
Python and Tkinter
A Developer's Guide
Tkithon
Label
Hello, Tkinter!
Button
Iprem Ipuem Text Me ...
Click Me
Entry
Yourr Byrem (Strop)
Textr
X
iotpetent {
import tkinter as tk
}

Guia Prático de Tkinter: Crie Aplicações com Python

Capítulo 1: Os Fundamentos Essenciais

Neste primeiro capítulo, vamos começar do zero. Aprenderemos a criar uma janela básica e adicionar os primeiros elementos visuais, conhecidos como **widgets**.

1. Criando sua Primeira Janela

Para iniciar qualquer aplicação com **Tkinter**, o primeiro passo é criar a janela principal. Essa janela será o "recipiente" de todos os seus outros elementos.

O código abaixo mostra como criar uma janela simples. As três linhas de código são a base de qualquer projeto com Tkinter.

```python
import tkinter as tk

# 1. Cria a janela principal
janela = tk.Tk()

# 2. Inicia o loop principal da aplicação.
# Ele mantém a janela aberta e responsiva a eventos.
janela.mainloop()
```

Para rodar o código, salve-o como um arquivo `.py` e execute-o. Você verá uma janela vazia aparecer na sua tela.

2. Adicionando Texto com o `Label`

O **`Label`** é um widget usado para exibir texto. Ele é ideal para títulos, descrições ou qualquer informação estática que você queira mostrar.

Para usá-lo, você precisa:

1. Criar uma instância do **`Label`**, passando a janela como primeiro argumento.
2. Definir o texto que ele irá exibir.
3. Empacotar (`pack()`) o widget para que ele seja exibido na janela.

Adicione o código abaixo ao nosso exemplo anterior:

```python
import tkinter as tk

janela = tk.Tk()

# Cria um widget de texto (Label) e o adiciona à janela
titulo = tk.Label(text="Olá, Tkinter!")

# O método pack() 'empacota' o widget e o exibe na janela
titulo.pack()

janela.mainloop()
```

Ao executar este novo código, você verá a janela com o texto "Olá, Tkinter!" no centro.

3. Adicionando um Botão com o `Button`

O **`Button`** é um dos widgets mais importantes, pois permite a interação do usuário. Um botão pode ser usado para fechar a aplicação, salvar um arquivo, ou qualquer outra ação.

Para adicionar um botão, o processo é o mesmo que o do **`Label`**.

Veja como o nosso código fica com a adição de um botão:

```python
import tkinter as tk

janela = tk.Tk()

# Adiciona o Label
titulo = tk.Label(text="Olá, Tkinter!")
titulo.pack()

# Adiciona um botão
botao = tk.Button(text="Clique Aqui")
botao.pack()

janela.mainloop()
```

Agora sua janela terá tanto o texto quanto o botão. Por enquanto, o botão não faz nada, pois não definimos uma ação para ele. Veremos como fazer isso em capítulos futuros.

Capítulo 2: Gerenciando o Layout da Interface

Depois de criar seus primeiros widgets, o próximo passo crucial é saber como organizá-los dentro da janela. Para isso, o **Tkinter** oferece três "gerenciadores de geometria": `pack()`, `grid()` e `place()`.

Importante: Para evitar erros, evite misturar diferentes gerenciadores na mesma janela ou frame. Escolha o mais adequado para o seu layout.

1. O Gerenciador `pack()`

O `pack()` é o mais simples e ideal para layouts lineares. Ele "empacota" os widgets em blocos e os posiciona lado a lado, um após o outro. Por padrão, ele os empilha verticalmente. Você pode usar o argumento `side` para controlar a direção (`"top"`, `"bottom"`, `"left"`, `"right"`).

Veja um exemplo de como empilhar três botões verticalmente e depois horizontalmente:

```
Python

import tkinter as tk

janela = tk.Tk()

# Exemplo de empilhamento vertical (padrão)
botao1 = tk.Button(text="Botão 1")
botao2 = tk.Button(text="Botão 2")
botao3 = tk.Button(text="Botão 3")

botao1.pack()
botao2.pack()
botao3.pack()

# Para empilhar horizontalmente, usamos o argumento side="left"
# Você pode usar left, right, top ou bottom
botao4 = tk.Button(text="Esquerda")
botao5 = tk.Button(text="Direita")

botao4.pack(side="left")
botao5.pack(side="right")

janela.mainloop()
```

2. O Gerenciador `grid()`

O `grid()` é o gerenciador mais flexível e recomendado para layouts mais complexos, como formulários. Ele organiza os widgets em uma grade (ou tabela) com linhas (`row`) e colunas (`column`).

Você pode especificar a linha e a coluna onde cada widget deve ficar.

```python
import tkinter as tk

janela = tk.Tk()
janela.title("Formulário de Login")

# Cria os widgets para o formulário
label_usuario = tk.Label(text="Usuário:")
entrada_usuario = tk.Entry()
label_senha = tk.Label(text="Senha:")
entrada_senha = tk.Entry(show="*")
botao_login = tk.Button(text="Entrar")

# Posiciona os widgets na grade usando grid()
label_usuario.grid(row=0, column=0)
entrada_usuario.grid(row=0, column=1)
label_senha.grid(row=1, column=0)
entrada_senha.grid(row=1, column=1)
botao_login.grid(row=2, column=1)

janela.mainloop()
```

3. O Gerenciador `place()`

O `place()` permite um controle absoluto sobre a posição dos widgets, usando coordenadas `x` e `y`. Embora ofereça precisão, ele não é muito usado, pois o layout não se adapta automaticamente quando a janela é redimensionada. É útil para posicionar elementos em pontos fixos ou para sobrepor widgets.

```python
import tkinter as tk

janela = tk.Tk()
janela.title("Posição Absoluta")

# Define o tamanho da janela
janela.geometry("300x200")

# Cria um widget e o posiciona com coordenadas fixas
label = tk.Label(text="Posicionado em 50, 50")
label.place(x=50, y=50)

janela.mainloop()
```

Agora que você sabe como organizar os widgets na sua interface, estamos prontos para a próxima etapa: fazer com que eles realizem ações.

Capítulo 3: Criando Aplicações Interativas

Um programa de interface gráfica se torna realmente útil quando o usuário pode interagir com ele. No **Tkinter**, isso é feito associando uma ação (uma função) a um evento, como o clique de um botão.

1. O Parâmetro `command`

O widget `Button` possui um parâmetro chamado **`command`** que nos permite dizer qual função deve ser executada quando o botão for clicado.

O segredo aqui é passar o **nome da função** sem os parênteses `()`. Se você os usar, a função será executada imediatamente, quando o programa iniciar, e não quando o botão for clicado.

Veja um exemplo simples onde um botão imprime uma mensagem no console:

```python
import tkinter as tk


# Define a função que será executada
def mostrar_mensagem():
    print("O botão foi clicado!")


janela = tk.Tk()


# Cria o botão e associa a função ao parâmetro 'command'
# Note que passamos apenas o nome da função (mostrar_mensagem), sem os '()'
botao = tk.Button(text="Clique Aqui", command=mostrar_mensagem)
botao.pack()


janela.mainloop()
```

2. Criando uma Aplicação Prática

Agora, vamos criar uma aplicação um pouco mais elaborada que usa um campo de texto (`Entry`) e um botão para mostrar uma mensagem personalizada. Para isso, vamos usar o módulo **`tkinter.messagebox`**, que exibe caixas de diálogo.

```python
import tkinter as tk
from tkinter import messagebox


# A função agora recebe um nome e exibe uma caixa de diálogo
def saudar_usuario():
    # Obtém o texto do campo de entrada
    nome_usuario = entrada_nome.get()

    # Verifica se o campo não está vazio antes de saudar
    if nome_usuario:
        messagebox.showinfo("Saudação", f"Olá, {nome_usuario}!")
    else:
        messagebox.showwarning("Aviso", "Por favor, insira seu nome.")


janela = tk.Tk()
janela.title("Saudação")

# Cria e posiciona o Label e o Entry
label_nome = tk.Label(text="Digite seu nome:")
label_nome.pack()
entrada_nome = tk.Entry()
entrada_nome.pack()
```

```
# Cria o botão e o associa à função 'saudar_usuario'
botao_saudar = tk.Button(text="Saudar", command=saudar_usuario)
botao_saudar.pack()

janela.mainloop()
```

Com este capítulo, você já tem as ferramentas para criar janelas, adicionar elementos e fazê-los interagir com o usuário.

Capítulo 4: Interagindo com Dados (CRUD Básico)

Neste capítulo, vamos mergulhar na parte mais avançada do seu eBook: como manipular dados. Para exibir informações em formato de tabela, o **Tkinter** oferece o widget **Treeview**, que faz parte da biblioteca mais moderna ttk.

1. Mostrando Dados em uma Tabela com Treeview

O Treeview é perfeito para o que você precisa. Para usá-lo, vamos primeiro simular uma tabela com uma lista de dados.

O código a seguir cria uma janela, define as colunas da nossa tabela (ID, Nome, Preço) e preenche o Treeview com dados de exemplo.

```
Python
import tkinter as tk
from tkinter import ttk

janela = tk.Tk()
janela.title("Tabela de Materiais")

# Define as colunas da tabela
colunas = ('id', 'nome', 'preco')
tabela = ttk.Treeview(janela, columns=colunas, show='headings')
```

```
# Define os títulos das colunas
tabela.heading('id', text='ID')
tabela.heading('nome', text='Nome')
tabela.heading('preco', text='Preço (R$)')

# Dados de exemplo para a tabela
dados = [
    (1, 'Granito Preto', 350.00),
    (2, 'Mármore Carrara', 700.50),
    (3, 'Quartizito Mont Blanc', 1200.00)
]

# Insere os dados na tabela
for dado in dados:
    tabela.insert('', 'end', values=dado)

tabela.pack()

janela.mainloop()
```

2. Adicionando Novos Dados à Tabela

Agora que sabemos como mostrar os dados, vamos adicionar a funcionalidade de incluir novos materiais. Para isso, criaremos um formulário simples com campos de entrada e um botão para "Adicionar".

A função `adicionar_material()` obtém os valores dos campos de texto e insere uma nova linha no `Treeview`.

Python

```
import tkinter as tk
from tkinter import ttk


janela = tk.Tk()
janela.title("Gerenciador de Materiais")


# Cria e configura a tabela como no exemplo anterior
colunas = ('id', 'nome', 'preco')
tabela = ttk.Treeview(janela, columns=colunas, show='headings')
tabela.heading('id', text='ID')
tabela.heading('nome', text='Nome')
tabela.heading('preco', text='Preço (R$)')
# ... (restante do código para criar e popular a tabela) ...


tabela.pack(pady=10)


# ---------- PARTE DO FORMULÁRIO ----------
# Cria os campos de entrada de dados
frame_form = tk.Frame(janela)
frame_form.pack()


label_nome = tk.Label(frame_form, text="Nome:")
label_nome.pack(side='left')
entrada_nome = tk.Entry(frame_form)
entrada_nome.pack(side='left')
```

```
label_preco = tk.Label(frame_form, text="Preço:")
label_preco.pack(side='left')
entrada_preco = tk.Entry(frame_form)
entrada_preco.pack(side='left')

# Função para adicionar um novo material
def adicionar_material():
    # Pega o próximo ID disponível
    novo_id = len(tabela.get_children()) + 1
    nome = entrada_nome.get()
    preco = entrada_preco.get()

    # Insere a nova linha na tabela
    tabela.insert('', 'end', values=(novo_id, nome, preco))

    # Limpa os campos de entrada
    entrada_nome.delete(0, 'end')
    entrada_preco.delete(0, 'end')

botao_adicionar = tk.Button(janela, text="Adicionar", command=adicionar_material)
botao_adicionar.pack(pady=5)

janela.mainloop()
```

3. Excluindo Dados da Tabela

Para finalizar, vamos adicionar a funcionalidade de exclusão. Um botão permitirá ao usuário apagar uma linha que foi selecionada.

A função `remover_material()` obtém o item selecionado e o remove da tabela.

```python
import tkinter as tk
from tkinter import ttk

janela = tk.Tk()
janela.title("Gerenciador de Materiais (CRUD)")

# Cria a tabela e o formulário de adição (código do passo anterior)
# ...

# Função para remover o material selecionado
def remover_material():
    # Obtém o ID do item selecionado na tabela
    item_selecionado = tabela.selection()
    if item_selecionado:
        # Deleta o item do Treeview
        tabela.delete(item_selecionado)

botao_remover = tk.Button(janela, text="Remover Selecionado", command=remover_material)
botao_remover.pack(pady=5)

janela.mainloop()
```
