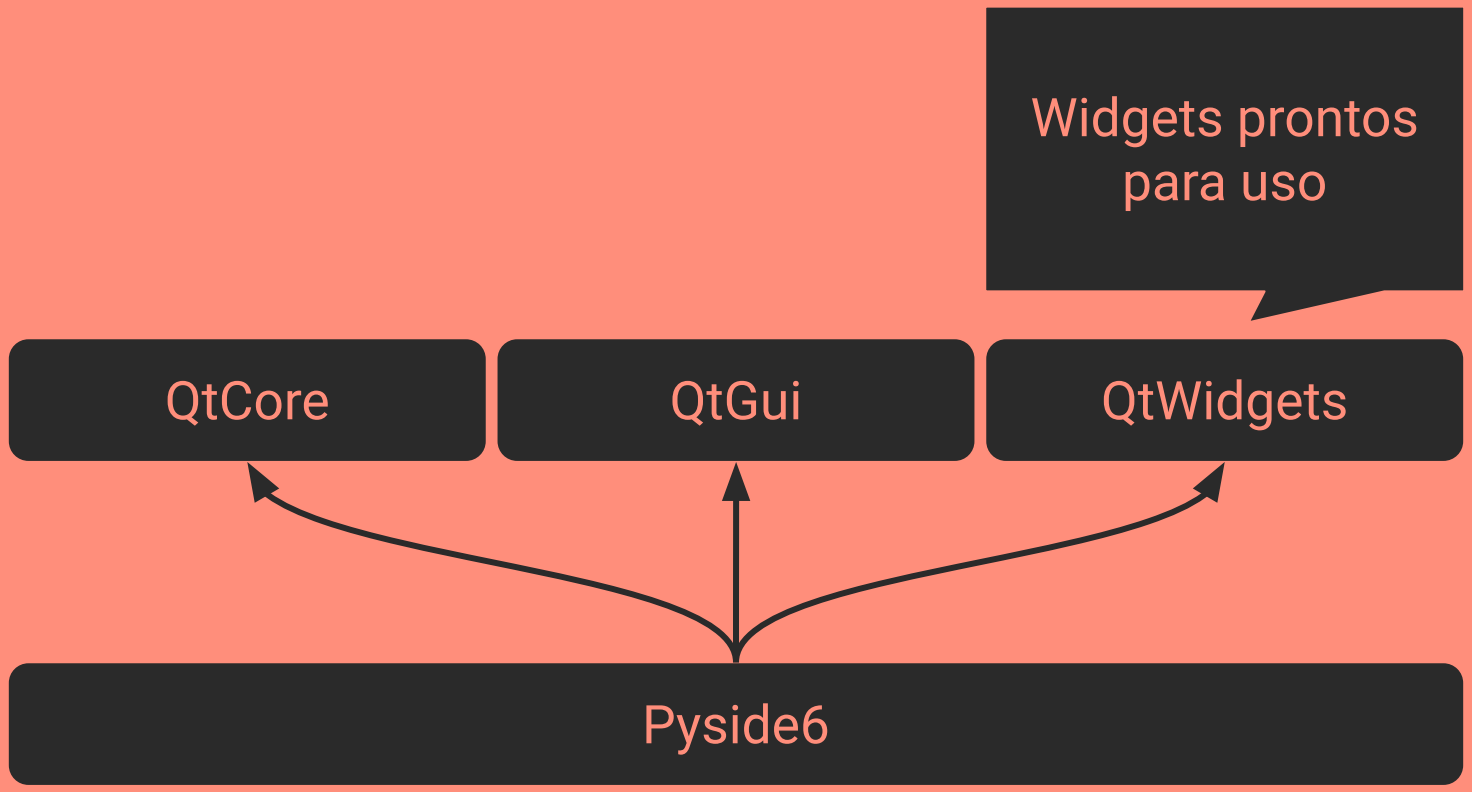
Os três componentes principais do pyside



Os três componentes principais do pyside



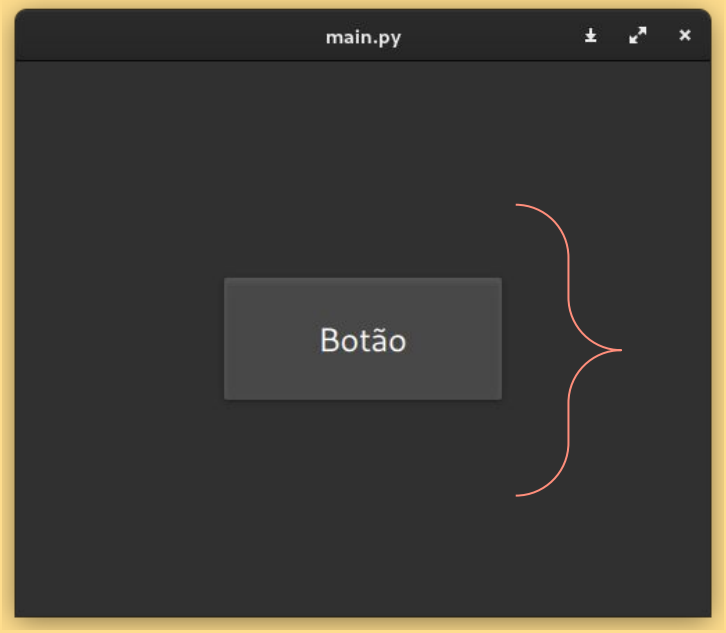
Os três componentes principais do pyside



Afinal, o que são widgets?

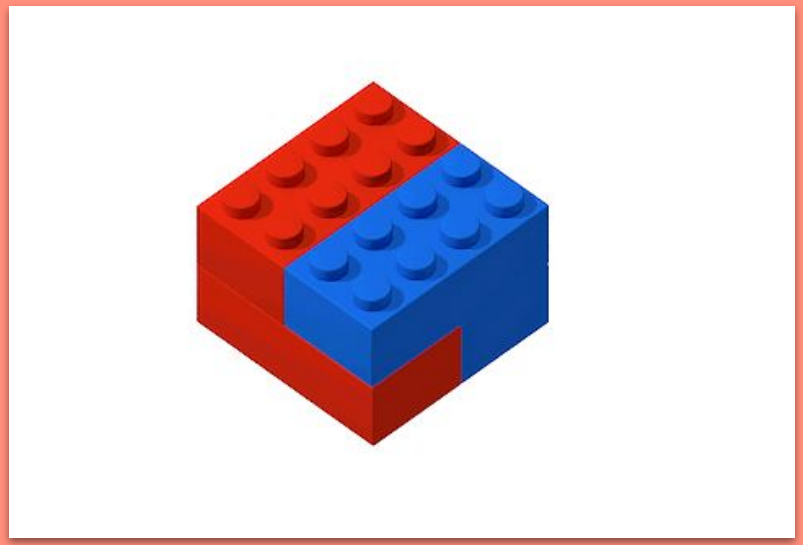
Os widgets são os componentes que compõem o design system.

Um botão é um exemplo de widget.



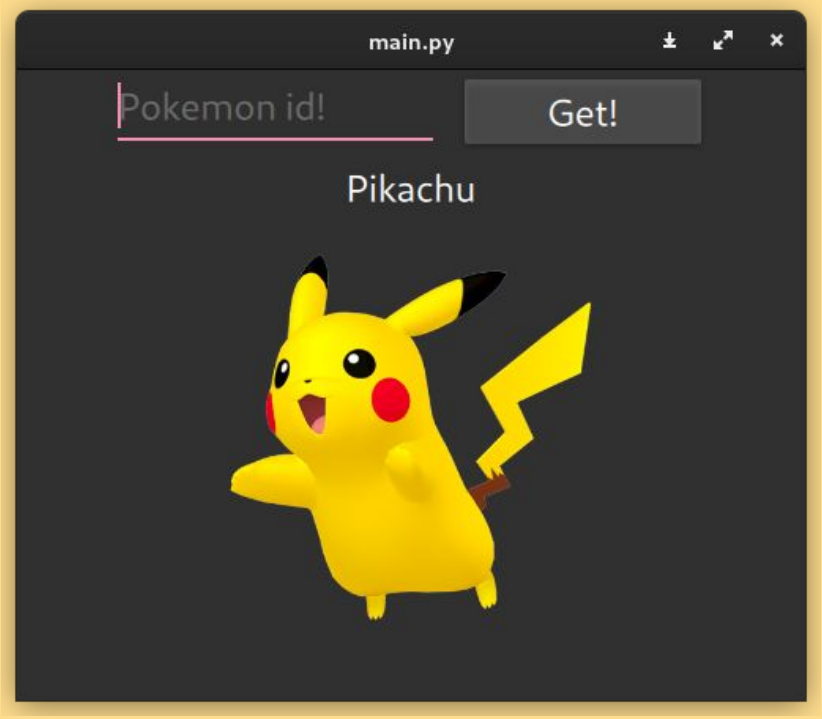
Afinal, o que são widgets?

Um aplicativo QT é, na verdade, um widget que contém diversos widgets que por consequência possuem mais alguns widgets dentro deles.



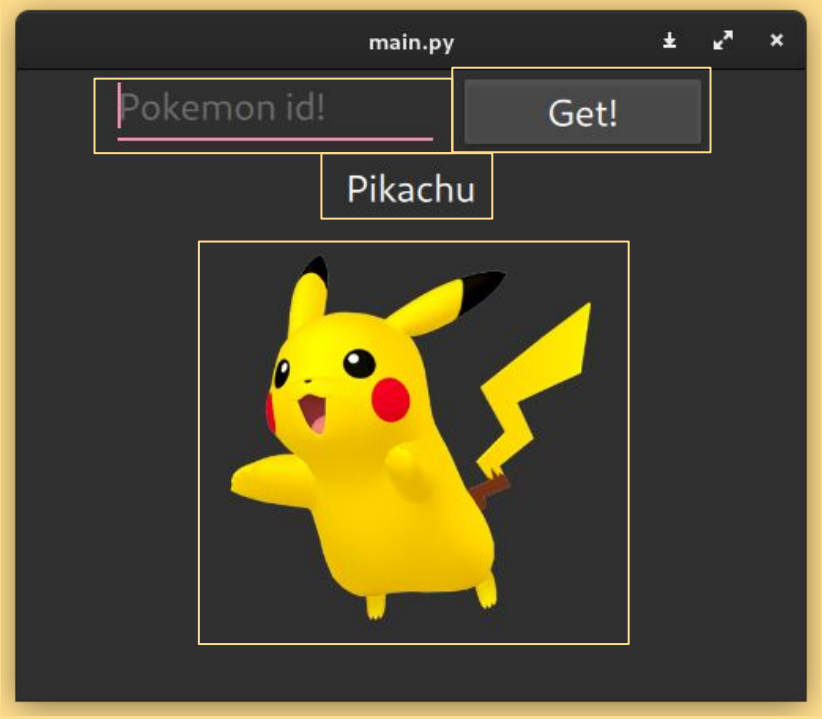
Afinal, o que são widgets?

Um aplicativo QT é, na verdade, um widget que contém diversos widgets que por consequência possuem mais alguns widgets dentro deles.



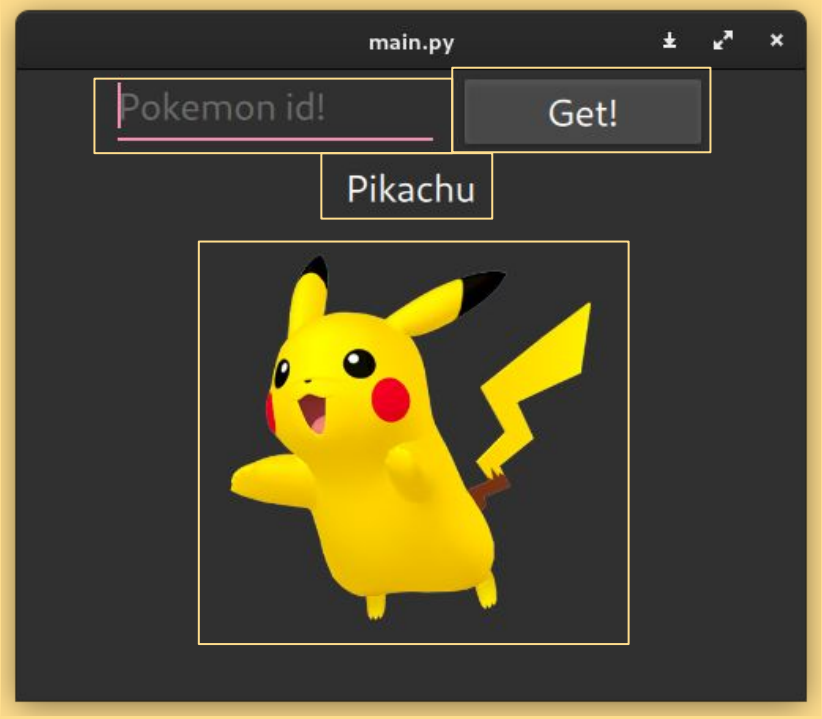
Afinal, o que são widgets?

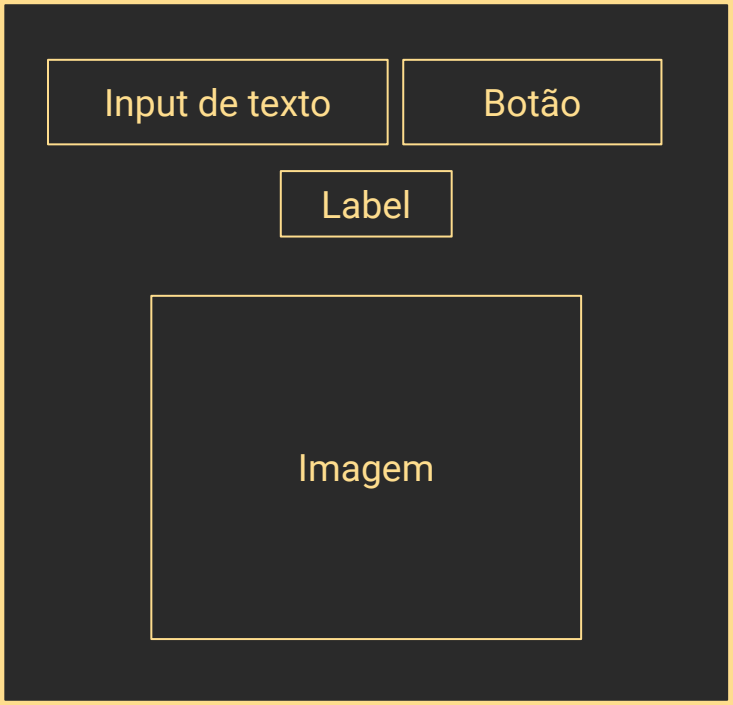
Um aplicativo QT é, na verdade, um widget que contém diversos widgets que por consequência possuem mais alguns widgets dentro deles.



Afinal, o que são widgets?

Um aplicativo QT é, na verdade, um widget que contém diversos widgets que por consequência possuem mais alguns widgets dentro deles.



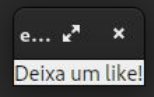


Um app base

Toda estrutura do QT começa com um widget chamado **QApplication**

```python
from PySide6.QtWidgets import QApplication

app = QApplication()
# Algum widget para ser usado
app.exec()
```

Adicionando nosso primeiro widget



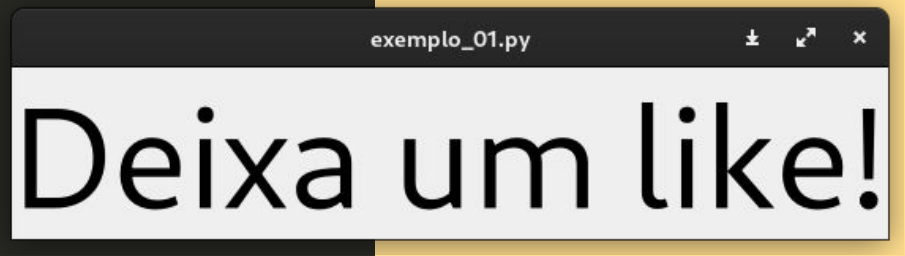
```python
from PySide6.QtWidgets import QApplication, QLabel

app = QApplication()

label = QLabel('Deixa um like!')
label.show()

app.exec()
```

Trazendo as fontes do QtGui

```python
from PySide6.QtGui import QFont
from PySide6.QtWidgets import QApplication, QLabel

app = QApplication()

font = QFont()
font.setPixelSize(90)

label = QLabel('Deixa um like!')
label.setFont(font)
label.show()

app.exec()
```

Adicionando o QtCore

```python
from PySide6.QtCore import Qt
from PySide6.QtGui import QFont
from PySide6.QtWidgets import QApplication, QLabel

app = QApplication()

font = QFont()
font.setPixelSize(90)

label = QLabel('Deixa um like!')
label.setFont(font)
label.setAlignment(Qt.AlignCenter)
label.show()

app.exec()
```



Widget Base

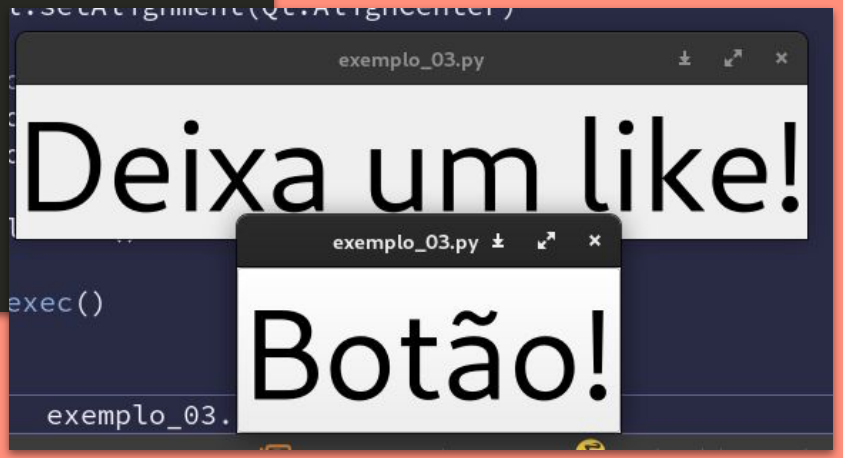
Vamos adicionar mais um botão simples na nossa aplicação para ver o que acontece

```
1   # ...
2   botao = QPushButton('Botão!')
3   botao.setFont(font)
4   botao.show()
5
6   # ...
```

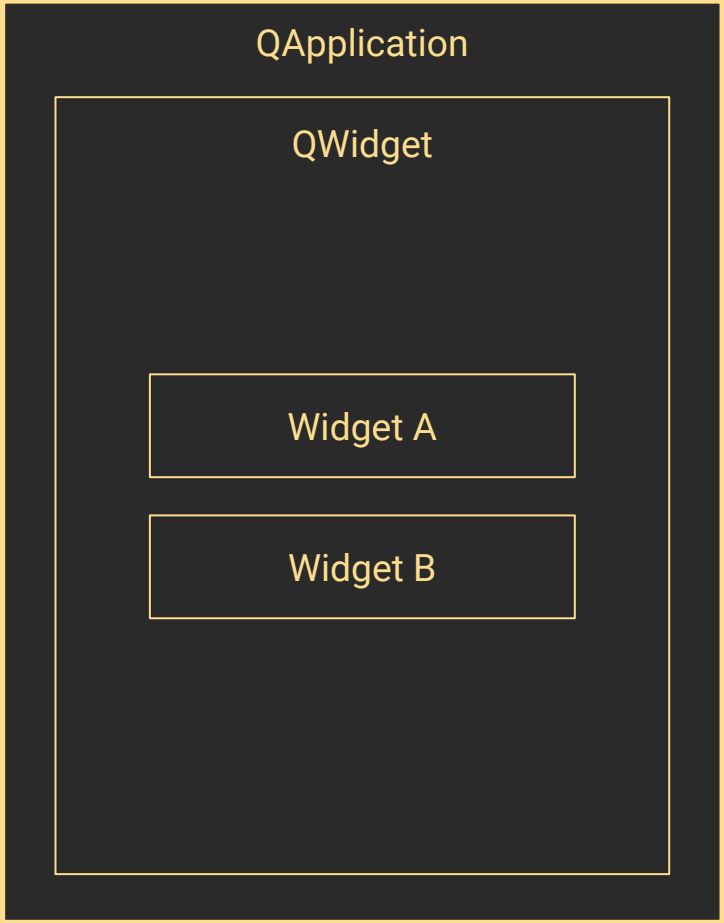
Widget Base

Vamos adicionar mais um botão simples na nossa aplicação para ver o que acontece

```
1   # ...
2   botao = QPushButton('Botão!')
3   botao.setFont(font)
4   botao.show()
5
6   # ...
```



Widget Base



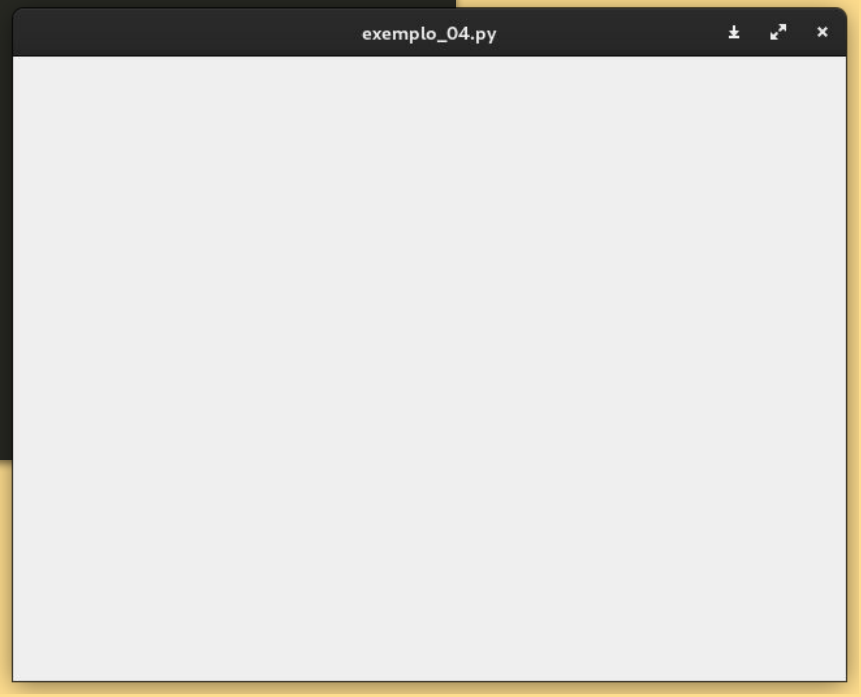
Precisamos de um widget que pode aceitar outros widgets em sua composição.

Esse é o **Qwidget**

```python
from PySide6.QtCore import Qt
from PySide6.QtGui import QFont
from PySide6.QtWidgets import QApplication, QWidget

app = QApplication()
base = QWidget()

base.show()

app.exec()

```

Widget Base

```python
from PySide6.QtCore import Qt
from PySide6.QtGui import QFont
from PySide6.QtWidgets import QApplication, QWidget

app = QApplication()
base = QWidget()

base.show()

app.exec()

```



Layouts, a cola de tudo

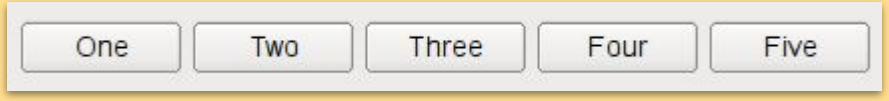
O **Qwidget**, não consegue adicionar outros widgets em sua estrutura. Ele precisa de um widget de **Layout**!

QBoxLayout	Alinha widgets filho horizontalmente ou verticalmente
QButtonGroup	Container para organizar grupos de widgets de botão
QFormLayout	Gerencia formulários de widgets de entrada e seus rótulos associados
QGraphicsÂncora	Representa uma âncora entre dois itens em um QGraphicsAnchorLayout
QGraphicsAnchorLayout	Layout onde se pode ancorar widgets juntos na Visualização Gráfica
QGridLayout	Apresenta widgets em uma grade
QGroupBox	Quadro de caixa de grupo com um título
QHBoxLayout	Alinha os widgets horizontalmente
QLayout	A classe base dos gerenciadores de geometria
QLayoutItem	Item abstrato que um QLayout manipula
QSizePolicy	Atributo de layout que descreve a política de redimensionamento horizontal e vertical
QSpaceItem	Espaço em branco em um layout
QStackedLayout	Pilha de widgets onde apenas um widget é visível por vez
QStackedWidget	Pilha de widgets onde apenas um widget é visível por vez
QVBoxLayout	Alinha os widgets verticalmente
QWidgetItemName	Item de layout que representa um widget

https://doc.qt.io/qt-6/layout.html

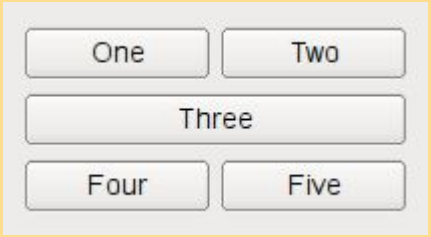
Os formatos principais de layout

QHBoxLayout

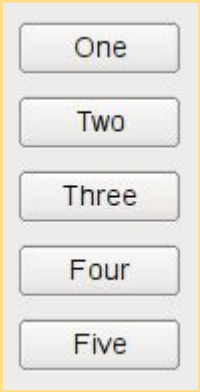


QFormLayout

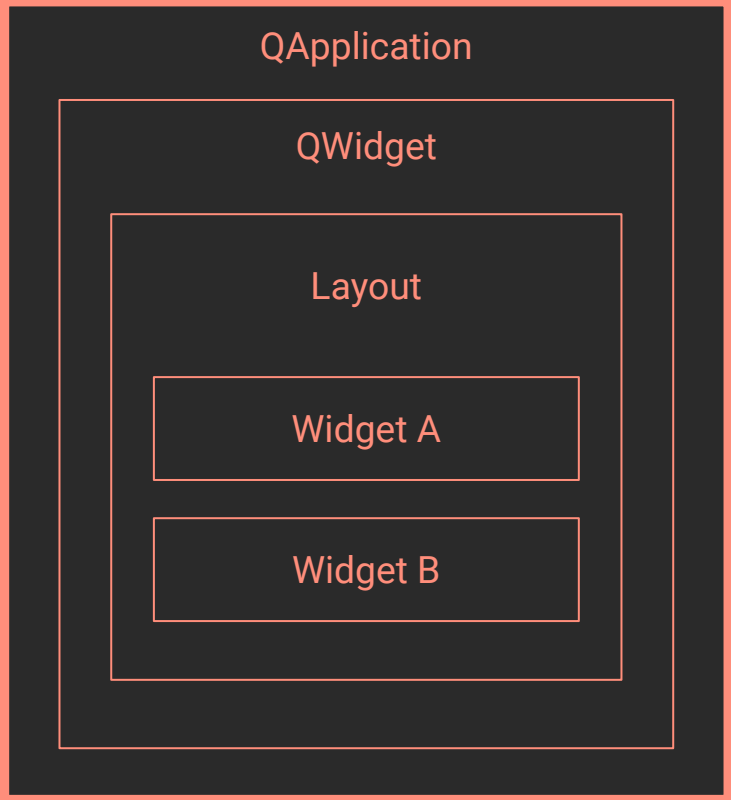
QGridLayout



QVBoxLayout



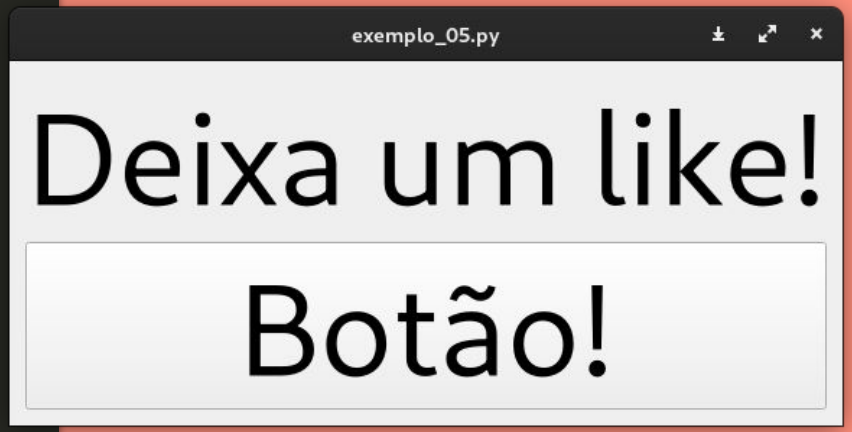
Então, no final. Temos algo assim [exemplo_05.py]




```python
from PySide6.QtCore import Qt
from PySide6.QtGui import QFont
from PySide6.QtWidgets import (
  QApplication, QLabel, QPushButton, QWidget, QVBoxLayout
)

app = QApplication()
base = QWidget()
layout = QBoxLayout()

font = QFont()
font.setPixelSize(90)

label = QLabel('Deixa um like!')
label.setFont(font)
label.setAlignment(Qt.AlignCenter)
layout.addWidget(label)

botao = QPushButton('Botão!')
botao.setFont(font)
layout.addWidget(botao)

base.setLayout(layout)
base.show()

app.exec()
```

Então, no final. Temos algo assim [exemplo_05.py]

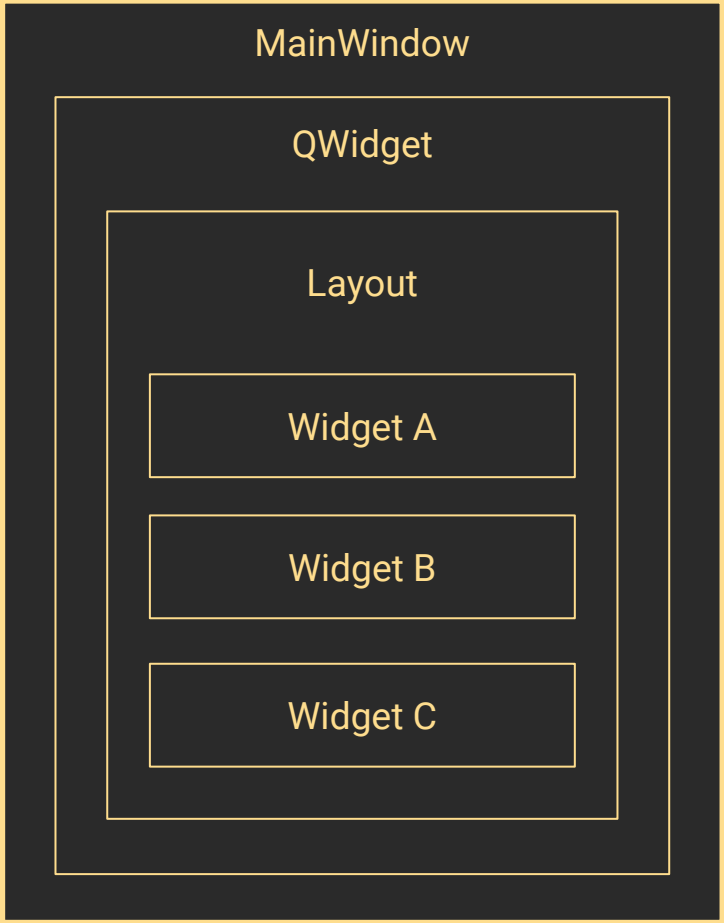
```python
from PySide6.QtCore import Qt
from PySide6.QtGui import QFont
from PySide6.QtWidgets import (
  QApplication, QLabel, QPushButton, QWidget, QVBoxLayout
)

app = QApplication()
base = QWidget()
layout = QBoxLayout()

font = QFont()
font.setPixelSize(90)

label = QLabel('Deixa um like!')
label.setFont(font)
label.setAlignment(Qt.AlignCenter)
layout.addWidget(label)

botao = QPushButton('Botão!')
botao.setFont(font)
layout.addWidget(botao)

base.setLayout(layout)
base.show()

app.exec()
```



Porém, nem tudo são flores

Além do widget principal, se quisermos estender as funcionalidades da Janela. Como Menus e docks. Precisamos de um widget de Janela. Nesse caso vamos usar o **QMainWindow**

Seguindo os padrões [exemplo_06.py]

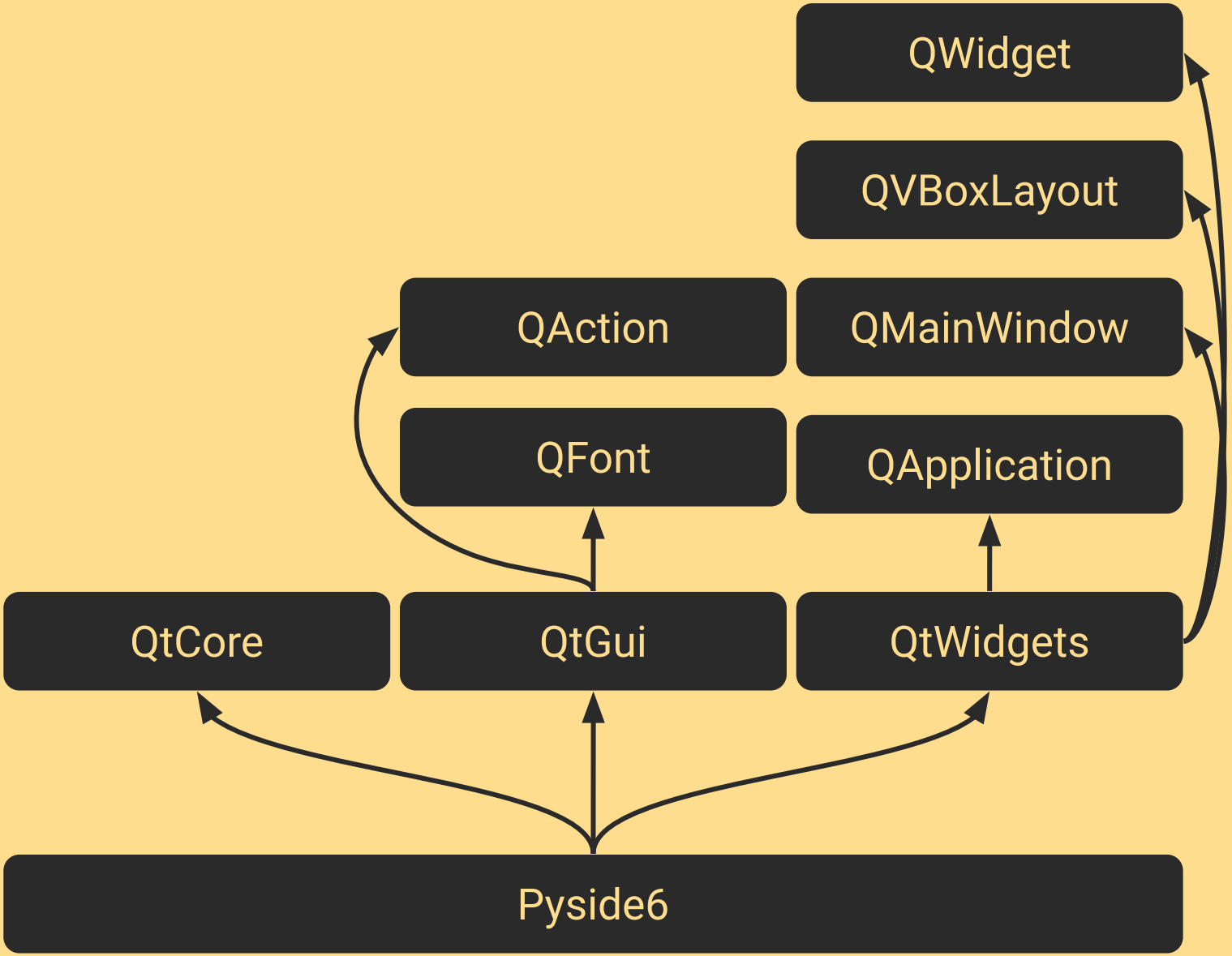



```python
from PySide6.QtCore import Qt
from PySide6.QtGui import QFont
from PySide6.QtWidgets import (
    QApplication, QLabel, QPushButton, QWidget,
    QVBoxLayout, QMainWindow
)

app = QApplication()
window = QMainWindow()
base = QWidget()
layout = QVBoxLayout()

font = QFont()
font.setPixelSize(90)

label = QLabel('Deixa um like!')
label.setFont(font)
label.setAlignment(Qt.AlignCenter)

botao = QPushButton('Botão!')
botao.setFont(font)

layout.addWidget(label)
layout.addWidget(botao)

base.setLayout(layout)
window.setCentralWidget(base)

window.show()

app.exec()
```

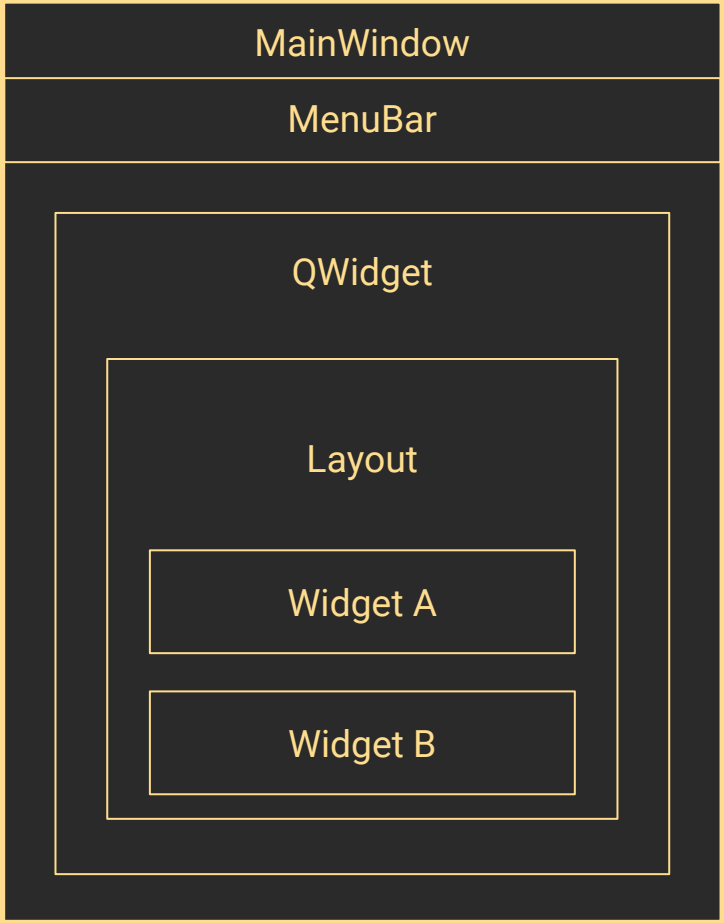
Menus e actions

Para adicionar items no menu da janela precisamos pedir seu menu e adicionar ações a elas. Essas ações são dadas por outro componente. O **QtGui.QAction**

```
1   from PySide6.QtGui import QtAction
2
3   menu = window.menuBar()
4   file_menu = menu.addMenu('Menu')
5   action = QAction('Ação')
6   file_menu.addAction(action)
```

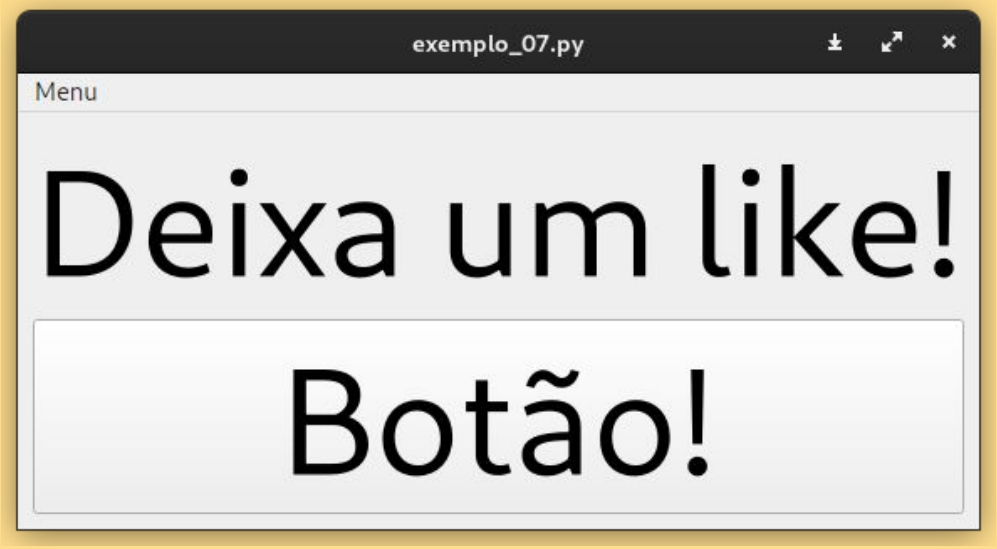

QWidget
QVBoxLayout
QAction
QMainWindow
QFont
QApplication
QtCore
QtGui
QtWidgets
Pyside6

Encapsulando a janela [exmeplo_07.py]




```python
class Window(QMainWindow):
    def __init__(self):
        super().__init__()

        base = QWidget()
        layout = QVBoxLayout()

        font = QFont()
        font.setPixelSize(90)
        base.setFont(font)

        label = QLabel('Deixa um like!')
        label.setAlignment(Qt.AlignCenter)

        botao = QPushButton('Botão!')

        layout.addWidget(label)
        layout.addWidget(botao)

        base.setLayout(layout)
        self.setCentralWidget(base)

        action = QAction('Ação', self)
        menu = self.menuBar()
        menu_geral = menu.addMenu('Menu')
        menu_geral.addAction(action)
```

Encapsulando a janela [exemplo_07.py]




```python
class Window(QMainWindow):
    def __init__(self):
        super().__init__()

        base = QWidget()
        layout = QVBoxLayout()

        font = QFont()
        font.setPixelSize(90)
        base.setFont(font)

        label = QLabel('Deixa um like!')
        label.setAlignment(Qt.AlignCenter)

        botao = QPushButton('Botão!')

        layout.addWidget(label)
        layout.addWidget(botao)

        base.setLayout(layout)
        self.setCentralWidget(base)

        action = QAction('Ação', self)
        menu = self.menuBar()
        menu_geral = menu.addMenu('Menu')
        menu_geral.addAction(action)
```

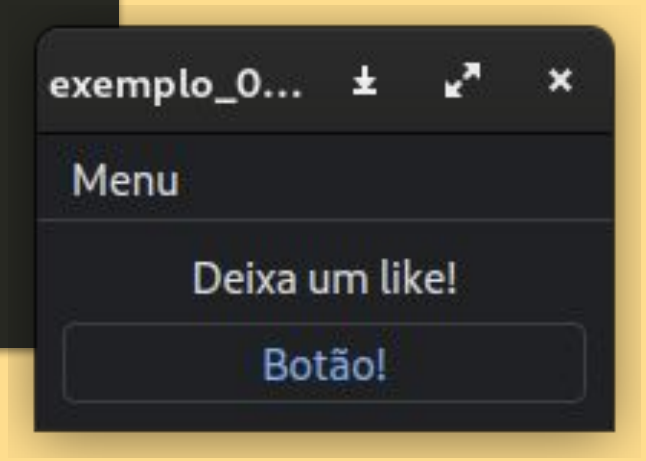
Temas!!!!

Existem diversas bibliotecas de temas para o Pyside, mas vou usar somente uma como exemplo:

```
pip install pyqtdarktheme
```

Tema no código [exemplo_08.py]

```python
from qdarktheme import load_stylesheet

app = QApplication()
app.setStyleSheet(load_stylesheet())

window = Window()
window.show()

app.exec()
```



Callbacks

Tudo que fizemos até agora é relacionado aos widgets. Mas como fazer os botões executarem alguma ação? Precisamos conectar ele a um sinal

```
1   class Window(QMainWindow):
2       def __init__(self):
3           super().__init__()
4           botao.clicked.connect(self.sinal_de_exemplo)
5           action.triggered.connect(self.sinal_de_exemplo)
6
7         def self.sinal_de_exemplo(self):
8             self.label.setText('Clicado!')
```

QT Design

O famoso arrasta e solta

Carregando o widget

```python
from PySide6.QtWidgets import QApplication
from PySide6.QtUiTools import QUiLoader

app = QApplication()

loader = QUiLoader()
window = loader.load('login.ui')
window.show()

app.exec()
```

Não esquecer!

Mostrar signals no qt design

Convertendo ui para .py

```
pyuic5 <arquivo>.ui -o <arquivo>.py
```

QT Quick

Programação descritiva

QML

QT Quick é uma maneira descritiva de criar interfaces.

Você escreve em QML de forma descritiva e economiza tempo escrevendo o que importa

```
1  import QtQuick
2  import QtQuick.Controls
3
4
5  ApplicationWindow {
6    visible: true
7    width:500
8    height: 400
9    font.pixelSize: 24
10
11   Button {
12     text: 'Botão'
13     anchors.horizontalCenter: parent.horizontalCenter
14     anchors.verticalCenter: parent.verticalCenter
15     width: 200
16     height: 100
17   }
18 }
```

Importando no código

```python
from PySide6.QtQml import QQmlApplicationEngine
from PySide6.QtGui import QGuiApplication

app = QGuiApplication()
engine = QQmlApplicationEngine()
engine.load('main.qml')

app.exec()
```

Pokedex!

Ser der tempo, claro!

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3