



Material de Apoio do Curso

INTERFACE HOMEM MÁQUINA MICROCONTROLADA

Material:

Interface Homem Máquina Microcontrolada

Ministrantes do Curso:

Guilherme Franco

Reinan Lima

Data do Curso:

10/12/2024

Apresentação

Nesse curso será abordado como desenvolver aplicativos desktop, interfaces, que sejam capazes de se comunicar com sistemas embarcados, os sistemas criados com microcontroladores. Para criação das interfaces usaremos Python como linguagem de programação, porque ela é uma das mais usadas no mundo e sua usabilidade se estende a qualquer área, seja criação de códigos simples, análise de dados, criação de interfaces, API's da web, cálculos matemáticos numéricos e simbólicos.

O Python nativamente já trás como parte da sua biblioteca padrão, então não é preciso instalar nada de terceiros para criar uma interface das mais simples, um pacote que nos permite criar interfaces, esse pacote chamado [Tkinter](#), é um sistema que encobre o framework Tk da linguagem de programação TCL.

Hello World

Para iniciar, vamos criar nossa primeira janela, uma janela simples com um botão, e nesse botão deve ter escrito a frase "Hello World".

Abaixo, o código usado para inicializar essa aplicação:

```
from tkinter import *
from tkinter import ttk

window = Tk()

def hello():
    print("Hello World")

ttk.Button(window, text="Hello World!", command=hello).pack()

window.mainloop()
```

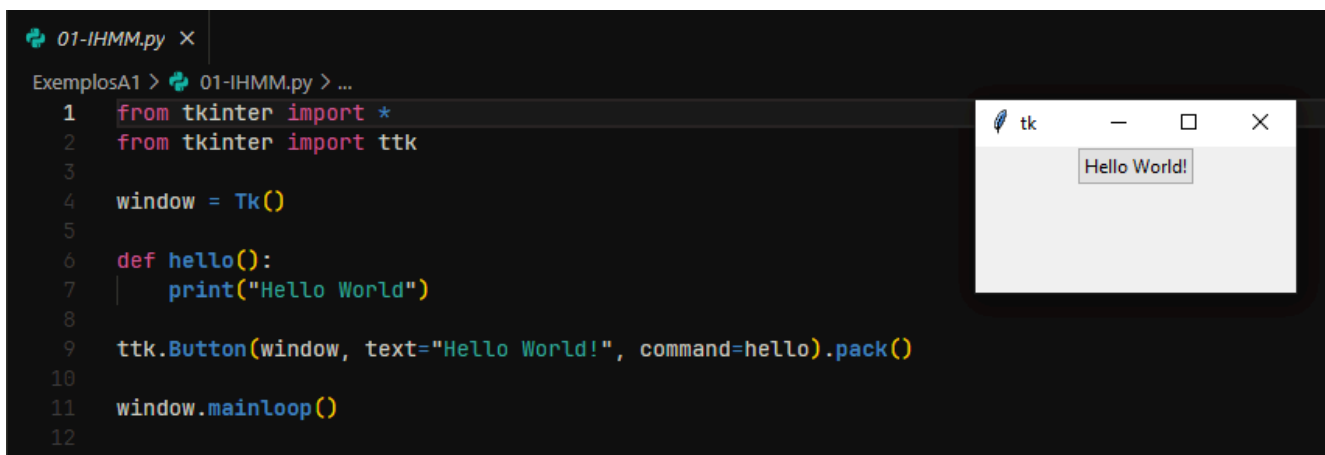
Explicando o que está acontecendo, nas linhas 1 e 2 estamos importando do pacote `tkinter`, todos os elementos disponíveis, e explicitando que estamos importando especificamente o módulo `ttk`.

A linha 4 chama uma instância da classe `Tk` e armazena na variável `window`, podemos pensar que essa variável é um painel onde você pode adicionar e posicionar os componentes da interface.

A linha 6 está declarando uma função, que apenas exibe no console a frase "Hello World".

A linha 9 cria a partir da classe `ttk`, um botão inserido na instancia `window`, com texto definido como `"Hello World"`, e como parâmetro de comando (`command`), uma referência da função `hello` declarada.

A linha 11 chama o método `mainloop` que a variável `window` herda da classe `Tk`. Esse `mainloop` é o manipulador de eventos do Tkinter que consegue exibir e detectar eventos na interface. Ele é o método usado para exibir a janela que criamos.

A screenshot of a code editor with a dark theme. The file is named '01-IHMM.py'. The code imports Tkinter and ttk, creates a Tk window, defines a 'hello' function that prints 'Hello World', creates a ttk.Button with the text 'Hello World!' and the 'hello' command, packs the button, and starts the mainloop. To the right, a small window titled 'tk' is shown with a single button that says 'Hello World!'.

Uma coisa que você pode ter notado, é que a estilização que o Tkinter utiliza, é o tema nativo do computador em que o programa está sendo executado, esse programa foi rodado em um Windows 10, e usando Tkinter dessa maneira deixa a nossa aplicação com um aspecto de computador antigo, mas a comunidade do Python conseguiu desenvolver várias maneiras de tematizar e modernizar as interfaces criadas com Tkinter.

Uma das mais famosas é chamada de [CustomTkinter](#), que permite customizar cada um dos widgets e ainda tem uma interface muito mais moderna. Vamos ver como fica a criação do Hello World usando CTK(CustomTkinter):

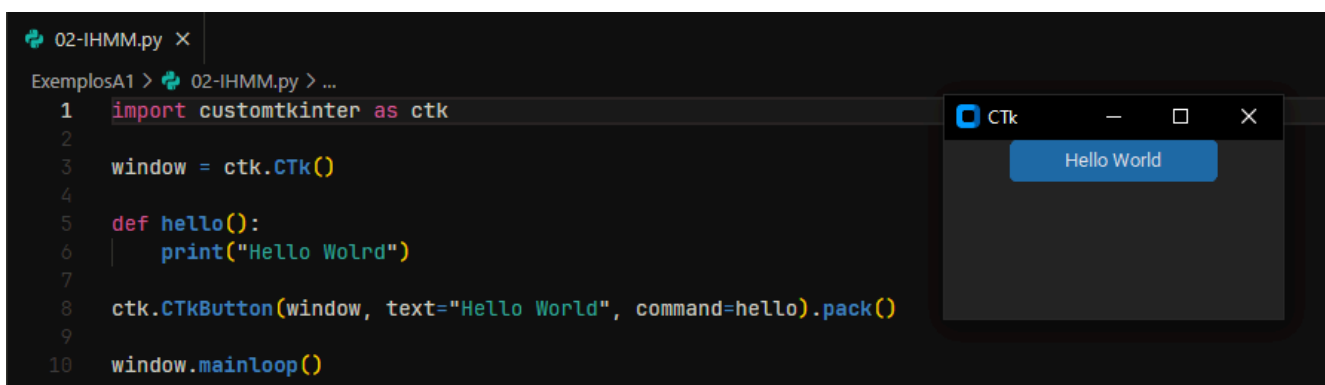
```
import customtkinter as ctk

window = ctk.CTk()

def hello():
    print("Hello Wolrd")

ctk.CTkButton(window, text="Hello World", command=hello).pack()

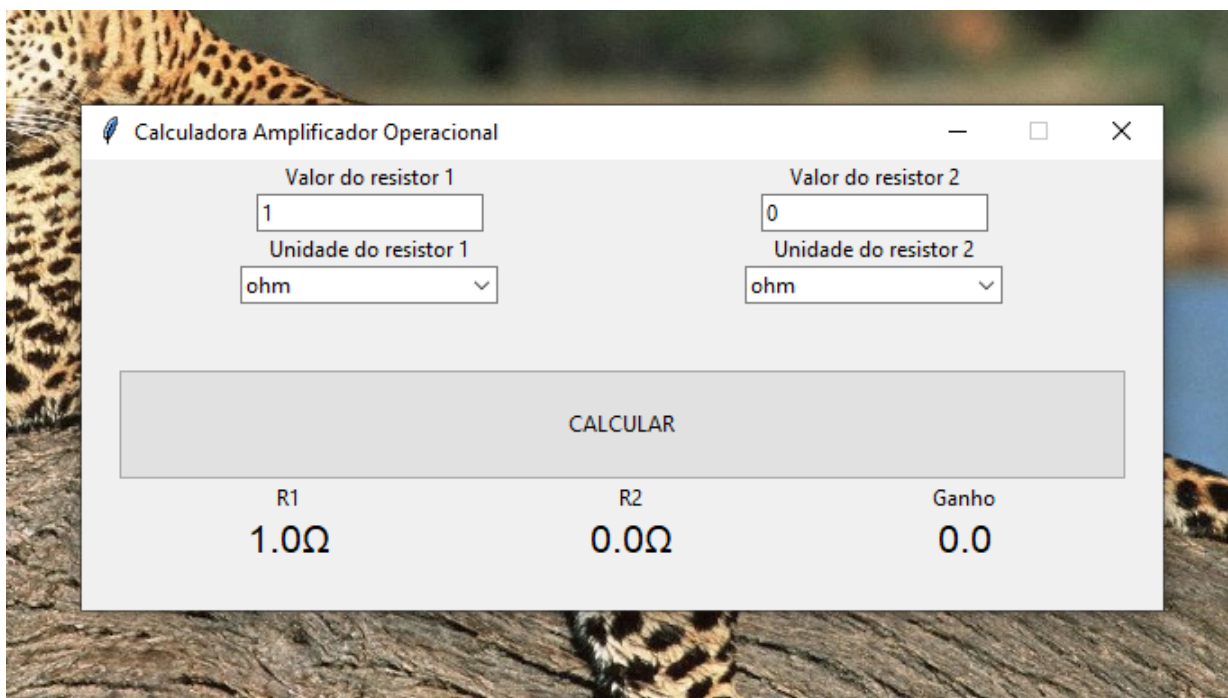
window.mainloop()
```

A screenshot of a code editor with a dark theme. The file is named '02-IHMM.py'. The code imports customtkinter as ctk, creates a CTk window, defines a 'hello' function that prints 'Hello Wolrd' (note the typo), creates a CTkButton with the text 'Hello World' and the 'hello' command, packs the button, and starts the mainloop. To the right, a small window titled 'CTk' is shown with a blue button that says 'Hello World'.

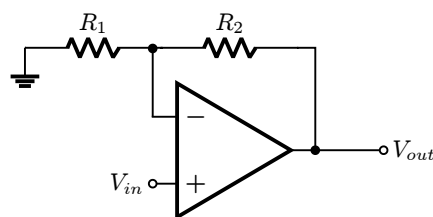
O CTK permite customizar o tema da aplicação, seja modo escuro ou claro, e até as cores dos widgets de forma muito mais fácil que o Tkinter.

Demonstração AmpOp

Depois de ter visto como a gente pode iniciar nosso projeto para criar uma interface simples, vou apresentar pra vocês um exemplo, primeiro usando Tkinter e depois como ele fica usando CTK, pra demonstrar o que é possível fazer com essas ferramentas. Essa interface foi criada pra demonstração, e é bem simples, tanto em aparência, quanto em funcionalidade, mas o conteúdo que foi necessário pra criá-la é bem diverso, envolvendo posicionamento no grid, armazenar e utilizar variáveis reativas, funções de callback no botão, e mais.



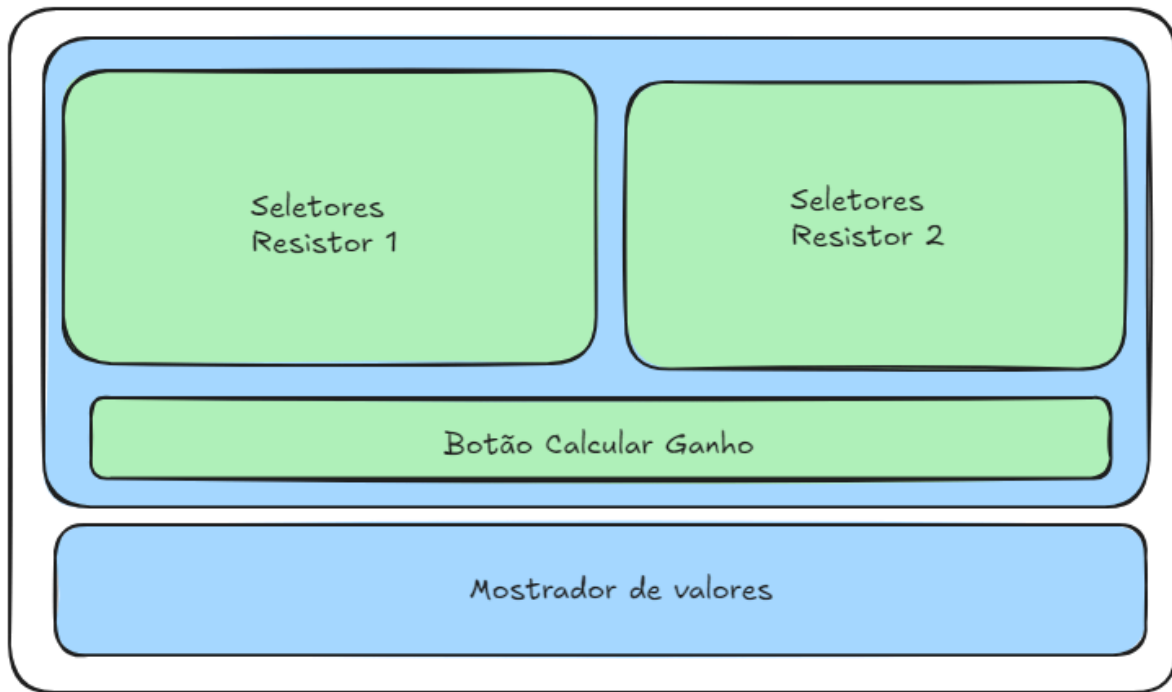
Esse programa consegue a partir da entrada de valores de dois resistores, calcular o ganho de um amplificador não inversor, ainda sendo possível escolher qual a unidade do resistor sendo aplicada, seja em ohms (Ω) ou em kilohms.



Analizando Layout

Nessa interface, o `grid` foi o principal método usado para dividir e posicionar os componentes na tela. E o `pack` foi usado para adicionar de forma fácil os componentes. O `grid` é o método de posicionamento que divide a interface em uma grade com colunas e linhas e podemos posicionar qualquer um dos widgets em qualquer combinação de coluna e linha escolhida.

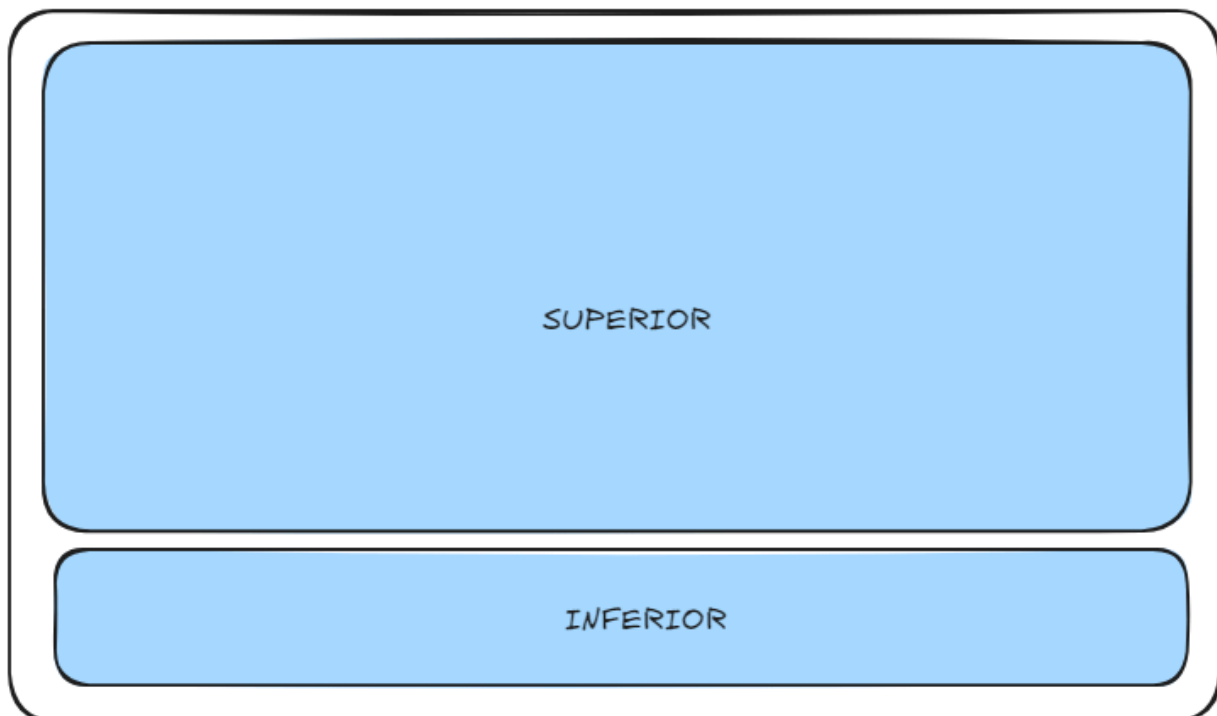
Interface



Nessa imagem é possível ver na totalidade como ficou organizado a aplicação criada, vários frames ficaram posicionados dentro de outros frames, isso mostra que é com Tkinter pode ser simples, mas também podemos criar interfaces mais robustas e complexas.

A diferença de cores na imagem, mostra como ficou organizado as camadas de posicionamento. Vamos separar cada uma das camadas para entender melhor:

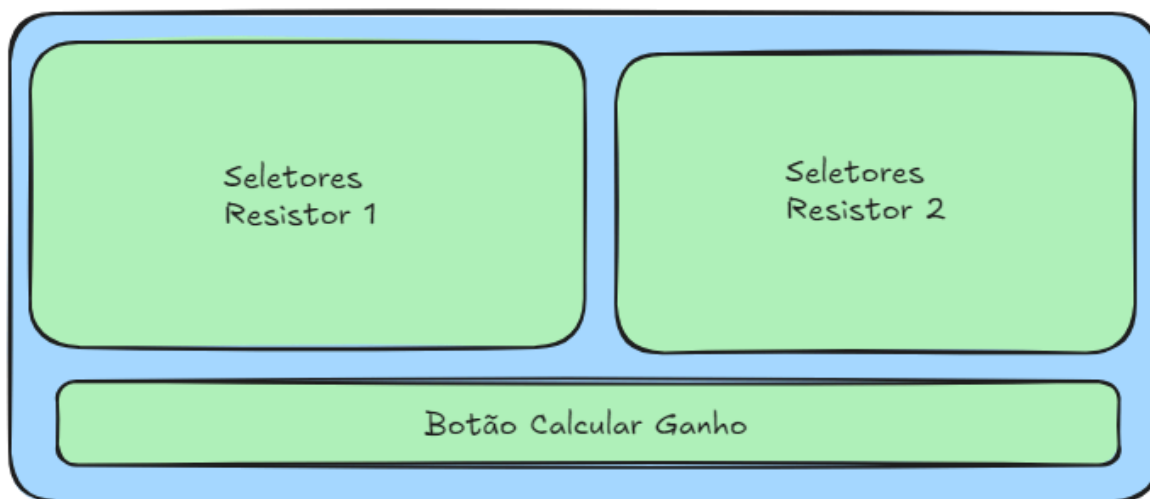
Janela Principal



A janela principal é dividida em uma única coluna com duas linhas, onde a primeira linha tem peso 3, e a segunda linha tem peso 1. Os pesos associados a linhas e colunas determinam quanto do espaço da tela ela vai ocupar, então a janela principal vai ficar dividida de uma forma que teremos duas linhas, a primeira com $\frac{3}{4}$ ou 75%, e a segunda ocupará $\frac{1}{4}$ da tela ou 25%.

Eu decidi separar a interface dessa maneira, para organizar os componentes onde o usuário vai clicar e escrever, na parte superior, e as informações calculadas e valores devem ser mostrados na parte inferior.

Parte Superior



Na parte superior, como mostrado anteriormente, é a área onde o usuário vai digitar o valor dos resistores e calcular o valor do ganho através de um botão. O valor em ohms é digitado em um widget chamado `Entry`, e a unidade em um `Combobox`.

O botão tem o texto `Calcular` escrito, e quando clicado executa um evento de callback que calcula o valor do ganho a partir do valor dos resistores selecionados, e atualiza o mostrador na parte inferior da janela principal.

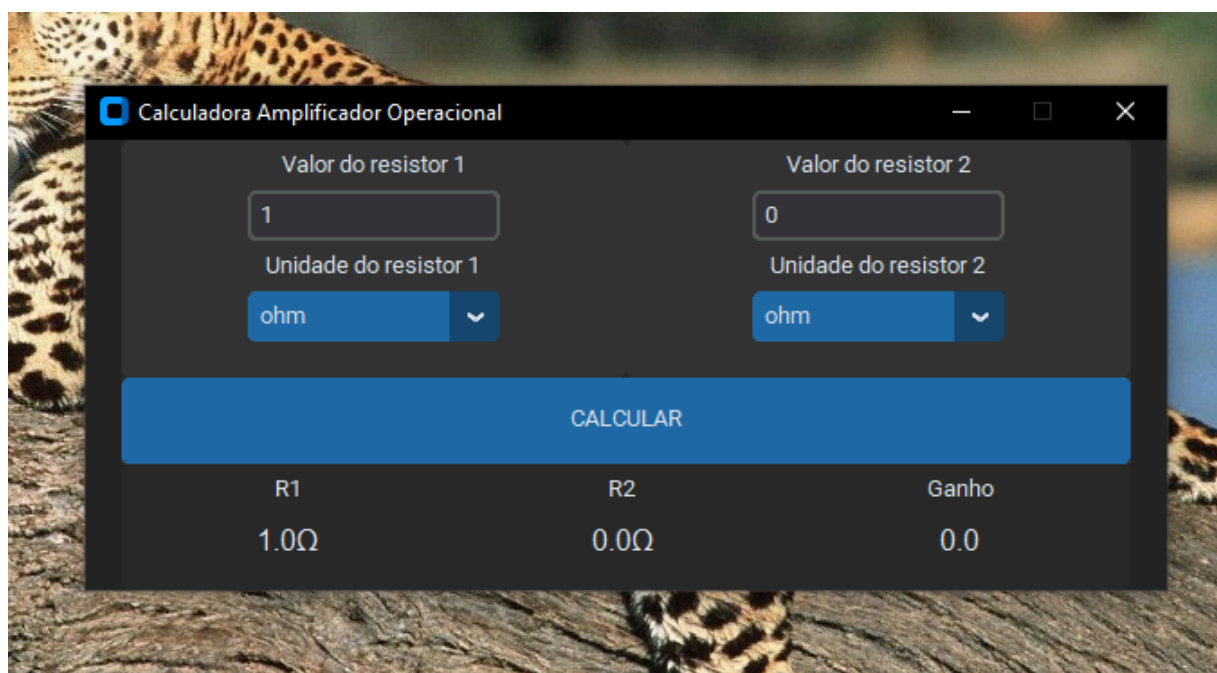
Um callback é uma função que é chamada em resposta a um evento específico, nesse caso, ao clicar no botão.

Parte Inferior



Na parte inferior, são posicionadas em 3 colunas, mostradores para o R1, R2 e ganho, onde os valores sempre são atualizados no momento que o botão de calcular é pressionado.

Agora voltando na questão de Tkinter e CustomTkinter, acima já foi mostrado o design da interface usando a biblioteca original Tkinter.



Agora temos o seu correspondente usando CTK, e ele possui um visual muito mais moderno, suporta modo escuro e tem muito mais decorações.

Conceitos

Widgets

Como foi mostrado as duas maneiras que podemos criar interfaces, usando Tkinter, a biblioteca original nativa do python, ou usando uma biblioteca externa chamada CustomTkinter que possui um visual moderno, mas com as mesmas funcionalidades e mais suporte a pacotes externos e customizações extras.

A partir de agora, usaremos CustomTkinter ou CTK, para criação das nossas interfaces.

Entrando agora na parte de conceitos, algo muito importante que precisamos entender é o que é um widget.

O tkinter considera todos os seus componentes um, um widget. Nos exemplos anteriores, tanto no Hello World, quanto na calculadora de ganho do amplificador, usamos widgets de botões, Label, Entry, e muitos outros que a biblioteca oferece.

Os widgets são objetos na linguagem do python, quer dizer que chamamos instancias das classes que representam botões, frames, etc para adicioná-los a nossa interface.

Outro ponto, é que além da classe do widget, precisamos de mais alguma coisa pra criar esse widget, precisamos saber quem vai ser o `parent`, ou seja, o widget pai.

Quando criamos o botão do Hello World, referenciamos o `window` nos argumentos da classe `CTkButton`.

```
ctk.CTkButton(window, text="Hello World", command=hello).pack()
```

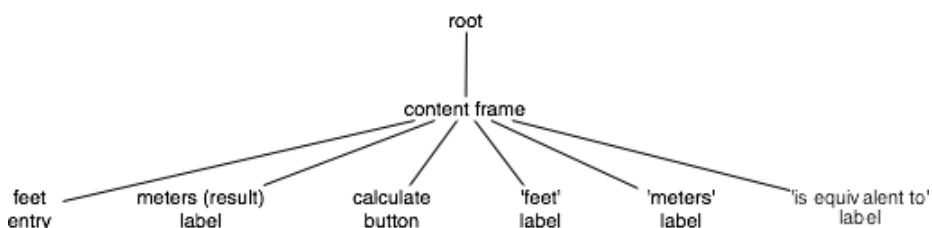
Todos os widgets que declaramos para nossa interface, precisam de um parent, e declaramos isso no argumento na posição `master`. Ao verificar os argumentos que a classe do botão recebe, podemos ver todas as opções, e a primeira é `master`, o qual se refere a qual o parent aquele botão fará parte.

Então se no botão, preenchemos o campo `master` com o valor `window`, declarado anteriormente como nossa janela principal, então nosso botão estará contido na janela principal.

```
02-IHMM.py x
ExemplosA1 > 02-IHMM.py > ...
1 import customtkinter as ctk
2
3 window = ctk.CTk()
4
5 def hello():
6     print("Hello Wolrd")
7
8 ctk.CTkButton(window, text="Hello World", command=hello).pack()
9
10 class CTkButton(
    master: Any,
    width: int = 140,
    height: int = 28,
    corner_radius: int | None = None,
    border_width: int | None = None,
    border_spacing: int = 2,
    bg_color: str | Tuple[str, str] = "transparent",
    fg_color: str | Tuple[str, str] | None = None,
    hover_color: str | Tuple[str, str] | None = None,
    border_color: str | Tuple[str, str] | None = None,
    text_color: str | Tuple[str, str] | None = None,
    text_color_disabled: str | Tuple[str, str] | None = None,
```

Isso cria uma hierarquia entre os widgets. Vamos imaginar por exemplo uma interface para calcular a conversão de valores de km para pés e vice versa. Nesse exemplo criamos um frame, uma espécie de container que pode armazenar outros widgets, esse frame tem como master a janela principal. e adicionamos algumas labels, botões e entrys para receber o valor em km e calcular o resultado.

Teremos algo mais ou menos como isso, como uma representação da hierarquia dos widgets dessa interface



Opções de Configuração

Todos os widgets tem várias opções de configurações que controlam como ele é exibido ou como ele se comporta.

As opções disponíveis dependem da classe do widget é usada, mas há algumas opções que são comuns entre diversos widgets. Por exemplo a configuração do `master` que está presente em todos os widgets.

Essas configurações podem ser definidas logo na criação da instância da classe do widget, podemos em outro momento verificar qual o valor atual definido para aquela configuração além de poder modificá-lo.

Por exemplo, definimos um simples botão:

```
button = ctk.CTkButton(window, text="Hello World", command=hello)
button.pack()
```

Podemos alterar os parâmetros já definidos, em um momento posterior, utilizando o método `.configure` do widget que definimos. Todos os widgets podem usar esse método.

```
button.configure(text="goodbye")
```

E usando o método `configure` é capaz de mudar mais de uma opção de configuração do widget, por exemplo podemos mudar o valor de `text` e o `command`:


```
button.configure(text="New text", command=lambda:print("new"))
```

Variáveis de Estado

Nós podemos a partir das variáveis de estado, guardar o valor atual de um widget, os widgets como `Entry`, `Checkbutton` ou `Scale`, tem a opção de associar a eles uma variável do Tkinter que pode guardar o valor atual delas.

O Tkinter tem 5 classes(tipos) de variáveis que podem ser usados para armazenar valores.

Geral

A classe `Variable` consegue guardar o valor dos widgets, e seu valor pode ser de qualquer tipo, inteiro, float, string ou booleano.

```
ctk.Variable()
```

Inteiro

A classe `IntVar` é uma subclasse de `Variable`, quer dizer que ela herda todos os atributos de `Variable`, mas ela é especificamente usada para guardar valores **inteiros**, por exemplo, guardar valores de widgets como `Scale`.

```
ctk.IntVar()
```

Float

A classe `DoubleVar` é uma subclasse de `Variable`, quer dizer que ela herda todos os atributos de `Variable`, mas ela é especificamente usada para guardar valores **decimais**(float/double).

```
ctk.DoubleVar()
```

String

A classe `StrVar` é uma subclasse de `Variable`, quer dizer que ela herda todos os atributos de `Variable`, mas ela é especificamente usada para guardar **textos**(string).

```
ctk.StringVar()
```

Boolean

A classe `BooleanVar` é uma subclasse de `Variable`, quer dizer que ela herda todos os atributos de `Variable`, mas ela é especificamente usada para guardar valores **binários**(booleanos, valores como True ou False).

```
ctk.BooleanVar()
```

Introdução a Classes

Os exemplos e demonstrações anteriores usavam uma abordagem de escrita de código em forma de script, fazendo uso de declaração de variáveis globais e funções simples. Em aplicações reais, o comum é que a interface fique mais robusta, o que resulta em um código mais extenso, mais funções precisam ser declaradas e muitos widgets adicionados, e para isso uma boa organização é essencial.

Há várias maneiras de organizar o código de uma forma mais profissional, usando módulos, declarando funções e classes. Aqui iremos desenvolver os exemplos e demonstrações declarando classes.

Declarando o código da interface usando uma classe, nos permite organizar de uma forma muito melhor, separar frames diferentes em métodos que podem ser chamados, isso simplifica a organização do código e deixa mais fácil para uma manutenção posterior.

Então pra começar, como declaramos uma classe ?

```
class App:
    ...
```

Podemos declarar uma classe de uma maneira muito parecida a declarar uma função, passamos a keyword `class` seguido do nome da classe a ser criada.

Para classes, variáveis que são declaradas em seu escopo são chamadas de **propriedades**, e funções declaradas em seu escopo são chamadas de **métodos**.

Para as classes que serão criadas no curso, declaramos um método chamado de *dunder method* `__init__`, é um método de classe que é executada toda vez que a classe é instanciada.

Classes são usadas para declarar objetos com propriedades e métodos internos, então podemos nos basear em objetos da vida real e simular uma versão deles, por exemplo classes que representem resistores e capacitores.

```
class Resistor:
    def __init__(self, resistencia:float, tolerancia:float):
        self.resistencia = resistencia
        self.tolerancia = tolerancia

    def info(self):
        return f"Resistor: {self.resistencia}R +- {self.tolerancia}%"

    def __add__(self, other:"Resistor"):
        return self.resistencia + other.resistencia

    def __mul__(self, other:"Resistor"):
        return round(1 / ( (1/self.resistencia) + (1/other.resistencia)), 4)
```

```
class Capacitor:
    def __init__(self, capacitancia:float, tensao_maxima:float):
        self.capacitancia = capacitancia
        self.tensao_maxima = tensao_maxima

    def info(self):
        return (f"Capacitor: {self.capacitancia}F, " +
            f"Tensão Máxima: {self.tensao_maxima}V")
```

Com uma abordagem como essa, diversas aplicações e possibilidades surgem, por exemplo, usando as classes de `Resistor` e `Capacitor` criadas acima, pode ser possível criar uma espécie de simulador ou calculadora fazendo uso desses componentes eletrônicos.

Hello World em OOP

Foi demonstrado com criar, declarar e utilizar classes em python, podemos voltar para a parte de interface, e recriar o "Hello World" que foi feito anteriormente com formato de script, mas dessa vez utilizando os conceitos de classe.

```
import customtkinter as ctk

class HelloWorld:
    def __init__(self, window: ctk.CTk):

        self.window = window

        def hello():
            print("Hello World")

        ctk.CTkButton(self.window, text="my button", command=hello).pack()

app = ctk.CTk()
HelloWorld(app)
app.mainloop()
```

- Nessa abordagem, estamos importando o `customtkinter` com o apelido de `ctk`.
- Estamos declarando a classe `HelloWorld`, com um botão que quando clicado, exibe no console a frase "Hello World".
- Estamos instanciando a classe `CTk`, do módulo importado `ctk`, dentro da variável `app`.
- Estamos instanciando a classe `HelloWorld` e adicionando seu conteúdo criado, dentro de `app`.
- Estamos chamando o método `mainloop` para executar a janela.

Widgets Principais

Frame

Widget no Tkinter: `ttk.Frame`

Widget no CustomTkinter: `ctk.CTkFrame`

Documentação: <https://customtkinter.tomschimansky.com/documentation/widgets/frame>

Descrição

É um widget que funciona como um container para outros widgets, serve para agrupar widgets relacionados, o que pode facilitar a organização visual e lógica da interface.

Como é usado

O `CTkFrame` é instanciado dentro de uma variável, para podermos referenciá-lo em outros widgets.

```
# Adicionando um CTkFrame
frame = ctk.CTkFrame(
    master=window,
    width=300,
```

```
height=200,
corner_radius=15, # Bordas arredondadas
border_width=2,   # Espessura da borda
border_color="white", # Cor da borda
fg_color="gray20"   # Cor de fundo
)
frame.pack(padx=20, pady=20)
```

E para adicionar widgets dentro desse frame, fazemos uma referência a ele, no argumento `master` de outros widgets, por exemplo:

```
# Adicionando widgets no CTkFrame
label = ctk.CTkLabel(master=frame, text="Bem-vindo ao CTkFrame!")
label.pack(pady=10)
```

Principais opções de configuração

Opção	Descrição	Valor padrão
<code>master</code>	Especifica o widget pai que conterá o frame.	Nenhum
<code>width</code>	Define a largura do frame.	200
<code>height</code>	Define a altura do frame.	200
<code>corner_radius</code>	Define o raio das bordas arredondadas (em pixels).	10
<code>fg_color</code>	Cor de fundo do frame (aceita strings como <code>"gray"</code> , códigos hexadecimais ou valores RGBA).	"default"
<code>border_width</code>	Define a espessura da borda em torno do frame.	0
<code>border_color</code>	Cor da borda em torno do frame.	"default"
<code>bg_color</code>	Cor do plano de fundo, usada apenas se o <code>fg_color</code> estiver definido como <code>"transparent"</code> .	"default"
<code>hover</code>	Ativa ou desativa a mudança visual ao passar o mouse sobre o frame.	False
<code>grid_rowconfigure</code>	Configura o peso e a expansão de linhas no layout de grade (opção adicional para layout).	None

Label

Widget no Tkinter: `ttk.Label`

Widget no CustomTkinter: `ctk.CTkLabel`

Documentação: <https://customtkinter.tomschimansky.com/documentation/widgets/label>

Descrição

Esse widget representa o rótulo de texto ou imagem. Ele é usado para exibir informações estáticas na interface.

Como é usado

O `CTkLabel` é instanciado dentro de uma variável, para podermos referenciá-lo em outros widgets.

```
# Criando um CTkLabel simples
label = ctk.CTkLabel(window, text="Olá, CustomTkinter!", text_color="blue")
```

```
label.pack()
```

O CTKLabel pode ser usado para exibir imagens também.

É possível criar uma label dinâmica, onde seu valor é alterado a partir do valor de outro widget. Por exemplo, o valor da label pode ser igual ao texto escrito em um `Entry`:

```
import customtkinter as ctk

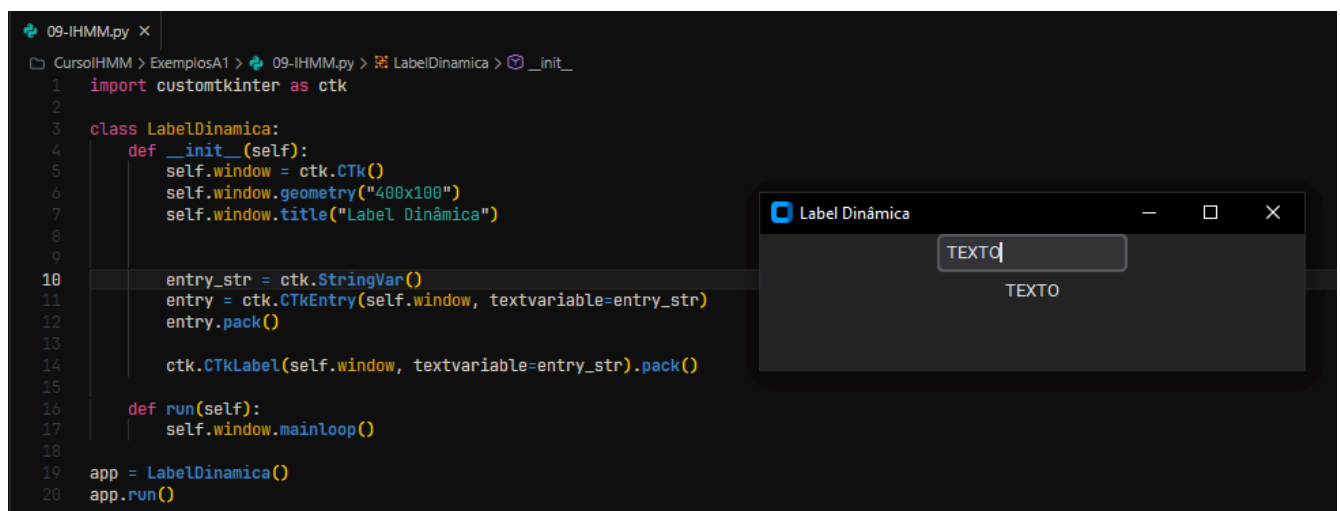
class LabelDinamica:
    def __init__(self):
        self.window = ctk.CTk()
        self.window.geometry("400x100")
        self.window.title("Label Dinâmica")

        entry_str = ctk.StringVar()
        entry = ctk.CTkEntry(self.window, textvariable=entry_str)
        entry.pack()

        ctk.CTkLabel(self.window, textvariable=entry_str).pack()

    def run(self):
        self.window.mainloop()

app = LabelDinamica()
app.run()
```



Nesse exemplo, é possível ver que a variável de estado definida como `entry_str` está associada a ambos `CTKEntry` e `CTKLabel`.

A Entry é responsável por alterar o valor da variável, e a Label a exibe.

Principais opções de configuração

Parâmetro	Descrição	Valor Padrão
<code>text</code>	O texto a ser exibido no rótulo.	<code>""</code> (string vazia)
<code>text_color</code>	Cor do texto. Pode ser uma string com nome da cor ou código hexadecimal.	Dependente do tema
<code>font</code>	Fonte do texto, aceita uma tupla como <code>("NomeFonte", Tamanho, "Estilo")</code> .	<code>None</code>
<code>anchor</code>	Define o alinhamento do texto/imagem (e.g., <code>w</code> , <code>e</code> , <code>center</code>).	<code>"center"</code>

Parâmetro	Descrição	Valor Padrão
<code>image</code>	Exibe uma imagem no rótulo. Deve ser um objeto de imagem criado com o <code>CTkImage</code> .	<code>None</code>
<code>compound</code>	Define a posição do texto em relação à imagem (<code>"top"</code> , <code>"bottom"</code> , <code>"left"</code> , <code>"right"</code> , <code>"center"</code>).	<code>"center"</code>
<code>fg_color</code>	Cor de fundo do rótulo. Aceita string ou <code>None</code> (transparente).	<code>None</code>
<code>corner_radius</code>	Define o raio de curvatura dos cantos do rótulo.	<code>0</code> (bordas retas)
<code>width</code>	Largura do rótulo em pixels.	<code>None</code> (ajuste automático)
<code>height</code>	Altura do rótulo em pixels.	<code>None</code> (ajuste automático)

Entry

Widget no Tkinter: `ttk.Entry`

Widget no CustomTkinter: `ctk.CTkEntry`

Documentação: <https://customtkinter.tomschimansky.com/documentation/widgets/entry>

Descrição

É um widget de entrada de texto que permite ao usuário inserir dados diretamente em um campo de texto. Ele pode ser configurado para entrada de senha, limites e tamanho e outras opções de personalização.

Como é usado

```
entry = ctk.CTkEntry(window, placeholder_text="Digite algo aqui...")
entry.pack()
```

A Entry pode ser configurada para exibir um caractere diferente para o texto, isso pode ser útil para campos de senha:

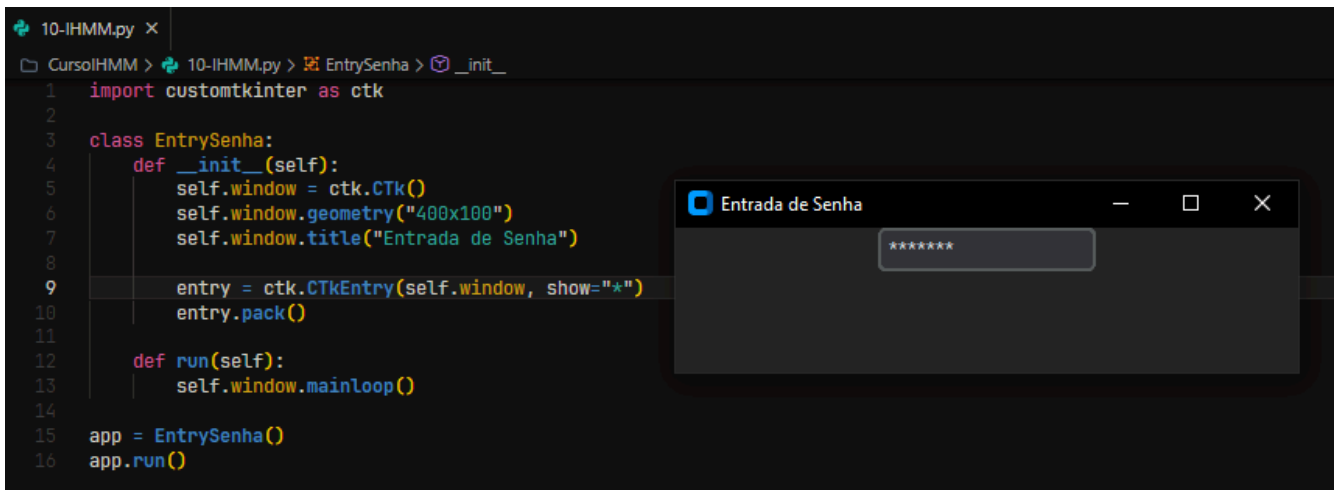
```
import customtkinter as ctk

class EntrySenha:
    def __init__(self):
        self.window = ctk.CTk()
        self.window.geometry("400x100")
        self.window.title("Label Dinâmica")

        entry = ctk.CTkEntry(self.window, show="*")
        entry.pack()

    def run(self):
        self.window.mainloop()

app = EntrySenha()
app.run()
```



Nesse exemplo, estamos usando o parâmetro `show` da classe `CTkEntry`, para fazer com que cada caractere escrito na caixa de texto seja substituído por um `*`.

Principais opções de configuração

Parâmetro	Descrição	Valor Padrão
<code>width</code>	Largura do campo de entrada em pixels.	<code>140</code>
<code>height</code>	Altura do campo de entrada em pixels.	<code>28</code>
<code>placeholder_text</code>	Texto exibido como dica quando o campo está vazio.	<code>None</code>
<code>textvariable</code>	Variável associada para sincronizar o conteúdo do campo.	<code>None</code>
<code>show</code>	Substitui os caracteres inseridos com um símbolo, como <code>*</code> (útil para entradas de senha).	<code>None</code>
<code>fg_color</code>	Cor do fundo do campo de entrada.	Dependente do tema
<code>border_color</code>	Cor da borda do campo.	Dependente do tema
<code>text_color</code>	Cor do texto dentro do campo.	Dependente do tema
<code>font</code>	Fonte do texto, aceita uma tupla como <code>("NomeFonte", Tamanho, "Estilo")</code> .	<code>None</code>
<code>state</code>	Define o estado do campo: <code>"normal"</code> (ativo) ou <code>"disabled"</code> (desativado).	<code>"normal"</code>

Métodos Úteis:

Método	Descrição
<code>get()</code>	Retorna o texto atualmente inserido no campo.
<code>insert(index, text)</code>	Insere o texto na posição especificada pelo índice.
<code>delete(start, end)</code>	Remove o texto entre os índices <code>start</code> e <code>end</code> .
<code>configure(**kwargs)</code>	Altera dinamicamente as opções do widget.

Button

Widget no Tkinter: `ttk.Button`

Widget no CustomTkinter: `ctk.CTkButton`

Documentação: <https://customtkinter.tomschimansky.com/documentation/widgets/button>

Descrição

Esse widget representa um botão interativo, usado para executar ações ou disparar eventos quando clicado.

Como é usado

O `CTkButton` é instanciado dentro de uma variável, para podermos referenciá-lo em outros widgets, mas também pode ser criado apenas instanciando a classe e posicionando na tela.

```
button = ctk.CTkButton(window, text="Clique aqui", command=lambda: print("Botão clicado!"))
button.pack()
```

O botão pode ser acionado a partir do parâmetro `command`. Passamos para `command` uma função, seja ela uma função mais robusta definida pela keyword `def`, ou uma função mais simples definida em um `lambda`:

```
import customtkinter as ctk
import os

class ButtonActions:
    def __init__(self):
        os.system("cls")
        self.window = ctk.CTk()
        self.window.geometry("400x100")
        self.window.title("Ações do Botão")

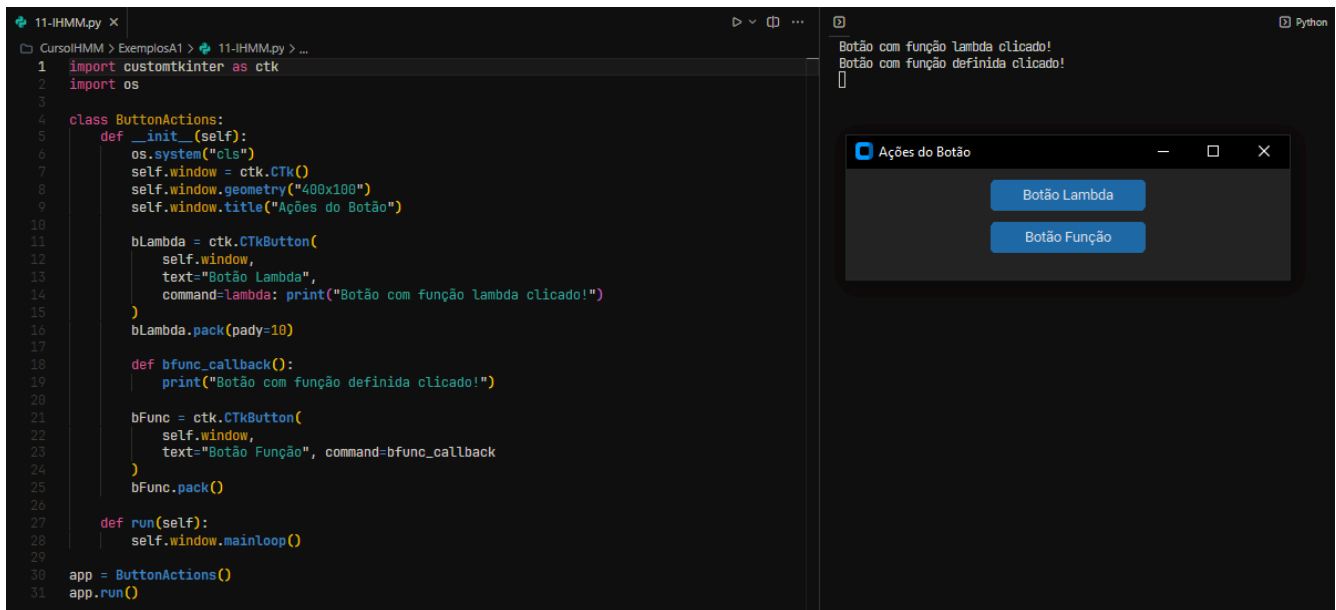
        bLambda = ctk.CTkButton(
            self.window,
            text="Botão Lambda",
            command=lambda: print("Botão com função lambda clicado!")
        )
        bLambda.pack(pady=10)

        def bfunc_callback():
            print("Botão com função definida clicado!")

        bFunc = ctk.CTkButton(
            self.window,
            text="Botão Função", command=bfunc_callback
        )
        bFunc.pack()

    def run(self):
        self.window.mainloop()

app = ButtonActions()
app.run()
```

Nessa exemplo, usamos dois botões com estilos diferentes de callback. O primeiro usando uma função inline denominada lambda, e como o nome diz, quer dizer que podemos definir funções em uma linha.

No segundo exemplo, nós passamos para o parâmetro `command` da classe `Entry` uma referência da função `bfunc_callback`, por isso nós não estamos passando `bfunc_callback()` com parênteses, passar o nome da função com parênteses na verdade vai executar a função, então estaríamos passando para o `command` o retorno da função, e não é isso que queremos, nós queremos passar a referência da função, para que o próprio botão execute a função apenas quando ele for pressionado.

Principais opções de configuração

Parâmetro	Descrição	Valor Padrão
<code>text</code>	Texto exibido no botão.	<code>""</code> (string vazia)
<code>command</code>	Função ou método a ser executado quando o botão é clicado.	<code>None</code>
<code>text_color</code>	Cor do texto no botão.	Dependente do tema
<code>font</code>	Fonte do texto, aceita uma tupla como <code>("NomeFonte", Tamanho, "Estilo")</code> .	<code>None</code>
<code>image</code>	Exibe uma imagem no botão (um objeto <code>CTkImage</code>).	<code>None</code>
<code>compound</code>	Define a posição do texto em relação à imagem (<code>"top"</code> , <code>"bottom"</code> , <code>"left"</code> , <code>"right"</code> , <code>"center"</code>).	<code>"center"</code>
<code>fg_color</code>	Cor de fundo do botão.	Dependente do tema
<code>hover_color</code>	Cor de fundo ao passar o mouse sobre o botão.	Automaticamente calculada
<code>corner_radius</code>	Define o raio de curvatura dos cantos do botão.	<code>10</code>
<code>width</code>	Largura do botão em pixels.	<code>120</code>
<code>height</code>	Altura do botão em pixels.	<code>32</code>
<code>state</code>	Define o estado do botão: ativo (<code>"normal"</code>) ou desativado (<code>"disabled"</code>).	<code>"normal"</code>

Checkbutton

Widget no Tkinter: `ttk.Checkbutton`

Widget no CustomTkinter: `ctk.CTkCheckBox`

Descrição

Um widget que representa uma caixa de seleção(checkbox), usado para permitir ao usuário selecionar ou desmarcar uma opção, funcionando como um interruptor binário para representar estados como "ligado/desligado" ou "verdadeiro/falso".

Como é usado

```
checkbox = ctk.CTkCheckBox(window, text="Aceitar Termos")
checkbox.pack()
```

O checkbox tem a opção de adicionar valores nos seus dois casos, quando selecionado e quando não está selecionado. Isso é feito pelos parâmetros `onvalue` e `offvalue` que podemos escolher qualquer tipo de valor e armazená-los em uma variável de estado correspondente.

```
import customtkinter as ctk
import os

class CheckActions:
    def __init__(self):
        os.system("cls")
        self.window = ctk.CTk()
        self.window.geometry("400x100")
        self.window.title("Checkbutton Actions")

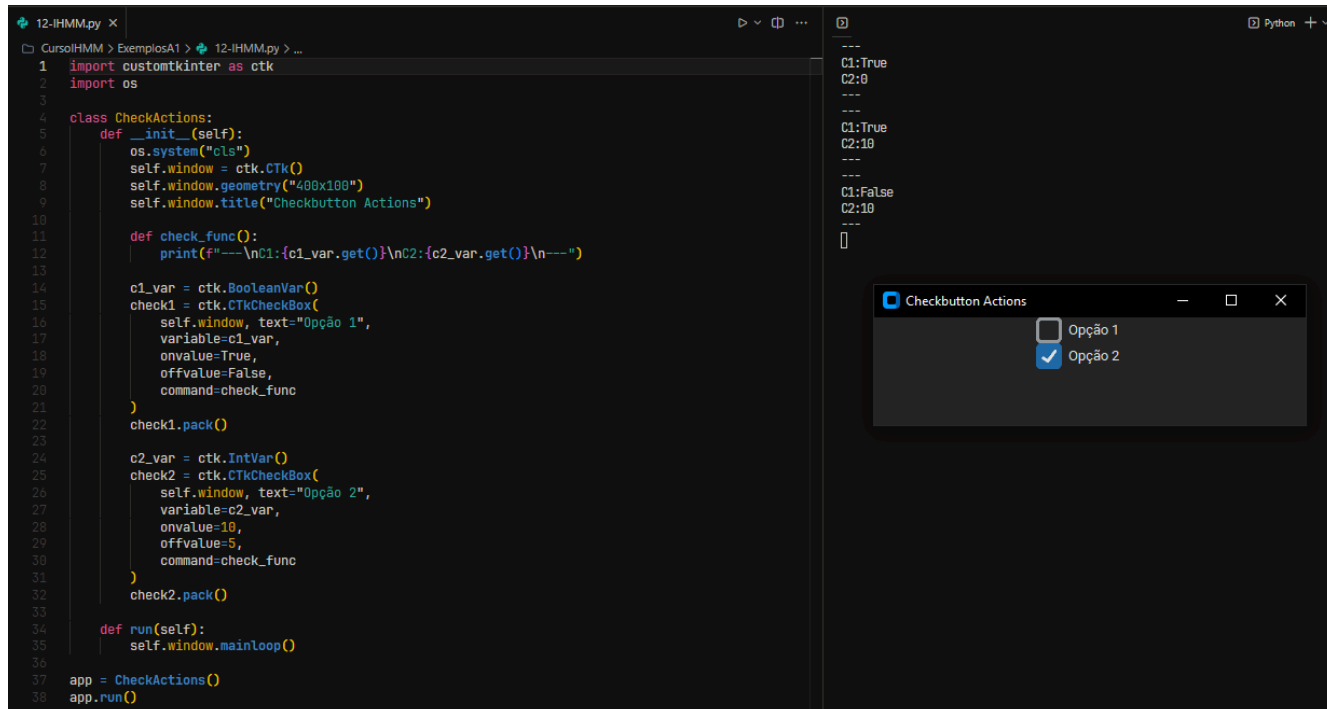
        def check_func():
            print(f"---\nC1:{c1_var.get()}\nC2:{c2_var.get()}\n---")

        c1_var = ctk.BooleanVar()
        check1 = ctk.CTkCheckBox(
            self.window, text="Opção 1",
            variable=c1_var,
            onvalue=True,
            offvalue=False,
            command=check_func
        )
        check1.pack()

        c2_var = ctk.IntVar()
        check2 = ctk.CTkCheckBox(
            self.window, text="Opção 2",
            variable=c2_var,
            onvalue=10,
            offvalue=5,
            command=check_func
        )
        check2.pack()

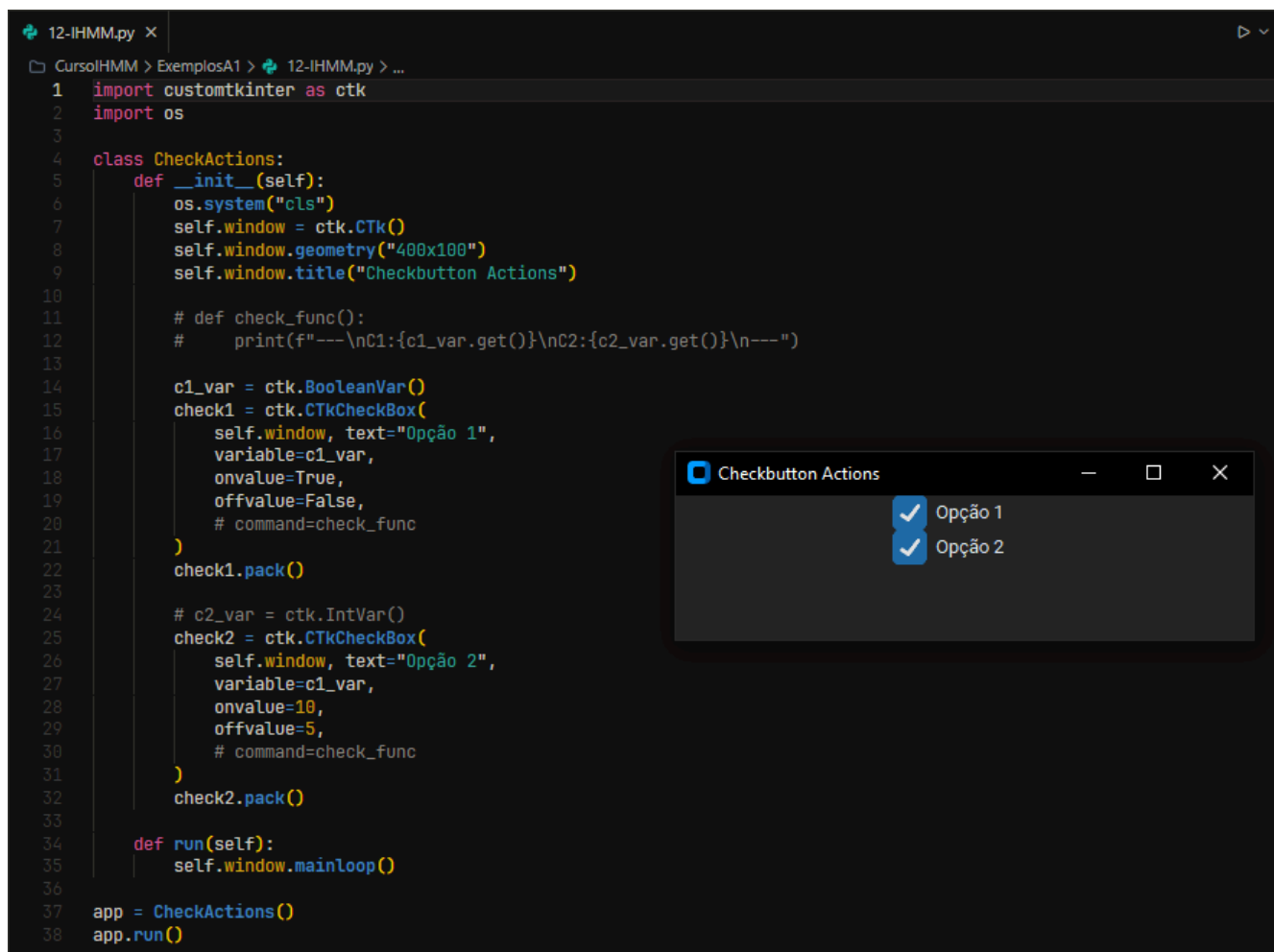
    def run(self):
        self.window.mainloop()

app = CheckActions()
app.run()
```



Nesse exemplo são criados dois checkbox, os estados de seleção do primeiro são `True` ou `False`. No segundo, os estados de seleção são números, 5 e 10. Abordagens como essa podem ser usadas para definir uma caixa de aceitação onde podemos verificar apenas se a caixa foi ou não foi selecionada, ou podemos fazer um sistema de pontos onde cada checkbox diferente possui uma pontuação que seria somada no final.

Uma mecânica interessante é que caso na interface tenha mais de um Checkbox e ambos estejam associadas a mesma variável de estado, quando qualquer das caixas de seleção for escolhida, ambos serão marcados automaticamente, eles irão funcionar como se fossem o mesmo widget.



Principais opções de configuração

Parâmetro	Descrição	Valor Padrão
<code>text</code>	Texto exibido ao lado da caixa de seleção.	<code>""</code> (string vazia)
<code>variable</code>	Variável vinculada ao estado da caixa (<code>BooleanVar</code> , <code>IntVar</code> , etc.).	<code>None</code>
<code>onvalue</code>	Valor definido para a variável quando a caixa está marcada.	<code>1</code>
<code>offvalue</code>	Valor definido para a variável quando a caixa está desmarcada.	<code>0</code>
<code>command</code>	Função ou método a ser executado quando o estado da caixa mudar.	<code>None</code>
<code>state</code>	Define o estado da caixa: ativa (<code>"normal"</code>) ou desativada (<code>"disabled"</code>).	<code>"normal"</code>
<code>fg_color</code>	Cor do preenchimento da caixa quando marcada.	Dependente do tema
<code>border_color</code>	Cor da borda da caixa.	Dependente do tema
<code>hover_color</code>	Cor ao passar o mouse sobre a caixa.	Automaticamente calculada
<code>corner_radius</code>	Define o raio de curvatura dos cantos da caixa de seleção.	<code>10</code>
<code>font</code>	Fonte do texto, aceita uma tupla como (<code>"NomeFonte", Tamanho, "Estilo"</code>).	<code>None</code>
<code>text_color</code>	Cor do texto ao lado da caixa de seleção.	Dependente do tema

RadioButton

Widget no Tkinter: `ttk.Radiobutton`

Widget no CustomTkinter: `ctk.CTkRadioButton`

Documentação: <https://customtkinter.tomschimansky.com/documentation/widgets/radiobutton>

Descrição

É um widget que representa botões de opção, usado para permitir ao usuário escolher apenas uma opção de um conjunto de opções mutuamente exclusivas.

O `Radiobutton` é ideal para cenários onde apenas uma escolha entre várias opções é permitida, como seleção de categorias, preferências ou modos de operação.

Como é usado

O `Radiobutton` funciona de uma maneira semelhante ao `CheckBox`, mas a diferença é que quando há mais de uma opção de itens de `Radiobutton`, o usuário pode escolher apenas uma das opções.

```
radio_button = ctk.CTkRadioButton(window, text="Opção 1")
radio_button.pack()
```

Com um `Radiobutton` também podemos armazenar seu valor em uma variável, e uma ação específica desse widget é que se uma variável for associada a mais de um `Radiobutton`, o Tkinter/CustomTkinter automaticamente faz com que seja escolhido apenas 1.

Se tivermos 2 RadioButtons alocados na janela, ambos com a mesma variável armazenando seus valores, se primeiramente o 1º for selecionado, tudo certo, mas quando o 2º for selecionado, o 1º é automaticamente desselecionado.

```
import customtkinter as ctk
import os

class RadioActions:
    def __init__(self):
        os.system("cls")
        self.window = ctk.CTk()
        self.window.geometry("400x150")
        self.window.title("Checkbox Actions")

    def radio_func():
        print(f"---\nGrupo 1:{g1_selected.get()}\n
              Grupo 2:{g2_selected.get()}\n---")

    # Frame para o Grupo 1
    frame_g1 = ctk.CTkFrame(self.window)
    frame_g1.pack(side="left", fill="both", expand=True, padx=10, pady=10)

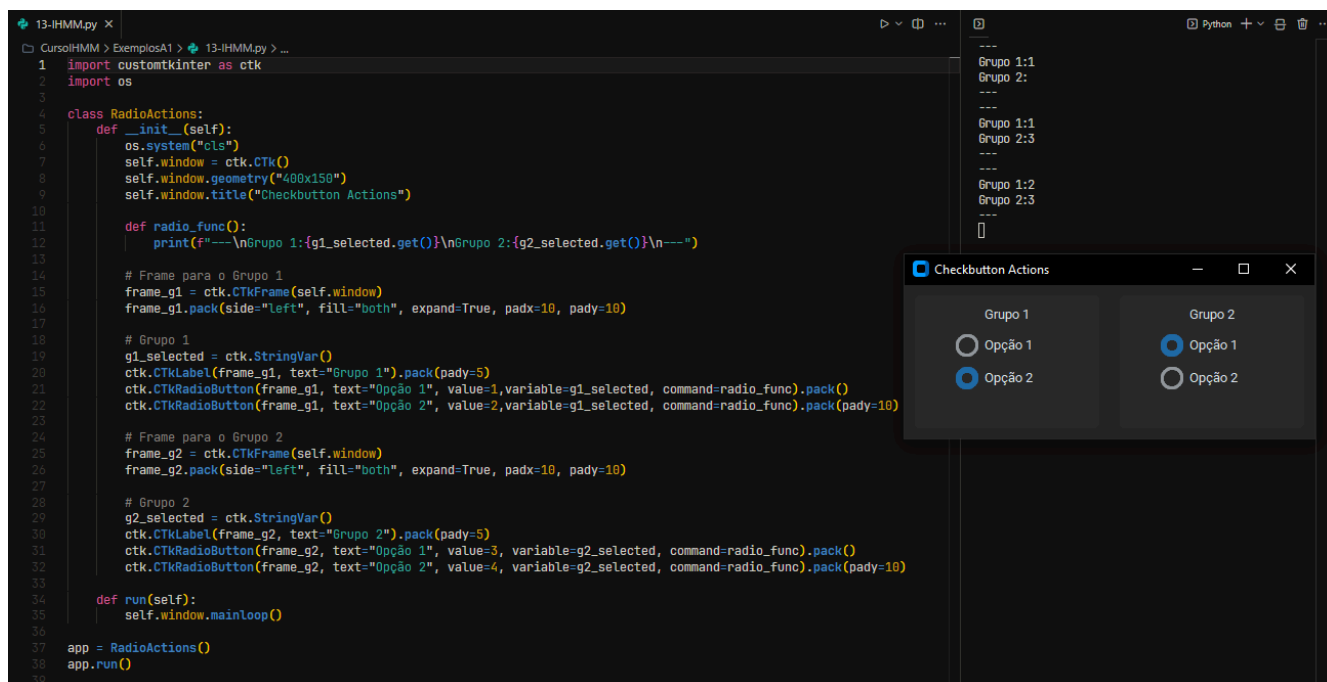
    # Grupo 1
    g1_selected = ctk.StringVar()
    ctk.CTkLabel(frame_g1, text="Grupo 1").pack(pady=5)
    ctk.CTkRadioButton(
        frame_g1, text="Opção 1", value=1,
        variable=g1_selected, command=radio_func
    ).pack()
    ctk.CTkRadioButton(
        frame_g1, text="Opção 2", value=2,
        variable=g1_selected, command=radio_func
    ).pack(pady=10)

    # Frame para o Grupo 2
    frame_g2 = ctk.CTkFrame(self.window)
    frame_g2.pack(side="left", fill="both", expand=True, padx=10, pady=10)

    # Grupo 2
    g2_selected = ctk.StringVar()
    ctk.CTkLabel(frame_g2, text="Grupo 2").pack(pady=5)
    ctk.CTkRadioButton(
        frame_g2, text="Opção 1", value=3,
        variable=g2_selected, command=radio_func
    ).pack()
    ctk.CTkRadioButton(
        frame_g2, text="Opção 2", value=4,
        variable=g2_selected, command=radio_func
    ).pack(pady=10)

    def run(self):
        self.window.mainloop()

app = RadioActions()
app.run()
```



Nesse exemplo foram criados dois grupos, cada um com dois widgets RadioButton. Os widgets de cada grupo foram associados a uma variável de estado sendo assim temos `g1_selected` e `g2_selected`. O comportamento mencionado anteriormente, de que só é possível selecionar um botão quando ambos tem a mesma variável de estado associada a ele, pode ser visto no exemplo.

Principais opções de configuração

Parâmetro	Descrição	Valor Padrão
<code>text</code>	Texto exibido ao lado do botão de rádio.	<code>""</code> (string vazia)
<code>variable</code>	Variável associada ao botão, compartilhada entre os botões do grupo para determinar a seleção.	<code>None</code>
<code>value</code>	Valor associado ao botão quando ele está selecionado.	<code>""</code> (string vazia)
<code>command</code>	Função ou método a ser executado quando o botão é selecionado.	<code>None</code>
<code>state</code>	Define o estado do botão: ativo (<code>"normal"</code>) ou desativado (<code>"disabled"</code>).	<code>"normal"</code>
<code>fg_color</code>	Cor do preenchimento do botão quando selecionado.	Dependente do tema
<code>hover_color</code>	Cor ao passar o mouse sobre o botão.	Automaticamente calculada
<code>border_color</code>	Cor da borda do botão.	Dependente do tema
<code>text_color</code>	Cor do texto ao lado do botão.	Dependente do tema
<code>font</code>	Fonte do texto, aceita uma tupla como <code>("NomeFonte", Tamanho, "Estilo")</code> .	<code>None</code>

Combobox

Widget no Tkinter: `ttk.Combobox`

Widget no CustomTkinter: `ctk.CTkComboBox`

Widget no CustomTkinter: `ctk.CTkOptionMenu`

Documentação ComboBox: <https://customtkinter.tomschimansky.com/documentation/widgets/combobox>

Documentação OptionMenu: <https://customtkinter.tomschimansky.com/documentation/widgets/optionmenu>

Descrição

O widget original Tkinter é chamado de `Combobox`, e o CustomTkinter tem dois widgets que representam o original, o `CTkComboBox` e o `CTkOptionMenu`.

Há uma diferença crucial entre eles, o `CTkComboBox` assim como o `ttk.Combobox` permitem o usuário digitar na entrada de texto onde as opções vão aparecer, mas essa funcionalidade chega a ser inútil ao projeto final. Já o `CTkOptionMenu` desabilita essa funcionalidade o que deixa o widget muito mais fácil de usar.

O `CTkOptionMenu` é um widget que permite ao usuário selecionar uma opção de uma lista suspensa de valores pré-definidos.

Ele é útil em situações onde é necessário limitar as entradas do usuário a um conjunto específico de opções, como escolher um tema, unidade de medida ou categoria

Como é usado

```
option_menu = ctk.CTkOptionMenu(window, values=["Opção 1", "Opção 2", "Opção 3"])
option_menu.pack()
```

Podemos armazenar o valor atualmente selecionado no OptionMenu usando uma variável de estado, assim como os outros widgets:

```
import customtkinter as ctk
import os

class ComboboxActions:
    def __init__(self):
        os.system("cls")
        self.window = ctk.CTk()
        self.window.geometry("400x150")
        self.window.title("Checkbutton Actions")

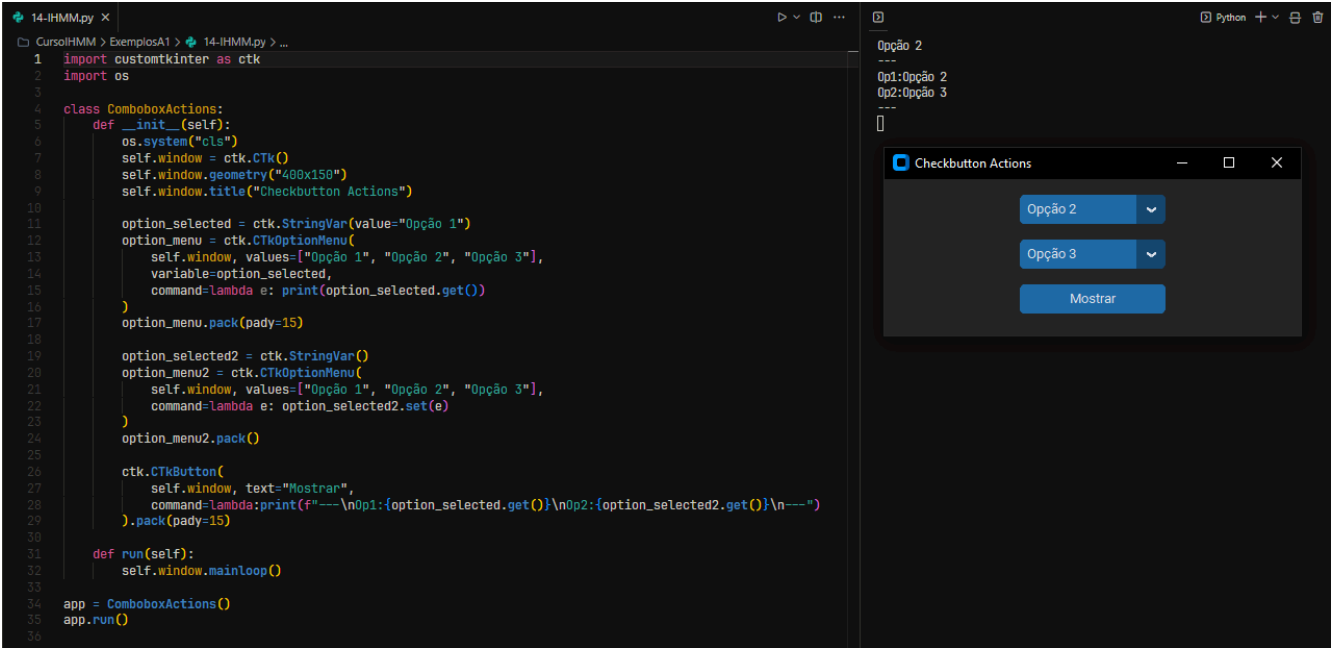
        option_selected = ctk.StringVar(value="Opção 1")
        option_menu = ctk.CTkOptionMenu(
            self.window, values=["Opção 1", "Opção 2", "Opção 3"],
            variable=option_selected,
            command=lambda e: print(option_selected.get())
        )
        option_menu.pack(pady=15)

        option_selected2 = ctk.StringVar()
        option_menu2 = ctk.CTkOptionMenu(
            self.window, values=["Opção 1", "Opção 2", "Opção 3"],
            command=lambda e: option_selected2.set(e)
        )
        option_menu2.pack()

        ctk.CTkButton(
            self.window, text="Mostrar",
            command=lambda: print(f"---\nOp1:{option_selected.get()}\n\nOp2:{option_selected2.get()}\n---")
        ).pack(pady=15)

    def run(self):
        self.window.mainloop()
```

```
app = ComboboxActions()
app.run()
```



Uma particularidade do OptionMenu é que ao passar no parâmetro `command` uma função lambda, ela espera um argumento, então precisamos criar lambda e declarar esse argumento, isso é feito colocando um `e` após o lambda. É preciso fazer isso mesmo que não seja usado o argumento `e`.

Isso é demonstrado no segundo item, o OptionMenu2 onde o valor da variável de estado associada a ele é alterado pela função lambda utilizando o argumento `e`, de `event`.

Principais opções de configuração

Parâmetro	Descrição	Valor Padrão
<code>values</code>	Lista de opções disponíveis no menu.	<code>[]</code> (vazio)
<code>command</code>	Função a ser chamada quando o usuário seleciona uma opção.	<code>None</code>
<code>variable</code>	Variável para sincronizar o valor selecionado.	<code>None</code>
<code>fg_color</code>	Cor de fundo do widget.	Dependente do tema
<code>button_color</code>	Cor do botão que abre o menu.	Dependente do tema
<code>button_hover_color</code>	Cor do botão ao passar o mouse sobre ele.	Dependente do tema
<code>text_color</code>	Cor do texto exibido no menu.	Dependente do tema
<code>dropdown_fg_color</code>	Cor de fundo do menu suspenso.	Dependente do tema
<code>dropdown_hover_color</code>	Cor do fundo das opções ao passar o mouse sobre elas.	Dependente do tema

Métodos Úteis:

Método	Descrição
<code>set(value)</code>	Define a opção selecionada no menu.
<code>get()</code>	Retorna o valor atualmente selecionado.
<code>configure(**kwargs)</code>	Permite alterar dinamicamente as configurações do widget.

Scale

Widget no Tkinter: `ttk.Scale`

Widget no CustomTkinter: `ctk.CTkSlider`

Documentação: <https://customtkinter.tomschimansky.com/documentation/widgets/slider>

Descrição

É um widget que oferece uma barra deslizante para entrada ou ajuste de valores numéricos dentro de um intervalo específico.

Útil em aplicações que exigem ajustes finos de configurações como volume, brilho ou limites numéricos

Como é usado

```
slider = ctk.CTkSlider(window from_=0, to=100)
slider.pack()
```

O Slider também pode ser associado a uma variável de estado. E podemos orientar o widget para que fique ou na horizontal ou na vertical.

```
import customtkinter as ctk
import os

class ScaleActions:
    def __init__(self):
        os.system("cls")
        self.window = ctk.CTk()
        self.window.geometry("400x330")
        self.window.title("Checkbutton Actions")

        frame3 = ctk.CTkFrame(self.window)
        frame3.pack(side="bottom", padx=5, pady=5, expand=True, fill="both")

        frame1 = ctk.CTkFrame(self.window)
        frame1.pack(side="left", padx=5, pady=5, expand=True, fill="both")

        frame2 = ctk.CTkFrame(self.window)
        frame2.pack(side="left", padx=5, pady=5, expand=True, fill="both")

        scale_value1 = ctk.IntVar(value=0)
        ctk.CTkLabel(frame1, textvariable=scale_value1).pack()
        scale = ctk.CTkSlider(frame1, from_=0, to=100, variable=scale_value1)
        scale.pack(padx=10, pady=10)

        scale_value2 = ctk.IntVar(value=0)
        scale2_label = ctk.CTkLabel(frame2, text="Scale 0")
        scale2_label.pack()
        scale2 = ctk.CTkSlider(
            frame2, from_=0, to=100, variable=scale_value2,
            command=lambda e: scale2_label.configure(
```

```

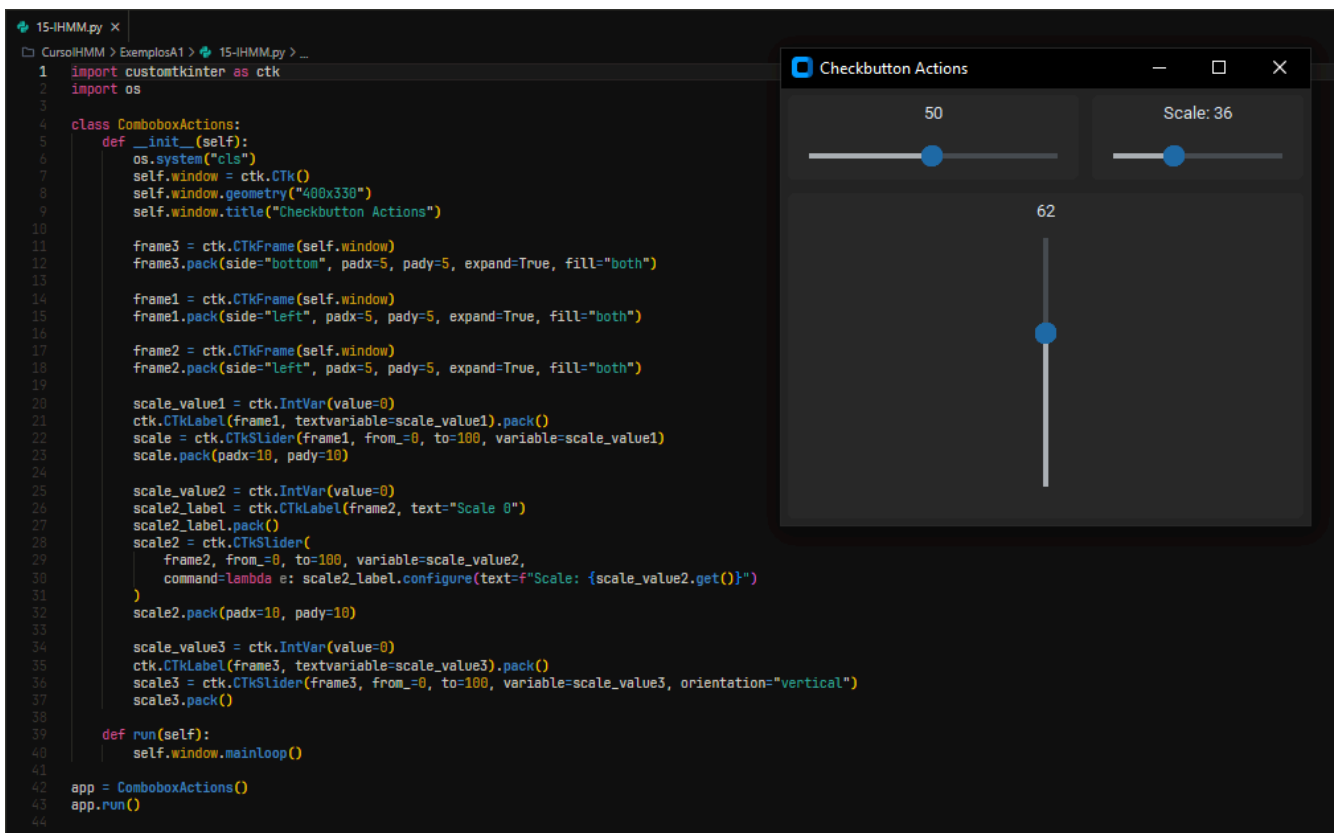
        text=f"Scale: {scale_value2.get()}"
    )
)
scale2.pack(padx=10, pady=10)

scale_value3 = ctk.IntVar(value=0)
ctk.CTkLabel(frame3, textvariable=scale_value3).pack()
scale3 = ctk.CTkSlider(
    frame3, from_=0, to=100, variable=scale_value3,
    orientation="vertical"
)
scale3.pack()

def run(self):
    self.window.mainloop()

app = ScaleActions()
app.run()

```



Nesse exemplo foi demonstrado algumas maneiras de usar o valor do Slider, e como mudar sua orientação.

Principais opções de configuração

Parâmetro	Descrição	Valor Padrão
<code>from_</code>	Valor mínimo permitido no slider.	0
<code>to</code>	Valor máximo permitido no slider.	1
<code>number_of_steps</code>	Número de etapas discretas entre <code>from_</code> e <code>to</code> .	None (valor contínuo)
<code>command</code>	Função chamada quando o valor do slider é alterado.	None
<code>orientation</code>	Orientação do slider ("horizontal" ou "vertical").	"horizontal"
<code>progress_color</code>	Cor da parte preenchida do slider.	Dependente do tema

Parâmetro	Descrição	Valor Padrão
<code>button_color</code>	Cor do botão do controle deslizante.	Dependente do tema
<code>button_hover_color</code>	Cor do botão ao passar o mouse sobre ele.	Dependente do tema

Métodos Úteis:

Método	Descrição
<code>get()</code>	Retorna o valor atual do slider.
<code>set(value)</code>	Define o valor do slider.
<code>configure(**kwargs)</code>	Permite alterar dinamicamente as configurações do widget.

Interface Final

Agora que já foi explicado a maioria dos widgets e suas funcionalidades, vamos criar uma interface simples, como finalização dessa aula. A interface que criaremos é uma simplificação da interface apresentada anteriormente usada para calcular o ganho do amplificador operacional não inversor.

Usaremos os conhecimentos de posicionamento de widgets, armazenamento dos valores das variáveis, botões que acionam funções e rótulos dinâmicos.

```
"""
- Interface final do minicurso -
Interface simplificada da Calculadora de
ganho do amplificador operacional não inversor
"""

import customtkinter as ctk

class AmpopCalc:
    def __init__(self):
        self.window = ctk.CTk()
        self.window.title("Calculadora AmpOp")
        self.window.geometry("400x350")
        self.window.resizable(False, False)

        self.r1 = ctk.DoubleVar()
        self.r2 = ctk.DoubleVar()
        self.ganho = ctk.StringVar()

        self._frame_superior()
        self._frame_inferior()

    def _frame_superior(self):
        frame = ctk.CTkFrame(self.window, height=80)
        frame.pack(
            side="top", expand=True, fill="both",
            pady=10, padx=10
        )

        ctk.CTkLabel(frame, text="Resistência R1:").pack(
            anchor="w", padx=10, pady=5
        )

        r1 = ctk.CTkEntry(frame, textvariable=self.r1)
        r1.pack(fill="x", padx=10, pady=5)

        ctk.CTkLabel(frame, text="Resistência R2:").pack(
```

```

        anchor="w", padx=10, pady=5
    )
    r2 = ctk.CTkEntry(frame, textvariable=self.r2)
    r2.pack(fill="x", padx=10, pady=5)

    def escrever_ganho():
        try:
            ganho = - self.r2.get() / self.r1.get() # Exemplo de cálculo
            self.ganho.set(f"Ganho: {ganho:.2f}")
        except ZeroDivisionError:
            self.ganho.set("Erro: Divisão por zero")
        except Exception as e:
            self.ganho.set(f"Erro: {e}")

    ctk.CTkButton(
        frame, text="Calcular Ganho",
        command=escrever_ganho
    ).pack(pady=10)

    def _frame_inferior(self):
        frame = ctk.CTkFrame(self.window, height=100)
        frame.pack(side="bottom", fill="both", pady=10, padx=10)

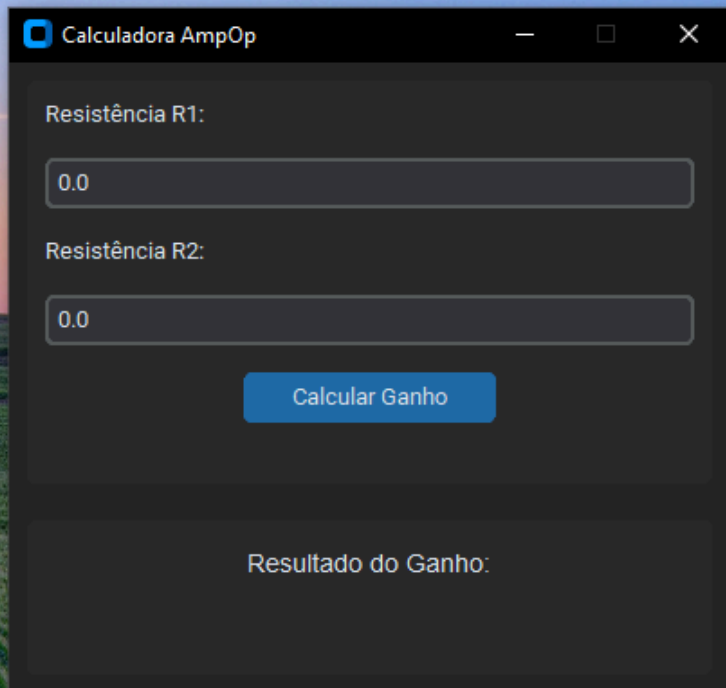
        ctk.CTkLabel(
            frame, text="Resultado do Ganho:",
            font=("Arial", 14)
        ).pack(pady=10)

        ctk.CTkLabel(
            frame, textvariable=self.ganho,
            font=("Arial", 16, "bold")
        ).pack(pady=5)

    def run(self):
        self.window.mainloop()

app = AmpopCalc()
app.run()

```



Calculadora AmpOp

Resistência R1:

0.0

Resistência R2:

0.0

Calcular Ganho

Resultado do Ganho: