python™

PySide6

Hadel Rachid Daher Junior

DREAMERS hadelrachid@gmail.com

Sumário

CAPÍTULO 1

1.1. INTRODUÇÃO

Se você quer criar aplicativos GUI com Python, pode ser complicado saber por onde começar. Há muitos conceitos novos que você precisa entender para fazer qualquer coisa funcionar. Mas, como qualquer problema de codificação, o primeiro passo é aprender a abordar o problema da maneira certa. Neste livro, eu levo dos princípios básicos do desenvolvimento de GUI para criar seus próprios aplicativos de desktop totalmente funcionais com **PySide6**.

A primeira edição deste livro foi lançada em 2016. Desde então, ele foi atualizado 14 vezes, adicionando e expandindo capítulos em resposta ao feedback do leitor. Há mais recursos do **PySide** disponíveis agora do que quando comecei, mas ainda há uma escassez de guias práticos e aprofundados para construir aplicativos completos. Este livro preenche essa lacuna!

O livro é formatado como uma série de capítulos explorando diferentes aspectos do PySide6 por vez. Eles são organizados para colocar os capítulos mais simples no início, mas se você tiver requisitos específicos para seu projeto, não tenha medo de pular. Cada capítulo o guiará pelo aprendizado dos conceitos fundamentais antes de levá-lo por uma série de exemplos de codificação para explorar gradualmente e aprender como aplicar as ideias você mesmo. Você pode baixar o código-fonte e os recursos para todos os exemplos neste livro em

http://www.pythonguis.com/d/pyside6-source.zip

Não é possível dar a você uma visão geral completa de todo o ecossistema Qt em um livro deste tamanho, então há links para recursos externos — tanto no site pythonguis.com quanto em outros lugares. Se você se pegar pensando "Será que consigo fazer isso?", a melhor coisa que você pode fazer é largar este livro e ir lá e descobrir! Apenas mantenha backups regulares do seu código ao longo do caminho para que você sempre tenha algo para consultar se você fizer uma grande bagunça.

Ao longo deste livro, há caixas como esta, dando informações, dicas e avisos. Todas elas podem ser puladas com segurança se você estiver com pressa, mas lê-las lhe dará um conhecimento mais profundo e completo do framework Qt.

1.2. Uma breve história da GUI

A **Interface Gráfica do Usuário** tem uma longa e respeitável história que remonta à década de 1960. O **NLS** (Sistema oN-Line) **de Stanford** introduziu os conceitos de **mouse** e **janelas**, demonstrados publicamente pela primeira vez em 1968. Isso foi seguido pelo sistema **Xerox PARC Smalltalk GUI** 1973, que é a base da maioria das GUIs modernas de uso geral.

Esses primeiros sistemas já tinham muitos dos recursos que consideramos garantidos nas GUIs de desktop modernas, incluindo janelas, menus, botões de opção, caixas de seleção e ícones. Essa combinação de recursos nos deu a sigla inicial usada para esses tipos de interfaces: **WIMP** (**Windows (janelas), Icons (ícones), Menus (menus), Point Device (dispositivo apontador — um mouse)**).

Em 1979, o primeiro sistema comercial com uma GUI foi lançado — a estação de trabalho PERQ. Isso estimulou uma série de outros esforços de GUI, incluindo, notavelmente, o **Apple Lisa** (1983), que adicionou o conceito de **barra de menu** e **controles de janela**, bem como outros sistemas da **Atari** (**GEM**) e **Amiga**. No UNIX, o X Window System surgiu em 1984, enquanto a primeira versão do Windows para PC foi lançada em 1985.

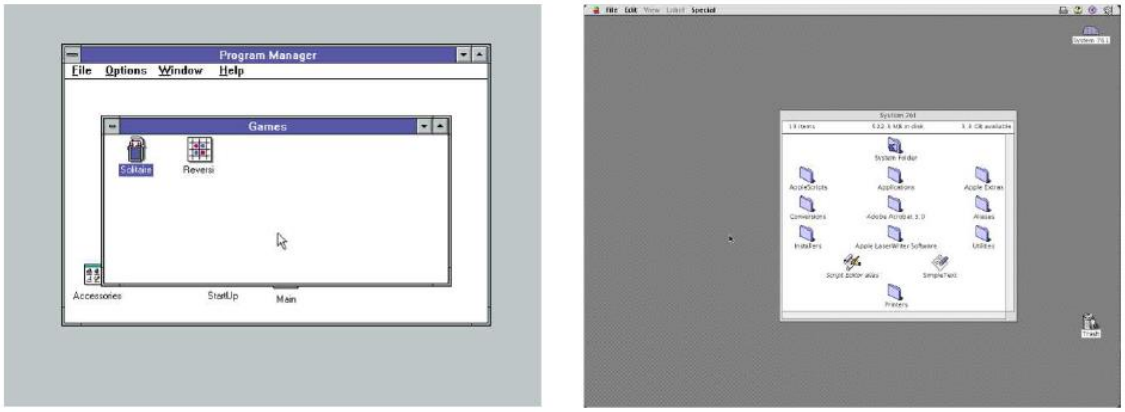

Figura 1. A área de trabalho no Microsoft Windows 3.1 (1992) e no Apple System 7 (1991)

As primeiras GUIs não foram o sucesso instantâneo que você pode imaginar, devido à falta de software compatível no lançamento e aos requisitos de hardware caros — particularmente para usuários domésticos. No entanto, lentamente, mas firmemente, o paradigma da GUI se tornou a maneira preferida de interagir com computadores e a metáfora WIMP se tornou firmemente estabelecida como o padrão. Isso não quer dizer que não tenha havido tentativas de substituir a metáfora WIMP no desktop. Microsoft Bob (1995), por exemplo, foi a tentativa muito difamada da Microsoft de substituir o desktop por uma casa.

Figura 2. Microsoft Bob — Descartando a metáfora do desktop para uma casa de desenho animado

Não faltaram interfaces de usuário aclamadas como revolucionárias em seu tempo, desde o lançamento do **Windows 95** (1995) até o **Mac OS X** (2001), **GNOME Shell** (2011) e **Windows 10** (2015). Cada uma delas reformulou a IU de seus respectivos sistemas, geralmente com muito alarde, mas fundamentalmente nada mudou realmente. Essas interfaces de usuário ainda são muito sistemas WIMP e funcionam da mesma forma que as GUIs desde a década de 1980.

Quando a revolução veio, ela era móvel — o mouse foi substituído pelo toque e as janelas por aplicativos de tela cheia. Mas mesmo em um mundo onde todos nós andamos por aí com smartphones no bolso, uma enorme quantidade de trabalho diário ainda é feita em computadores de mesa. O WIMP sobreviveu a 40 anos de inovação e parece sobreviver a muitos mais.

1.3. Um pouco sobre o Qt

O **Qt** é um kit de ferramentas de **widgets** gratuito e de **código aberto** para criar aplicativos de GUI multiplataforma, permitindo que os aplicativos tenham como alvo várias plataformas do **Windows**, **macOS**, **Linux** e **Android** com uma única base de código. Mas o **Qt** é muito mais do que um kit de ferramentas de widgets e recursos integrados de suporte para multimídia, bancos de dados, gráficos vetoriais e interfaces MVC, é mais preciso pensar nele como uma estrutura de desenvolvimento de aplicativos. O Qt foi iniciado por **Eirik Chambe-Eng e Haavard Nord** em 1991, fundando a primeira empresa **Qt**, a **Trolltech**, em 1994. O Qt é atualmente desenvolvido pela **The Qt Company** e continua sendo atualizado regularmente, adicionando recursos e estendendo o suporte móvel e multiplataforma.

O nome "Qt" foi escolhido por seus criadores, **Eirik Chambe-Eng** e **Haavard Nord**, como uma referência ao nome da fonte utilizada no primeiro emulador de terminais que eles desenvolveram, que se chamava "Quasar". A escolha da letra "Q" vem de Quasar, e a letra "t" foi adicionada simplesmente porque a letra "Q" seguida de "t" formava um bom nome curto e fácil de lembrar.

Qt é conhecido por sua robustez, eficiência, e a capacidade de criar aplicações que funcionam em diferentes sistemas operacionais, como Windows, macOS, Linux, e outras plataformas.

1.4. Qt e PySide6

PySide6, também conhecido como Qt para Python, é uma ligação Python do kit de ferramentas Qt, atualmente desenvolvido pela **The Qt Company**. Quando você escreve aplicativos usando **PySide6**, o que você está realmente fazendo é escrever aplicativos em Qt. A biblioteca PySide6 é, na verdade, um wrapper em torno da biblioteca C++ Qt, o que torna possível usá-la em Python.

Como esta é uma interface Python para uma biblioteca C++, as convenções de nomenclatura usadas no **PySide6** não aderem aos padrões **PEP8**. Por exemplo, funções e variáveis são nomeadas usando **mixedCase** em vez de **snake case**. Se você adere a este padrão em seus próprios aplicativos depende inteiramente de você, no entanto, acho útil continuar a seguir os padrões Python para meu próprio código, para ajudar a esclarecer onde o código PySide6 termina e o seu começa.

Por fim, embora haja documentação específica do PySide6 disponível, você frequentemente se verá lendo a documentação do Qt em si, pois ela é mais completa. Se você precisar de conselhos sobre como converter código Qt C++ para Python, dê uma olhada em Traduzindo C++ Exemplos para Python.

1.5. Recursos básicos do PySide6

É hora de dar os primeiros passos na criação de aplicativos GUI com o PySide6! Neste capítulo, você será apresentado aos conceitos básicos do PySide6, que são as bases de qualquer aplicativo que você criar. Desenvolveremos um aplicativo simples em janela na sua área de trabalho. Adicionaremos widgets, os organizaremos usando layouts e conectaremos esses widgets a funções, permitindo que você acione o comportamento do aplicativo a partir da sua GUI. Use o código fornecido como seu guia, mas sinta-se sempre à vontade para experimentar. Essa é a melhor maneira de aprender como as coisas funcionam.

Antes de começar, você precisa de uma instalação funcional do PySide6. Se você ainda não tem uma, confira: Instalando o PySide6.

Não se esqueça de baixar o código-fonte que acompanha este livro em http://www.pythonguis.com/d/pyside6-source.zip

1.5.1 O PyQt6 e o PySide6 são parecidos?

Sim, o PyQt6 e o PySide6 são muito parecidos, pois ambos são **bindings** (vínculos) em Python para o **framework Qt**, que é usado para o desenvolvimento de interfaces gráficas (GUIs) e outras aplicações interativas. No entanto, existem algumas diferenças importantes entre eles:

Similaridades

- **Base no Qt**: Tanto PyQt6 quanto PySide6 são baseados no mesmo framework Qt, o que significa que a maior parte da funcionalidade que você pode acessar através de um está disponível no outro.
- **APIs Semelhantes**: As APIs do PyQt6 e do PySide6 são muito semelhantes, com muitos nomes de classes, métodos, e funcionalidades sendo quase idênticos. Isso facilita a migração de um para o outro, se necessário.
- **Cross-Platform**: Ambos permitem a criação de aplicações que podem rodar em diferentes sistemas operacionais (Windows, macOS, Linux) sem a necessidade de modificar o código.

Diferenças

Licenciamento:

- **PyQt6**: É licenciado sob a GPL (*General Public License*) e também oferece uma licença comercial. Se você deseja usar o PyQt6 em um projeto proprietário (fechado), precisa de uma licença comercial.
- **PySide6**: É licenciado sob a LGPL (*Lesser General Public License*), o que é mais permissivo para projetos comerciais, pois permite o uso em softwares proprietários sem a necessidade de adquirir uma licença comercial.

Suporte Oficial:

- **PyQt6**: Desenvolvido pela **Riverbank Computing**, que também oferece suporte e atualizações para os usuários que adquirirem a licença comercial.
- **PySide6**: É desenvolvido pela própria **Qt Company**, a mesma empresa responsável pelo Qt. Isso significa que PySide6 recebe suporte direto da empresa que desenvolve o Qt.

Compatibilidade de Código:

- **PyQt6**: Às vezes, pode haver pequenas diferenças na API que exigem modificações no código ao migrar de PyQt para PySide ou vice-versa.
- **PySide6**: Em geral, a compatibilidade com a API Qt tende a ser mais direta, visto que é mantido pela mesma organização que desenvolve o Qt.

Ferramentas de Desenvolvimento:

- **PyQt6**: Inclui algumas ferramentas específicas, como `pyuic6` para converter arquivos `.ui` (criadas com Qt Designer) em código Python.
- **PySide6**: Inclui ferramentas semelhantes, mas algumas diferenças nas opções e na forma de utilização podem existir.

Conclusão

Ambos são opções excelentes para o desenvolvimento de GUIs em Python. A escolha entre **PyQt6** e **PySide6** muitas vezes se resume a preferências pessoais, requisitos de licenciamento, e considerações sobre suporte e comunidade.

Capítulo 2

2.1. Meu primeiro aplicativo

Vamos criar nosso primeiro aplicativo! Para começar, crie um novo arquivo Python — você pode chamá-lo do que quiser (por exemplo, myapp.py) e salvá-lo em algum lugar acessível. Escreveremos nosso aplicativo simples neste arquivo.

Estaremos editando este arquivo à medida que avançamos, e você pode querer voltar para versões anteriores do seu código, então lembre-se de manter backups regulares.

2.1.1. Criando seu aplicativo

O código-fonte do seu primeiro aplicativo é mostrado abaixo. Digite-o na íntegra e tome cuidado para não cometer erros. Se você errar, o Python informará o que está errado. Se você não quiser digitar tudo, o arquivo está incluído no código-fonte deste livro.

```
from PySide6.QtWidgets import QApplication, QWidget

# Necessário apenas para acesso a argumentos de linha de comando
import sys

# Você precisa de uma (e somente uma) instância QApplication por
# aplicativo.
# Passe sys.argv para permitir argumentos de linha de comando para
# seu aplicativo.
# Se você sabe que não usará argumentos de linha de comando,
# QApplication([]) também funciona.

app = QApplication(sys.argv) # Instancia um objeto Qpplication com ar-
gumentos.

# Crie um widget Qt, que será nossa janela.
window = QWidget()
window.show() # IMPORTANTE!!!!! As janelas ficam ocultas por padrão.

# Inicia o loop de eventos.
app.exec()
# Seu aplicativo não chegará aqui até que você saia e
# o loop de eventos tenha parado.
```

Salve o arquivo como: **window.py**

Primeiro, inicie seu aplicativo. Você pode executá-lo a partir da linha de comando como qualquer outro script Python, por exemplo:

```
> python window.py
```

Ou usando o Python 3

```
> python3 window.py
```

Execute-o! Agora você verá sua janela. O Qt cria automaticamente uma janela com as decorações normais de janela e você pode arrastá-la e redimensioná-la como qualquer janela.

O que você verá dependerá da plataforma em que você está executando este exemplo. A imagem abaixo mostra a janela como exibida no Windows, macOS e Linux (Ubuntu).

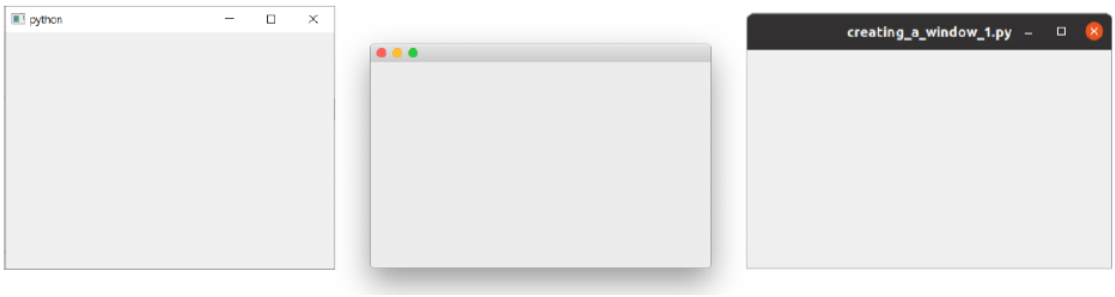

Figura 3. Nossa janela, vista no Windows, macOS e Linux.

2.1.2. Percorrendo o código

Vamos percorrer o código linha por linha, para entendermos exatamente o que está acontecendo.

Primeiro, importamos as classes **PySide6** que precisamos para o aplicativo. Aqui, estamos importando **QApplication**, o manipulador do aplicativo, e **QWidget**, um widget GUI básico vazio, ambos do módulo **QtWidgets**.

```
from PySide6.QtWidgets import QApplication, QWidget
```

Os principais módulos do **Qt** são **QtWidgets**, **QtGui** e **QtCore**.

Você pode usar *from <module> import **,* mas esse tipo de importação global geralmente é desaprovado em Python, então vamos evitá-lo aqui.

Em seguida, criamos uma instância do **QApplication**, passando **sys.arg**, que é uma lista Python contendo os argumentos de linha de comando passados para o aplicativo.

```
app = QApplication(sys.argv)
```

Se você sabe que não usará argumentos de linha de comando para controlar o Qt, você pode passar uma lista vazia, por exemplo:

```
app = QApplication([])
```

Em seguida, criamos uma instância de um **QWidget** usando o nome da variável **window**.

```
# Crie um widget Qt, que será nossa janela.
window = QWidget()
window.show() # IMPORTANTE!!!!! As janelas ficam ocultas por padrão.
```

No **Qt**, todos os **widgets** de **nível superior** são **janelas** — ou seja, eles não têm um pai e não estão aninhados dentro de outro widget ou layout. Isso significa que você pode tecnicamente criar uma janela usando qualquer widget que desejar.

Não consigo ver minha janela!

Widgets sem um pai são invisíveis por padrão. Então, depois de criar o objeto ***window***, devemos sempre chamar ***.show()*** para torná-lo visível. Você pode remover o .show() e executar o aplicativo, mas não terá como fechá-lo!

O que é uma janela?

- Contém a interface de usuário do seu aplicativo.
- Cada aplicativo precisa de pelo menos uma (...mas pode ter mais)
- O aplicativo irá (por padrão) fechar quando a última janela for fechada.

Por fim, chamamos **app.exec()** para iniciar o loop de eventos.

2.2. O que é o loop de eventos?

Antes de colocar a janela na tela, há alguns conceitos-chave para introduzir sobre como os aplicativos são organizados no mundo **Qt**. Se você já estiver familiarizado com loops de eventos, pode pular com segurança para a próxima seção.

O núcleo de cada aplicativo **Qt** é a classe **QApplication**. Cada aplicativo precisa de um — e apenas um — objeto **QApplication** para funcionar. Este objeto contém o loop de eventos (**event loop)** do seu aplicativo — o loop principal que governa toda a interação do usuário com a **GUI**.

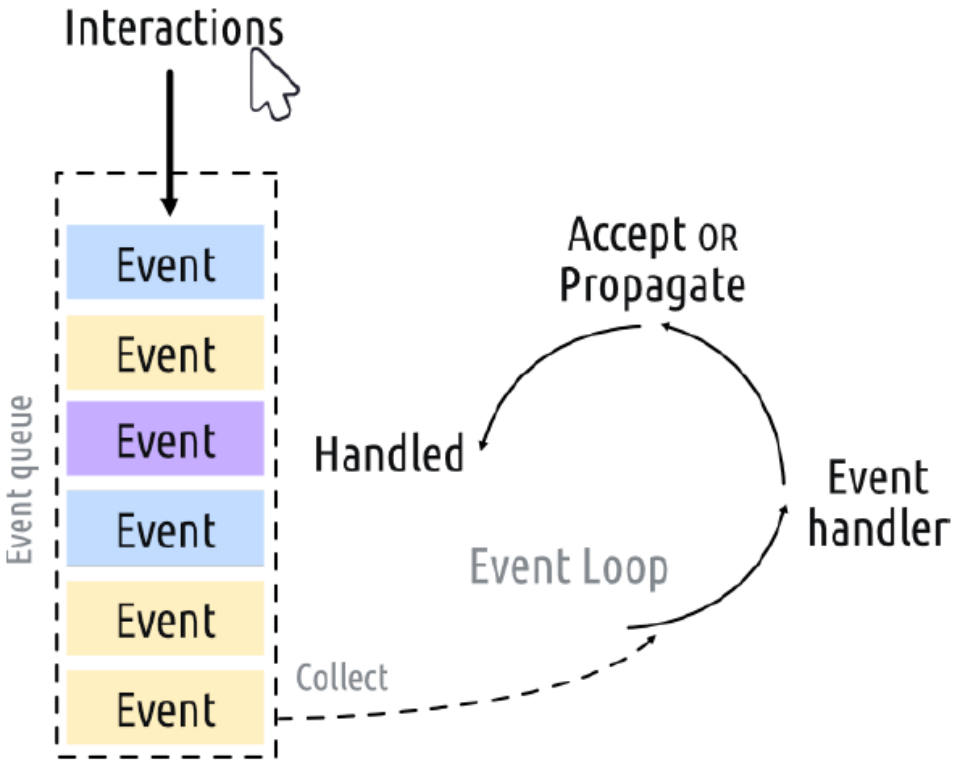

Figure 4. The event loop in Qt.

Figura 4. O loop de eventos no Qt.

Cada interação com seu aplicativo — seja um pressionamento de tecla, clique de mouse ou movimento do mouse — gera um evento que é colocado na **fila de eventos** (**event queue**). No **loop de eventos** (**event loop**), a fila é verificada em cada iteração e, se um evento em espera for encontrado, o evento e o controle são passados para o **manipulador de eventos** (***event handler)*** específico para o evento. O **manipulador de eventos** lida com o evento e, em seguida, passa o controle de volta para o loop de eventos para esperar por mais eventos. Há apenas um loop de eventos em execução por aplicativo.

A classe QApplication

- **QApplication** mantém o loop de eventos Qt
- Uma instância **QApplication** necessária
- Seu aplicativo fica esperando no loop de eventos até que uma ação seja tomada
- Há apenas um loop de eventos a qualquer momento

QMainWindow

Como descobrimos na última parte, no Qt qualquer widget pode ser uma janela. Por exemplo, se você substituir **QtWidget** por **QPushButton**. No exemplo abaixo, você obteria uma janela com um único botão "pressionável" nela.

```
import sys

from PySide6.QtWidgets import QApplication, QPushButton

app = QApplication(sys.argv)

window = QPushButton("Push Me")
window.show()

app.exec()
```

Veja a imagem a seguir:

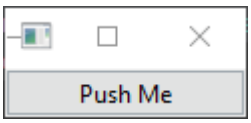

Figura 5. QPushButton como uma QtWidget.

Isso é legal, mas não muito útil — é raro que você precise de uma UI que consiste em apenas um único controle! Mas, como descobriremos mais tarde, a capacidade de aninhar widgets dentro de outros widgets usando layouts significa que você pode construir UIs complexas dentro de um **QWidget** vazio.

Mas, o Qt já tem uma solução para você — o **QMainWindow**. Este é um widget pré-fabricado que fornece muitos recursos de janela padrão que você usará em seus aplicativos, incluindo barras de ferramentas, menus, uma barra de status, widgets encaixáveis e muito mais.

Veremos esses recursos avançados mais tarde, mas, por enquanto, adicionaremos um **QMainWindow** simples e vazio ao nosso aplicativo.

```
from PySide6.QtWidgets import QApplication, QMainWindow

import sys

# Instancia um QApplication
app = QApplication(sys.argv)

# Instancia um QMainWindow: Janela Padrão
window = QMainWindow()
window.show() # IMPORTANTE!!!!! As janelas ficam ocultas por padrão.

# Inicia o Loop de Eventos
app.exec()
```

Execute-o! Agora você verá sua janela principal. Ela parece exatamente a mesma de antes!.

Figura 6. QMainWindow

Então nossa **QMainWindow** não é muito interessante no momento. Podemos consertar isso adicionando algum conteúdo. Se você quiser criar uma janela personalizada, a melhor abordagem é subclassificar **QMainWindow** e então incluir a configuração para a janela no bloco **__init__**. Isso permite que o comportamento da janela seja autocontido. Podemos adicionar nossa própria subclasse de **QMainWindow** — chame-a de MainWindow para manter as coisas simples.

```python
import sys

from PySide6.QtCore import QSize, Qt
from PySide6.QtWidgets import (
    QApplication, # O QApplication mantém o loop de eventos do Qt
    QMainWindow,  # Uma Janela Padrão
    QPushButton   # Um botão
)

# Subclasse QMainWindow para personalizar
# a janela principal do seu aplicativo.
class MainWindow(QMainWindow): # MainWindow herda de QMainWindow
    def __init__(self):
        super().__init__() # Chama o construtor Pai

        # Configura a janela principal!

        # Coloca um título na barra de Título
        self.setWindowTitle("Primeira Janela")

        # Adiciona um botão
        button = QPushButton("Press Me!")

        # Define o widget central da janela.
        self.setCentralWidget(button)


def main():
    app = QApplication(sys.argv) # Incia o Applicativo

    window = MainWindow()       # Instancia a classe MainWindow
    window.show()               # Mostra a Janela

    app.exec()                  # Inicia o Loop de Eventos


if __name__ == '__main__':
    main()
```

1

2

3

1. Os widgets comuns do Qt são sempre importados do *namespace* **QtWidgets**.
2. Devemos sempre chamar o método **__init__** da classe ***super()***.
3. Use ***.setCentralWidget*** para colocar um widget no **QMainWindow**.

Ao criar uma subclasse de uma classe Qt, você deve sempre chamar a função super ***__init__*** para permitir que o Qt configure o objeto.

Em nosso bloco ***__init__***, primeiro usamos **.setWindowTitle()** para alterar o título da nossa janela principal. Então, adicionamos nosso primeiro widget — um **QPushButton** — ao meio da janela. Este é um dos widgets básicos disponíveis no Qt. Ao criar o botão, você pode passar o texto que deseja que o botão exiba.

Finalmente, chamamos **.setCentralWidget()** na janela. Esta é uma função específica do **QMainWindow** que permite que você defina o widget que vai no meio da janela.

Execute-o! Agora você verá sua janela novamente, mas dessa vez com o widget **QPushButton** no meio. Pressionar o botão não fará nada, vamos resolver isso em seguida.

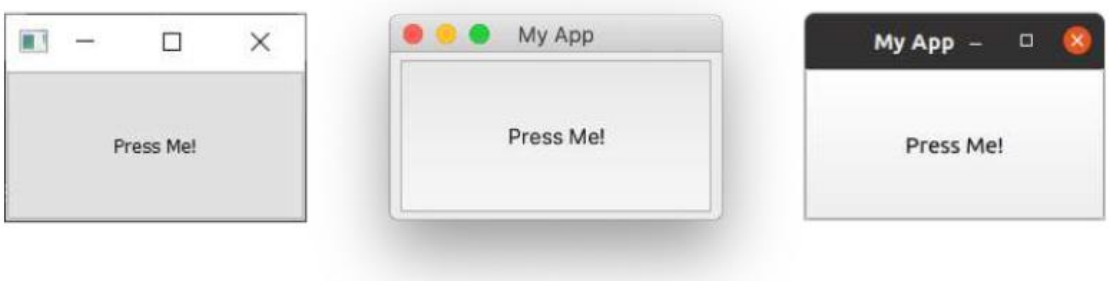

Figura 7. Nossa QMainWindow com um único QPushButton no Windows, macOS e Linux.

Com fome de widgets?

Abordaremos mais widgets em detalhes em breve, mas se você estiver impaciente e quiser pular para a frente, pode dar uma olhada na documentação do **QWidget**. Tente adicionar os diferentes widgets à sua janela!

Parou pág 16....

Referência:

FITZPATRICK, Martin. Create GUI Applications with Python & Qt6, 5th Edition | PySide6: The hands-on guide to making apps with Python. 5.0 ed., 2022.