Linguagem de Programação Back-End

Prof. Wellington S. S. Silva

Python: Bibliotecas Gráficas (GUI)

Introdução: Como funciona uma Biblioteca Gráfica em Linguagem Python?

Biblioteca Gráficas – O que são e como funcionam

As **Bibliotecas Gráficas (GUI) do Python** são coleções de módulos de código que permitem que você crie **Interfaces Gráficas de Usuário** (*Graphical User Interfaces*).

Em termos simples, elas são as ferramentas que transformam um programa de console (que interage apenas por texto no terminal) em um **aplicativo de desktop** com elementos visuais interativos.

Algumas das Bibliotecas (GUI) utilizadas

1. Tkinter
2. PyQt
3. PySide
4. Streamlit
5. Dash

Propósito das Bibliotecas (GUI)

O propósito fundamental de uma biblioteca GUI é fornecer os blocos de construção e a infraestrutura necessários para:

Criar Janelas: Estabelecer a principal área visual do seu aplicativo.

Widgets: Fornecer componentes de interface prontos para uso, como:

- **Botões** (Buttons)
- **Caixas de Texto** (Entry ou Text Area)
- **Rótulos** (Labels)
- **Menus** (Menus)
- **Caixas de Seleção** (Checkboxes) e **Botões de Rádio** (Radio Buttons)

Propósito das Bibliotecas (GUI)

Gerenciamento de Eventos: Lidar com as ações do usuário. A biblioteca monitora eventos (como um clique do mouse, uma tecla pressionada ou o redimensionamento da janela) e executa o código Python apropriado (as suas funções).

Layout: Ajudar a organizar e posicionar esses *widgets* dentro da janela de forma coerente e responsiva.

Aparência: Fazer com que o aplicativo se pareça com um programa nativo do sistema operacional (Windows, macOS, Linux) ou tenha uma aparência personalizada e moderna.

Biblioteca – Tkinter (TK Interface)

A biblioteca Tkinter é a interface padrão do Python para criar Interfaces Gráficas de Usuário (GUIs - Graphical User Interfaces). Em termos simples, ela permite que você crie janelas, botões, caixas de texto e outros elementos visuais para interagir com seus programas Python.

- Biblioteca Padrão do Python: O Tkinter é um pacote que já vem pré-instalado com a maioria das distribuições Python. Isso é uma grande vantagem, pois você não precisa instalar nada extra – basta usar o import tkinter e começar a programar.

Biblioteca – Tkinter (TK Interface)

Interface Tcl/Tk: O nome "Tkinter" significa "Tk interface". Ele funciona como uma "ponte" ou um invólucro (wrapper) entre o código Python e o kit de ferramentas gráfico subjacente chamado Tcl/Tk. Você não precisa aprender a linguagem Tcl/Tk, pois o Python cuida de toda a comunicação.

- Criação de Aplicações Desktop: O principal uso do Tkinter é para criar aplicativos de desktop (ou seja, programas que rodam diretamente no seu computador, como um bloco de notas ou uma calculadora), que funcionam nos principais sistemas operacionais (Windows, macOS e Linux).

A seguir temos um exemplo de código utilizando essa biblioteca.

```
import tkinter as tk
from tkinter import ttk

# --- VARIÁVEL GLOBAL PARA O TEXTO ---
# Deve ser definida antes da função para que a função possa acessá-la e alterá-la
# Usamos tk.StringVar para que o Tkinter monitore as mudanças nesta variável
texto_dinamico = None   # Será inicializada abaixo

def alterar_texto():
    """Altera o texto do rótulo manipulando a tk.StringVar."""

    # Obtém o valor atual da variável usando .get()
    texto_atual = texto_dinamico.get()

    if texto_atual == "Clique no botão abaixo!":
        # Altera o valor da variável usando .set()
        texto_dinamico.set("O texto mudou! Bem-vindo ao Tkinter!")
    else:
        # Altera o valor de volta
        texto_dinamico.set("Clique no botão abaixo!")

# --- 1. CONFIGURAÇÃO DA JANELA PRINCIPAL ---
janela = tk.Tk()
janela.title("Exemplo Tkinter Top Sucesso!!!")
janela.geometry("400x150")

# Inicializa a variável tk.StringVar, que será manipulada pela função
texto_dinamico = tk.StringVar(value="Clique no botão abaixo!")

# --- 2. CRIAÇÃO DOS WIDGETS ---
frame_principal = ttk.Frame(janela, padding="10")
frame_principal.grid(row=0, column=0, sticky="nsew")

# Cria o Rótulo (Label) e o VINCULA à variável texto_dinamico
# Usamos textvariable=texto_dinamico
label_mensagem = ttk.Label(
    frame_principal,
    textvariable=texto_dinamico, # Agora o Label lê o valor desta variável
    font=("Arial", 14)
)
label_mensagem.grid(row=0, column=0, pady=20, sticky="n")

# Cria o Botão (Button)
botao_acao = ttk.Button(frame_principal, text="Alterar Mensagem", command=alterar_texto)
botao_acao.grid(row=1, column=0, pady=10)


# --- 3. CONFIGURAÇÃO DE LAYOUT E LOOP ---
janela.columnconfigure(0, weight=1)
janela.rowconfigure(0, weight=1)
frame_principal.columnconfigure(0, weight=1)

janela.mainloop()
```

Biblioteca – PyQt

A biblioteca **PyQt** é uma das bibliotecas GUI mais populares e poderosas para Python. Ela não é apenas uma biblioteca; é um *binding* (uma "ponte") para um *framework* de desenvolvimento muito maior e mais robusto, chamado **Qt**.

O Qt é um *framework* originalmente escrito em C++ e amplamente utilizado em todo o mundo para desenvolver aplicativos de nível profissional e multiplataforma. O PyQt permite que desenvolvedores Python acessem todas as funcionalidades ricas do Qt, escrevendo código **apenas em Python**.

Biblioteca – PyQt

O que torna o PyQt especial?

1. **Baseado no Framework Qt (Poder e Profissionalismo)**

Rico em Recursos: Diferente de soluções mais simples como o Tkinter, o PyQt oferece milhares de classes e módulos que vão além dos *widgets* básicos. Ele tem ferramentas para trabalhar com:

- Gráficos e visualização de dados 2D e 3D.
- Integração com bancos de dados (SQL).
- Multimídia (áudio e vídeo).
- Rede e conectividade.
- Internacionalização e suporte a múltiplos idiomas.

Aplicações Profissionais: PyQt é a escolha para construir grandes e complexos aplicativos de desktop. Aplicações conhecidas como a IDE **Spyder** (muito usada em Ciência de Dados) e o *software* de modelagem 3D **Autodesk Maya** (que usa o Qt) dependem dessa tecnologia.

Biblioteca – PyQt

pip install PyQt6

2. Multiplataforma Completa

O Qt é inerentemente multiplataforma. Isso significa que um aplicativo desenvolvido com PyQt pode ser compilado e executado em **Windows, macOS, Linux** e até mesmo em algumas plataformas móveis, mantendo uma aparência **nativa** (ou quase nativa) em cada sistema.

3. O Sistema de Sinais e Slots

O PyQt usa um mecanismo de comunicação chamado **Sinais e Slots**, que é o coração de como ele lida com a interatividade.

Sinais: São eventos que um *widget* emite. Por exemplo, quando um botão é clicado, ele emite um **sinal** chamado clicked().

Slots: São as funções (ou métodos) que são executadas em resposta a um sinal.

4. Qt Designer (Desenho Visual)

O PyQt vem com uma ferramenta chamada **Qt Designer**, que permite criar a interface gráfica de forma visual (arrastando e soltando *widgets*), sem precisar escrever código Python para cada elemento. Você projeta a interface no Designer e depois carrega ou converte esse arquivo (.ui) para o seu código Python, acelerando drasticamente o desenvolvimento.

```
import sys
from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QPushButton
from PyQt6.QtGui import QFont
from PyQt6.QtCore import Qt # Necessário para o alinhamento

# 1. Definir a classe da Janela Principal (Herda de QWidget)
class MinhaJanela(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Exemplo PyQt com Botão Fechar')
        self.setGeometry(100, 100, 450, 250) # Aumentamos um pouco o tamanho

        self.iniciar_componentes()

    def iniciar_componentes(self):
        # --- RÓTULO (LABEL) ---
        label = QLabel("Olá, PyQt!", self)
        label.setFont(QFont('Arial', 24))
        # Centralizando o rótulo
        label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        # Posiciona o rótulo
        label.setGeometry(50, 50, 350, 50)


        # --- BOTÃO FECHAR (QPushButton) ---
        botao_fechar = QPushButton('Fechar', self)
        botao_fechar.setFont(QFont('Arial', 12))

        # Posiciona o botão (x, y, largura, altura)
        botao_fechar.setGeometry(150, 150, 150, 40)

        # CONEXÃO: Conecta o sinal 'clicked' do botão ao método 'close' da janela.
        # O método self.close() é um método padrão de QWidget que fecha a janela.
        botao_fechar.clicked.connect(self.close)

# --- 2. EXECUTAR A APLICAÇÃO ---
if __name__ == '__main__':
    # Garante que o aplicativo seja executado corretamente em diferentes ambientes
    app = QApplication(sys.argv)

    # Cria e exibe a janela
    janela_principal = MinhaJanela()
    janela_principal.show()

    # Inicia o loop de eventos
    sys.exit(app.exec())
```

Biblioteca – PySide

pip install PySide6

A biblioteca **PySide** é, em essência, a **alternativa oficial e licenciada de forma mais permissiva ao PyQt**. Assim como o PyQt, o PySide é um *binding* que permite que a linguagem Python acesse e utilize o poderoso *framework* de desenvolvimento de aplicações **Qt** (originalmente escrito em C++).

O que é o PySide?

Binding Oficial do Qt: O PySide é o *binding* Python mantido e desenvolvido pela **The Qt Company** (a empresa por trás do *framework* Qt). Desde que foi adotado oficialmente, é a escolha recomendada pela própria Qt para o desenvolvimento Python.

Acesso Total ao Qt: Ele fornece o mesmo nível de funcionalidade avançada que o PyQt. Isso inclui a capacidade de criar interfaces gráficas complexas, usar o sistema de **Sinais e Slots**, trabalhar com gráficos, multimídia, redes, e usar a ferramenta visual **Qt Designer**.

Versões:

PySide2: Utiliza o Qt na versão 5.
PySide6: Utiliza o Qt na versão 6 (a mais recente).

A Diferença Chave: Licenciamento

A principal razão histórica e prática para a existência do PySide é a sua licença, que resolve um problema de uso comercial que existia com o PyQt.

```
import sys
# A única diferença significativa é que importamos do PySide6
from PySide6.QtWidgets import QApplication, QWidget, QLabel, QPushButton
from PySide6.QtGui import QFont
from PySide6.QtCore import Qt # Necessário para o alinhamento

# 1. Definir a classe da Janela Principal (Herda de QWidget)
class MinhaJanela(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Exemplo PySide Simples')
        self.setGeometry(100, 100, 450, 250)

        self.iniciar_componentes()

    def iniciar_componentes(self):
        # --- RÓTULO (QLabel) ---
        label = QLabel("Olá, PySide!", self)
        label.setFont(QFont('Arial', 24))

        # Centralizando o rótulo
        label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        label.setGeometry(50, 50, 350, 50)


        # --- BOTÃO FECHAR (QPushButton) ---
        botao_fechar = QPushButton('Fechar', self)
        botao_fechar.setFont(QFont('Arial', 12))

        # Posiciona o botão (x, y, largura, altura)
        botao_fechar.setGeometry(150, 150, 150, 40)

        # CONEXÃO: Conecta o sinal 'clicked' do botão ao método 'close' da
janela.
        # A sintaxe de Sinais e Slots é idêntica à do PyQt.
        botao_fechar.clicked.connect(self.close)


# --- 2. EXECUTAR A APLICAÇÃO ---
if __name__ == '__main__':
    # Cria a instância da QApplication (obrigatória)
    app = QApplication(sys.argv)

    # Cria e exibe a janela
    janela_principal = MinhaJanela()
    janela_principal.show()

    # Inicia o loop de eventos
    sys.exit(app.exec())
```

Biblioteca – Streamlit

A biblioteca **Streamlit** é uma ferramenta revolucionária e extremamente popular no ecossistema Python, especialmente entre **cientistas de dados, analistas e engenheiros de Machine Learning**.

Sua principal missão é: **Permitir que desenvolvedores Python criem e compartilhem aplicativos web interativos (dashboards, ferramentas de análise ou interfaces para modelos de ML) usando apenas código Python simples, sem a necessidade de HTML, CSS ou JavaScript.**

O que torna o Streamlit único?

O Streamlit é diferente das bibliotecas GUI de desktop (Tkinter, PyQt) e de *frameworks* web tradicionais (Flask, Django) por sua **simplicidade orientada a dados**:

1. **Transformação de Script**

Você não precisa de uma estrutura de classes ou de um *framework* MVC (Modelo-Visão-Controlador). Você simplesmente escreve um script Python **linear** de cima para baixo.

O Streamlit interpreta este script e o transforma em uma **página web interativa** automaticamente.

Biblioteca – Streamlit

2. Interatividade Padrão

Adicionar um *widget* interativo (como um controle deslizante, botão ou caixa de seleção) é feito com uma única linha de código, como st.slider().

Quando um usuário interage com um *widget*, o Streamlit **reexecuta automaticamente** todo o seu script, com o novo valor do *widget*. Isso faz com que a reatividade e a atualização de dados sejam incrivelmente fáceis de implementar.

3. Foco em Dados

Possui funções integradas e otimizadas para exibir resultados de análise de dados. Você pode mostrar:

Tabelas do Pandas (st.dataframe()).

Gráficos do Matplotlib, Seaborn ou Plotly (st.pyplot(), st.plotly_chart()).

Imagens, vídeos e mapas.

Ele inclui um sistema de **cache** inteligente (@st.cache_data) que memoriza os resultados de funções de computação pesada, como carregamento de dados ou treinamento de modelos de Machine Learning, garantindo que o aplicativo seja rápido.

Biblioteca – Streamlit

pip install streamlit

Após digitar o Código, **abra o terminal** e execute conforme o exemplo abaixo.

streamlit run app_streamlit.py

Principais Casos de Uso

O Streamlit é a escolha ideal para:

Prototipagem Rápida: Transformar uma análise de dados ou um modelo de ML que está em um Jupyter Notebook em um aplicativo compartilhável em minutos.

Dashboards Internos: Criar ferramentas de visualização de dados e *dashboards* para equipes internas.

Interfaces de Modelo de ML: Construir interfaces simples para que usuários não técnicos possam inserir dados e ver a previsão de um modelo de Machine Learning em tempo real.

Em resumo, o **Streamlit** é o "botão fácil" do Python para criar aplicações web interativas voltadas para dados, permitindo que o foco permaneça na lógica de dados, e não na complexidade do desenvolvimento web.

Biblioteca – Streamlit

Após digitar o Código, **abra o terminal** e execute conforme o exemplo abaixo.

streamlit run app_streamlit.py

O que vai acontecer ao executar o Programa da próxima página.

O Streamlit abrirá automaticamente uma nova guia no seu navegador (geralmente em http://localhost:8501/) exibindo o seu aplicativo.

pip install streamlit

A Mágica do Streamlit

Aparência Web: O script Python foi transformado em uma página web com estilos modernos.

Reatividade:

Slider: Mova o controle deslizante (st.slider). Você verá o texto **Sua idade** selecionada é: **X anos** atualizar instantaneamente sem que você precise programar nenhum *callback* ou loop. O Streamlit reexecuta todo o script em segundo plano com o novo valor.

Botão: O código de saudação só é executado quando o botão é clicado, ativado a mensagem de sucesso (st.success).

Funções de Exibição: st.title, st.write, st.text_input, st.button, st.success, st.warning e ***st.info*** são comandos simples do **Streamlit** que cuidam de todo o *frontend* para você.

```
import streamlit as st

# --- 1. Exibição de Título e Texto ---
st.title('Meu Primeiro App Streamlit')
st.write('Olá! Digite seu nome e clique em "Saudar" para ver a mágica.')

# --- 2. Criação de Widgets Interativos ---

# st.text_input cria uma caixa de texto e armazena o valor digitado na variável 'nome'
nome = st.text_input('Qual é o seu nome?', 'Visitante')

# st.button cria um botão. Ele retorna True se foi clicado, False caso contrário.
if st.button('Saudar'):
    # O código dentro deste 'if' será executado APENAS quando o botão for clicado

    if nome and nome != 'Visitante':
        st.success(f'Olá, {nome}! Seja muito bem-vindo(a) ao Streamlit.')
    else:
        st.warning('Por favor, digite um nome válido.')

# --- 3. Demonstração de um Slider ---
# Adicionando outro widget para mostrar a reatividade automática
st.subheader('\nDemonstração de Reatividade:')
idade = st.slider('Selecione sua idade:', 0, 100, 25)

# Este texto é atualizado automaticamente sempre que você move o slider!
st.info(f'Sua idade selecionada é: {idade} anos.')
```

Biblioteca – Dash

A biblioteca **Dash**, desenvolvida pela **Plotly**, é um *framework* de código aberto para a criação de **aplicações analíticas web** e *dashboards* interativos.

Assim como o Streamlit, o principal objetivo do Dash é permitir que cientistas de dados, engenheiros e analistas criem ***softwares*** sofisticados baseados em dados **usando apenas Python**, evitando a necessidade de escrever código HTML, CSS ou JavaScript.

O que compõe o Dash?

O Dash é mais do que uma biblioteca; é um *framework* construído sobre três pilares essenciais:

1 - Flask: Usado como o *backend* (servidor web) que lida com as requisições e executa o aplicativo.

2 - Plotly.js: Usado no *frontend* para renderizar gráficos de altíssima qualidade e interativos.

3 - React.js: Usado no *frontend* para gerenciar a interface de usuário e a reatividade.

O papel do Dash: O Dash atua como a **"ponte"** que permite que você use Python para controlar todos esses componentes de *backend* (Flask) e *frontend* (Plotly/React).

Biblioteca – Dash

Principais Características e Diferenciais

1. Abordagem Baseada em Componentes e HTML

Diferente do Streamlit (que usa um fluxo de script linear), o Dash adota uma abordagem mais estruturada, próxima do desenvolvimento web tradicional:

Layout: Você define o layout do seu *dashboard* usando classes Python que representam elementos HTML (ex: html.Div, html.H1) e componentes interativos (ex: dcc.Graph, dcc.Dropdown). Isso oferece **controle total** sobre a estrutura e o estilo (via CSS/HTML), algo que o Streamlit simplifica demais.

2. Sistema de Callbacks (Funções de Reatividade)

A interatividade é o ponto forte do Dash e é gerenciada pelo sistema de **Callbacks** (funções de retorno):

Você define funções Python que são rotuladas com o *decorator* @app.callback.

Essas funções especificam explicitamente qual *widget* fornece a **entrada** (Input) e qual *widget* será a **saída** (Output).

A Vantagem: O Dash só reexecuta a função de *callback* específica quando a entrada muda. Isso o torna altamente eficiente e **escalável** para aplicações grandes com muitos componentes interativos, pois nem todo o script precisa ser reexecutado (como faz o Streamlit).

Biblioteca – Dash

3. Visualização com Plotly

O Dash se integra perfeitamente com a biblioteca **Plotly**, permitindo a criação de gráficos interativos (que permitem zoom, *hover*, etc.) por padrão, sem código adicional.

Dash vs. Streamlit

Característica	Dash	Streamlit
Filosofia	Framework Componentizado (Modelo MVC)	Fluxo de Script Linear ("Python Notebook")
Escalabilidade	**Melhor.** Ideal para aplicações grandes e de nível empresarial.	Melhor para prototipagem rápida e aplicações menores/internas.
Reatividade	**Explícita** via *Callbacks* (Input, Output).	**Automática e Implícita** (reexecuta o script inteiro).
Customização	**Alta.** Permite o uso de HTML/CSS para layout e estilo detalhado.	Baixa. Estilos predefinidos; difícil customização profunda.
Curva de Aprendizado	Mais íngreme (requer entender *callbacks* e a estrutura de componentes)	Muito baixa (parece um script Python normal).

Biblioteca – Dash

Em resumo, o **Dash** é a escolha se você precisa de **controle máximo**, **escalabilidade** para grandes aplicações e visualizações altamente **personalizadas**, aceitando uma curva de aprendizado um pouco mais acentuada.

pip install Dash Pandas Plotly

Após digitar o Código, **abra o terminal** e execute conforme o exemplo abaixo.

streamlit app_dash.py

O Dash abrirá uma página no seu navegador (geralmente em http://127.0.0.1:8050/).

```
from dash import Dash, dcc, html, Input, Output
import pandas as pd
import plotly.express as px

# 1. Inicializa o aplicativo Dash
# 'name' é o nome do módulo principal
app = Dash(__name__)

# --- 2. DEFINIÇÃO DO LAYOUT ---
# O layout define a estrutura visual da página usando componentes
app.layout = html.Div([

    # Título (Usamos um componente HTML)
    html.H1(children='Dash - Reatividade Simples', style={'textAlign': 'center'}),

    # Componente de Entrada (Dash Core Components)
    html.Div(children='Digite um número:', style={'marginTop': 20, 'marginBottom': 10}),

    # dcc.Input é o widget de entrada de texto
    dcc.Input(
        id='entrada-numero',
        type='number', # Define que o tipo de entrada é número
        value=5,       # Valor inicial
        style={'fontSize': 16}
    ),

    # Rótulo de Saída (Onde o resultado será exibido)
    html.Div(id='saida-resultado', style={'marginTop': 20, 'fontSize': 20, 'color': 'darkblue'})
])
```

```
# --- 3. DEFINIÇÃO DO CALLBACK (A LÓGICA DE INTERAÇÃO) ---

@app.callback(
    # OUTPUT: Onde o resultado será injetado
    Output(component_id='saida-resultado', component_property='children'),
    # INPUT: De qual componente e propriedade o valor será lido
    Input(component_id='entrada-numero', component_property='value')
)
def atualizar_saida(valor_de_entrada):
    """
    Esta função é um callback. Ela é executada automaticamente sempre
    que o valor (value) do componente 'entrada-numero' muda.
    """
    try:
        # Se o valor for válido (não nulo), calcula o quadrado
        numero = float(valor_de_entrada)
        resultado = numero ** 2
        return f'O quadrado de {numero} é: {resultado:.2f}'
    except:
        # Retorna uma mensagem de erro se a entrada não for um número válido
        return 'Por favor, insira um número válido.'


# --- 4. INICIA O SERVIDOR ---
if __name__ == '__main__':
    # Roda o aplicativo no modo debug para ver as alterações em tempo real
    app.run(debug=True)
```

Obrigado

Wellington S. S. Silva

guitomw@outlook.com