Tkinter: Interfaces gráficas em Python

As interfaces gráficas do usuário (GUI – Graphic User Interface) são bastante populares no uso de softwares em geral, e os programadores devem estar aptos a trabalhar com a criação de interfaces, já que torna o uso mais fácil além de aumentar a produtividade.

Para quem trabalha com desenvolvimento em Python, existem diversos frameworks e ferramentas que permitem a criação interfaces gráficas.

A seguir podemos ver alguns frameworks gráficos que permitem desenvolver interfaces em Python:

- **WxWidgets;**
- **Tkinter;**
- **Kivy;**
- **PyGTK;**
- **PySide;**
- **QT.**

Estes frameworks listados são apenas alguns que podem ser usados para criar interfaces em Python. Ao longo deste artigo você verá como criar interfaces usando a biblioteca Tkinter.

O que é Tkinter?

Tkinter é uma biblioteca da linguagem Python que acompanha a instalação padrão e permite desenvolver interfaces gráficas. Isso significa que qualquer computador que tenha o interpretador Python instalado é capaz de criar interfaces gráficas usando o Tkinter, com exceção de algumas distribuições Linux, exigindo que seja feita o download do módulo separadamente

Por que usar Tkinter?

Um dos motivos de estarmos usando o Tkinter como exemplo é a sua facilidade de uso e recursos disponíveis. Outra vantagem é que é nativo da linguagem Python, tudo o que precisamos fazer é importá-lo no momento do uso, ou seja, estará sempre disponível.

Instalação e primeiro uso do Tkinter

Para que você possa trabalhar com Tkinter, é necessário que tenha pelo menos algum conhecimento em Python, então, não abordaremos a instalação padrão do interpretador e seus recursos.

A instalação padrão do Python acompanha além do interpretador, um ambiente de desenvolvimento, o IDLE, que está incluso no pacote Python na maioria das versões, com exceção de algumas distribuições Linux.

Na Figura 1 vemos o ambiente de desenvolvimento incluso no pacote Python, que por sinal, foi feito com o próprio Tkinter.

Durante todos os exemplos do artigo estaremos usando o IDLE na versão 3.5 do Python.

O Tkinter já vem por padrão na maioria das instalações Python, então tudo que temos que fazer é importar a biblioteca.

Para importar todo o conteúdo do módulo usamos o seguinte comando:

From Tkinter import *

Conceitos de GUI (Graphic User Interface)

Uma GUI aborda muitos conceitos, dos quais os mais comuns são:

Container – É uma analogia a um container físico e tem como objetivo organizar e guardar objetos. Da mesma forma este conceito serve para um container em interface. Nesse caso, os objetos que estamos armazenando são os widgets;

Widget – É um componente qualquer na tela, que pode ser um botão, um ícone, uma caixa de texto, etc.;

Event Handler – São tratadores de eventos. Por exemplo, ao clicarmos em um botão para executar uma ação, uma rotina é executada. Essa rotina é chamada de event handler;

Event Loop – O event loop verifica constantemente se outro evento foi acionado. Caso a hipótese seja verdadeira, ele irá executar a rotina correspondente.

Montando a interface

Vamos começar a montar a interface. Começaremos a escrever nosso primeiro código usando a Listagem 1.

```
class Application:
  def __init__(self, master=None):
    pass
```

```
root = Tk()

Application(root)

root.mainloop()
```

Listagem 1. Início da interface - Criando frame top level

Vamos entender o código: primeiro importamos todos os componentes do módulo Tkinter. Logo após, criamos uma classe chamada Application, que no momento ainda não possui nenhuma informação. É nela que criaremos os controles que serão exibidos na tela.

Depois instanciamos a classe TK() através da variável root, que foi criada no final do código. Essa classe permite que os widgets possam ser utilizados na aplicação.

Em Application(root) passamos a variável root como parâmetro do método construtor da classe Application. E para finalizar, chamamos o método root.mainloop() para exibirmos a tela. Sem o event loop, a interface não será exibida.

O código foi salvo em um arquivo .py (extensão Python) e depois executado. O resultado obtido é o mesmo da Figura 2.

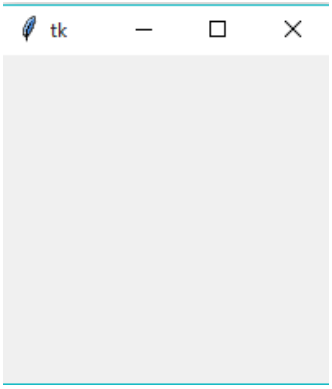
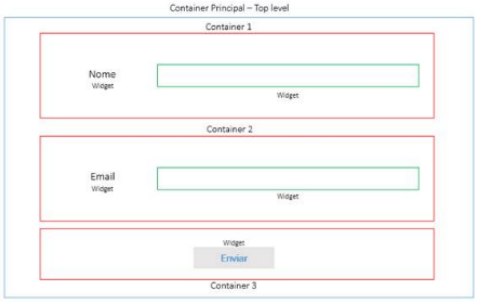


Figura 2. Primeiro contato com a GUI

Adicionando Widgets e montando a interface

Para poder trabalhar com widgets é necessário entender o conceito de container, que é uma estrutura onde os widgets são colocados. Por questão de organização e para sua correta criação, definimos os containeres, e dentro de cada container, um ou mais widgets que o compõe.

Com a Figura 3 entenderemos melhor os conceitos de container e widget.



Sempre que um container for criado dentro de outro, devemos, no momento de sua criação, definir qual é o container pai. A mesma questão de hierarquia serve para a criação de widgets, devendo ser definido na sua criação qual o container pai, ou seja, em que container ele será incluído.

Logo após definirmos a hierarquia de containeres e widgets, podemos posicionar os elementos na tela, indicando a posição em que o elemento irá aparecer.

O módulo Tkinter oferece três formas de trabalharmos com geometria e posicionamento:

- Pack;
- Grid;
- Place.

Neste artigo abordaremos o gerenciador de posicionamento Pack. Caso um widget não seja aplicado a nenhum gerenciador geométrico, ele continua existindo, mas invisível ao usuário.

Vamos criar nosso primeiro widget, como uma caixa de texto dentro do container principal, como mostra a Listagem 2.

```
from tkinter import *

class Application:

def __init__(self, master=None):

    self.widget1 = Frame(master)

    self.widget1.pack()

    self.msg = Label(self.widget1, text="Primeiro widget")

    self.msg.pack ()

root = Tk()

Application(root)

root.mainloop()
```

Listagem 2. Adicionando o primeiro widget à interface

O código resultará na interface da **Figura 4**.

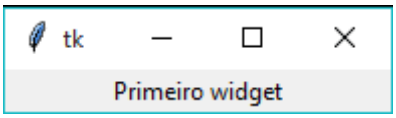


Figura 4. Criando o primeiro widget

Comparando com a primeira figura, quando não tínhamos definido nenhum widget, a tela diminuiu de tamanho. Isso aconteceu porque como não tinha nenhum conteúdo definido e a tela assumiu o tamanho padrão, mas a partir do momento que definimos um widget dentro da janela top-level (principal), a mesma assumiu o tamanho deste widget.

Veja na linha 4 do código que criamos o primeiro container chamado widget1 e passamos como referência o container pai.

O frame master é o nosso top level, que é o elemento máximo da hierarquia, ou seja, é a nossa janela de aplicação, que contém o título, e botões de maximizar, minimizar e fechar.

Na linha 5 informamos o gerenciador de geometria pack e usamos um widget label para imprimir na tela as palavras "Primeiro widget" e informamos que o container pai é o widget1, que foi passado como parâmetro, conforme a linha 6.

Por fim, exibimos na tela mais uma vez, usamos o gerenciador pack (linha 7).

Agora, vamos adicionar outros widgets à nossa interface, conforme mostra a Listagem 3.

```
class Application:
  def __init__(self, master=None):
    self.widget1 = Frame(master)
    self.widget1.pack()
    self.msg = Label(self.widget1, text="Primeiro widget")
    self.msg["font"] = ("Verdana", "10", "italic", "bold")
    self.msg.pack ()
    self.sair = Button(self.widget1)
    self.sair["text"] = "Sair"
    self.sair["font"] = ("Calibri", "10")
    self.sair["width"] = 5
    self.sair["command"] = self.widget1.quit
    self.sair.pack ()
root = Tk()
Application(root)
root.mainloop()
```

Listagem 3. Adicionando elemento do tipo button à interface

O código resultará na interface da **Figura 5**.

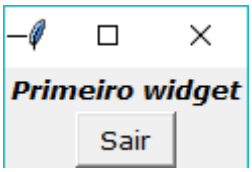


Figura 5. Adicionando um elemento button

Veja que adicionamos um widget do tipo Button e depois atribuímos os valores e estilização, como altura, largura e tipo da fonte a ser exibida.

Existem muitas outras configurações que podem ser usadas. Vamos ver mais algumas delas quando estivermos adicionando elementos à nossa interface.

Posicionando e atribuindo valores a elementos da tela

Quem já trabalha ou já trabalhou com **CSS** já está familiarizado com a maioria das configurações de estilo.

Vamos ver algumas configurações de estilo mais comuns que podemos definir:

- Width – Largura do widget;
- Height – Altura do widget;
- Text – Texto a ser exibido no widget;
- Font – Família da fonte do texto;
- Fg – Cor do texto do widget;
- Bg – Cor de fundo do widget;
- Side – Define em que lado o widget se posicionará (Left, Right, Top, Bottom).

Por padrão, o side vem definido como Top, então vamos usá-lo para ver o que acontece.

Vamos modificar o último código, conforme mostra a **Listagem 4**.

```
from tkinter import *
class Application:
  def __init__(self, master=None):
    self.widget1 = Frame(master)
    self.widget1.pack(side=RIGHT)
    self.msg = Label(self.widget1, text="Primeiro widget")
    self.msg["font"] = ("Verdana", "10", "italic", "bold")
    self.msg.pack ()
    self.sair = Button(self.widget1)
    self.sair["text"] = "Sair"
    self.sair["font"] = ("Verdana", "10")
    self.sair["width"] = 5
    self.sair["command"] = self.widget1.quit
    self.sair.pack (side=RIGHT)
root = Tk()
Application(root)
```

```
root.mainloop()
```

Listagem 4. Usando o side para definir a localização do widget

O resultado é o mesmo da **Figura 6**.

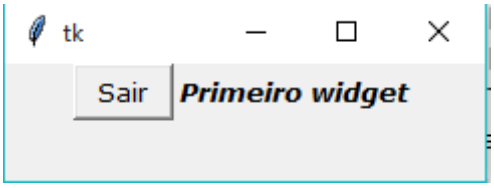


Figura 6. Posicionando widget à direita com o Side

Adicionamos a configuração pack(side=RIGHT) no elemento de texto e ele se alinhou à direita, colocando o restante do conteúdo na área restante.

Aplicando o event handler

Os event handlers são ações que executadas como resposta a determinado evento. Sempre que um evento ocorre, o event loop o interpreta como uma string. Por exemplo, ao clicar com o botão esquerdo do mouse, o event loop interpreta esta ação pela string "": o botão ENTER é representado pela string "" e o botão direito do mouse pela string "".

Vamos começar a usar esses conceitos. Para adicionar um evento de clique devemos primeiro criar o método event handler e fazer o binding no botão, conforme mostra a **Listagem 5**.

```
from tkinter import *
class Application:
  def __init__(self, master=None):
    self.widget1 = Frame(master)
    self.widget1.pack()
    self.msg = Label(self.widget1, text="Primeiro widget")
    self.msg["font"] = ("Calibri", "9", "italic")
    self.msg.pack ()
    self.sair = Button(self.widget1)
    self.sair["text"] = "Clique aqui"
    self.sair["font"] = ("Calibri", "9")
    self.sair["width"] = 10
    self.sair.bind("<Button-1>", self.mudarTexto)
    self.sair.pack ()

  def mudarTexto(self, event):
```

```
        if self.msg["text"] == "Primeiro widget":
            self.msg["text"] = "O botão recebeu um clique"
        else:
            self.msg["text"] = "Primeiro widget"
root = Tk()
Application(root)
root.mainloop()
```

Listagem 5. Usando binding e event handler

Quando executamos este código e o botão que foi criado recebe um clique, o texto é modificado na tela, graças ao event handler associado ao bind, como mostra a Figura 7.

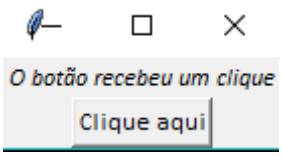


Figura 7. Adicionando um event handler

Perceba que quando criamos o método mudarTexto() adicionamos dois parâmetros: o self e o event. Nesse caso eles são obrigatórios para o funcionamento e devem ser sempre os dois primeiros parâmetros do método.

Outra opção que temos é passar diretamente o comando como atributo do widget, conforme mostra a **Listagem 6**.

```
from tkinter import *
class Application:
  def __init__(self, master=None):
    self.widget1 = Frame(master)
    self.widget1.pack()
    self.msg = Label(self.widget1, text="Primeiro widget")
    self.msg["font"] = ("Calibri", "9", "italic")
    self.msg.pack ()
    self.sair = Button(self.widget1)
    self.sair["text"] = "Clique aqui"
    self.sair["font"] = ("Calibri", "9")
    self.sair["width"] = 10
    self.sair["command"] = self.mudarTexto
```

```
        self.sair.pack ()


    def mudarTexto(self):
        if self.msg["text"] == "Primeiro widget":
            self.msg["text"] = "O botão recebeu um clique"
        else:
            self.msg["text"] = "Primeiro widget"
root = Tk()
Application(root)
root.mainloop()
```

Listagem 6. Utilizando o command

No código não foi preciso passar o argumento event como parâmetro do método, já que passamos o evento como atributo de um widget e assim ele estará sempre associado ao clique do mouse, não sendo necessário passar a string .

Recebendo dados do usuário

Para receber dados do usuário vamos usar o widget Entry, onde os mesmos são capturados como string, de forma semelhante ao método input.

Vamos criar uma entrada de dados simples utilizando o Entry, como mostra a Listagem 7.

```
from tkinter import *
class Application:
    def __init__(self, master=None):
        self.fontePadrao = ("Arial", "10")
        self.primeiroContainer = Frame(master)
        self.primeiroContainer["pady"] = 10
        self.primeiroContainer.pack()


        self.segundoContainer = Frame(master)
        self.segundoContainer["padx"] = 20
        self.segundoContainer.pack()


        self.terceiroContainer = Frame(master)
```

```
        self.terceiroContainer["padx"] = 20
        self.terceiroContainer.pack()

        self.quartoContainer = Frame(master)
        self.quartoContainer["pady"] = 20
        self.quartoContainer.pack()

        self.titulo = Label(self.primeiroContainer, text="Dados do usuário")
        self.titulo["font"] = ("Arial", "10", "bold")
        self.titulo.pack()

        self.nomeLabel = Label(self.segundoContainer,text="Nome",
font=self.fontePadrao)
        self.nomeLabel.pack(side=LEFT)

        self.nome = Entry(self.segundoContainer)
        self.nome["width"] = 30
        self.nome["font"] = self.fontePadrao
        self.nome.pack(side=LEFT)

        self.senhaLabel = Label(self.terceiroContainer, text="Senha",
font=self.fontePadrao)
        self.senhaLabel.pack(side=LEFT)

        self.senha = Entry(self.terceiroContainer)
        self.senha["width"] = 30
        self.senha["font"] = self.fontePadrao
        self.senha["show"] = "*"
        self.senha.pack(side=LEFT)

        self.autenticar = Button(self.quartoContainer)
        self.autenticar["text"] = "Autenticar"
```

```
        self.autenticar["font"] = ("Calibri", "8")
        self.autenticar["width"] = 12
        self.autenticar["command"] = self.verificaSenha
        self.autenticar.pack()

        self.mensagem = Label(self.quartoContainer, text="",
font=self.fontePadrao)
        self.mensagem.pack()

    #Método verificar senha
    def verificaSenha(self):
        usuario = self.nome.get()
        senha = self.senha.get()
        if usuario == "usuariodevmedia" and senha == "dev":
            self.mensagem["text"] = "Autenticado"
        else:
            self.mensagem["text"] = "Erro na autenticação"

root = Tk()
Application(root)
root.mainloop()
```

Listagem 7. Recebendo dados com o Entry

A execução do código resultará na Figura 8.

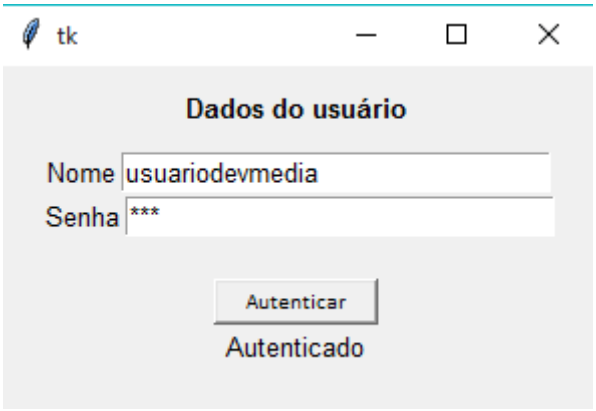


Figura 8. Utilizando o widget Entry para entrada de dados

Veja no código que tudo que precisamos fazer para receber a entrada foi usar o widget Entry e informar o container em que foi aplicado.

Usamos também o método get() para recuperar o texto que foi digitado pelo usuário e atribuímos as variáveis usuário e senha.