

Interfaces gráficas desktop com PyQt6

Disciplina: Programação aplicada à engenharia cartográfica

Maurício C. M. de Paulo - D.Sc.

11 de fevereiro de 2026

PyQt6 é um conjunto de bindings Python para o framework gráfico **Qt 6**.

Qt é usado para:

- Aplicações desktop multiplataforma
- Interfaces gráficas profissionais
- Softwares científicos e comerciais

Principais características:

- Multiplataforma (Windows, Linux, macOS)
- Orientado a objetos
- Baseado em sinais e slots

Estrutura de uma aplicação PyQt6

Toda aplicação PyQt6 segue o mesmo padrão:

```
from PyQt6.QtWidgets import QApplication, QWidget
import sys

app = QApplication(sys.argv)

janela = QWidget()
janela.setWindowTitle("Minha aplicação")
janela.show()

sys.exit(app.exec())
```

Componentes principais:

- QApplication: gerencia a aplicação
- QWidget: elemento visual básico
- exec(): loop de eventos

Widgets e Layouts

Widgets são os elementos visuais:

- Botões (QPushButton)
- Campos de texto (QLineEdit)
- Rótulos (QLabel)

Layouts organizam os widgets:

- QVBoxLayout — vertical
- QHBoxLayout — horizontal
- QGridLayout — grade

Boa prática:

- Nunca posicionar widgets manualmente
- Sempre usar layouts

Sinais e Slots

PyQt6 usa o modelo de **sinais e slots** para eventos.

```
from PyQt6.QtWidgets import QPushButton

botao = QPushButton("Clique aqui")

def ao_clicar():
    print("Botão clicado!")

botao.clicked.connect(ao_clicar)
```

Ideia central:

- Sinal: algo aconteceu
- Slot: função que responde ao evento

Recomendações:

- Criar classes para janelas
- Separar lógica da interface
- Evitar código no escopo global

Estrutura comum de projeto:

- `main.py` — inicialização
- `ui/` — interfaces
- `logic/` — regras de negócio

Qt Designer é uma ferramenta visual para criar interfaces gráficas.

Passos:

- 1 Abrir o Qt Designer
- 2 Criar um projeto do tipo **Main Window**
- 3 Adicionar widgets (ex.: QLabel, QPushButton)
- 4 Salvar como `main_window.ui`

Resultado:

- Arquivo XML `.ui`
- Descreve apenas a interface
- Nenhuma lógica de aplicação

Componentes principais:

- QApplication: gerencia o ciclo da aplicação
- QMainWindow: janela principal
- Arquivo .ui: interface gráfica

Estrutura de arquivos:

```
projeto/  
  main.py  
  main_window.ui
```

Aplicação PyQt6 — Código mínimo

```
import sys
from PyQt6.QtWidgets import QApplication, QMainWindow
from PyQt6.uic import loadUi

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("main_window.ui", self)
        self.setWindowTitle("Aplicação PyQt6 mínima")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

- O .ui é carregado dinamicamente
- A lógica fica separada da interface
- A aplicação entra no loop de eventos

PyQt6 — Acessando widgets do .ui

Quando um arquivo .ui é carregado com loadUi, os widgets tornam-se atributos da classe.

```
from PyQt6.QtWidgets import QMainWindow
from PyQt6.uic import loadUi

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("main_window.ui", self)

# Acessando widgets do .ui
self.pushButton.setText("Clique aqui")
self.label.setText("Olá, PyQt6!")
```

Importante:

- O nome do atributo vem do **objectName** no Qt Designer
- Ex.: pushButton, label, lineEdit

Widgets emitem **sinais** quando algo acontece.

```
class MainWindow(QMainWindow):  
    def __init__(self):  
        super().__init__()  
        loadUi("main_window.ui", self)  
  
        self.pushButton.clicked.connect(self.ao_clicar)  
  
    def ao_clicar(self):  
        self.label.setText("Botão clicado!")
```

Fluxo:

- Usuário clica no botão
- Sinal clicked é emitido
- Método ao_clicar() é executado

Sinais e slots conectam eventos à lógica da aplicação.

```
self.pushButton.clicked.connect(self.processar)

def processar(self):
    texto = self.lineEdit.text()
    self.label.setText(texto)
```

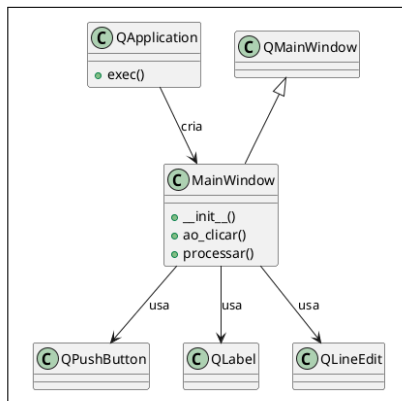
Resumo conceitual:

- **Sinal:** evento (ex.: clique, texto alterado)
- **Slot:** função Python
- O .ui define a interface
- O Python define o comportamento

Vantagem:

- Interface desacoplada da lógica

PyQt6 — Diagrama UML da aplicação



Objetivo do diagrama:

- Visualizar responsabilidades
- Relacionar Qt Designer e código Python

Desenhem uma interface gráfica que tenha:

- Um label que começa com "Programação pyqt"
- Uma caixa de texto
- Um botão, que ao ser clicado copia o texto da caixa para o label.

Empacotando PyQt6 com Pixi

Objetivo: gerar executável (.exe) mantendo ambiente reprodutível.

Criar projeto Pixi (se ainda não existir)

```
pixi init
```

Adicionar dependências

```
pixi add python pyqt pyinstaller
```

Pixi garante:

- Ambiente isolado
- Controle de versões
- Reprodutibilidade do build

Executando PyInstaller via Pixi

Rodar dentro do ambiente Pixi:

```
pixi run pyinstaller --onefile --windowed main.py
```

Arquivos gerados:

- dist/main.exe
- build/
- main.spec

Vantagem:

- Não depende do Python global do sistema
- Build consistente entre máquinas

Automatizando com pixi.toml

Podemos criar uma task no `pixi.toml`:

```
[tasks]
build = "pyinstaller --onefile --windowed main.py"
```

Executar com:

```
pixi run build
```

Boas práticas:

- Versionar o `pixi.lock`
- Testar o executável fora do ambiente
- Ajustar o `.spec` se houver recursos externos