TKINTER

- Biblioteca que permite interação com o user através de uma interface gráfica (GUI).
- Interfaces são criadas por janelas, menus, ícones onde permitem a interação com rato e teclado.
- A programação baseada em ambiente GUI é feita através de recurso a programação guiada por eventos (PGE).
- Desta forma, terá que existir um ciclo de inicialização que aguarda ocorrência de eventos.
- De cada vez que o evento ocorre, é encaminhado para a parte do programa que responde a esse evento.

TKINTER

- Biblioteca gráfica nativa do Python. Python faz uso desta biblioteca. Não é desenvolvida em Python (Tcl/Tk)
- Multiplataforma (Linux, Microsoft, …)
- from tkinter import * ##na versão 2 a biblioteca era chamada Tkinter

- Este módulo funciona com base em princípios básicos:
  - Definir uma janela;
  - Nessa janela são colocados um conjunto de elementos (widgets deriva da expressão Windows gadget)

TKINTER

- Objetos (tudo é um objeto)
  - Atributos (características dos objetos, como a cor, a largura, o ícone, …)
    - Métodos e eventos (permitem alterar os atributos)

Os métodos são ações que podem decorrer da intervenção do utilizador como redimensionar uma janela, minimizá-la, fazer clique num botão, …)

Eventos são o resultado da ação dos métodos. Podem ser ativados pela ação do utilizador como por ação da própria app, como por exemplo, ter na app um relógio que se atualiza de segundo ao segundo…

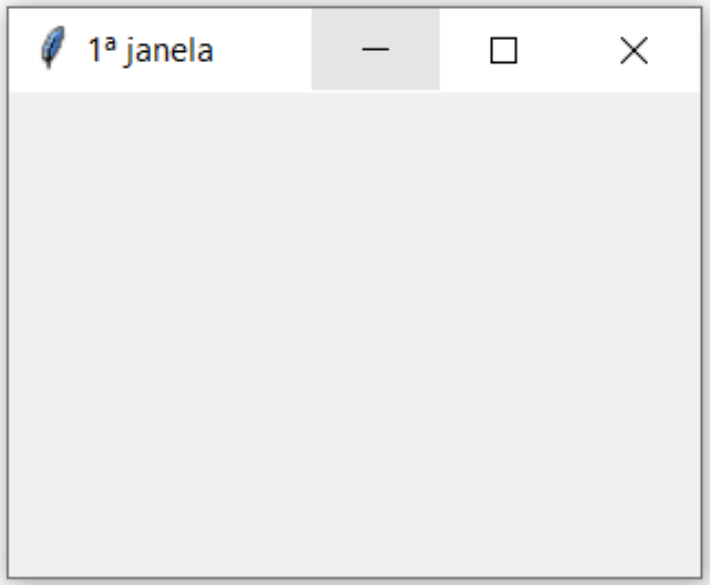
Criar janela

Para criar uma simples janela basta invocar uma instância Tk()

```
from tkinter import *
janela = Tk()

#invocar o ciclo de gestão de eventos
janela.mainloop()  #permite ficar à espera que ocorra qualquer evento
```

Janela com título e dimensão

```
from tkinter import *

janela=Tk()
##mudar título da janela
janela.title('1ª janela')

#definir dimensões da janela
janela.geometry('1250x1175')

janela.mainloop()
```



Neste exemplo, após executar podemos constatar que abre uma janela, com o título «1ª janela» com a dimensão 250x175 px

Redimensionar janelas

- resizable(largura, altura)
  - Os valores do método resizable são booleanos
  - Exemplo:

    janela.resizable(True, False)

Nota: em vez dos valores True e False podemos usar os números 1 e 0 respetivamente: janela.resizable(1, 0)

Redimensionar entre valores mínimos e máximos

- Podemos definir mínimos e máximos da resolução da janela

minsize(largura, altura)

```
janela.minsize(width = 800, height = 600)
```

maxsize(largura, altura)

```
janela.maxsize(1024, 768)
```

Os valores podem ser dados simplesmente com os números ou então utilizar as propriedades width e height

Arranque da janela Maximizada/Minimizada

- Podemos arrancar a nossa janela maximizada ou ainda minimizada.
- Para tal, devemos utilizar o método state(tipo)

- Exemplo:
  - janela.state("zoomed") #maximizado
  - janela.state("iconic") #minimizado

Janela – cor de fundo e posição fixa

```
from tkinter import *
janela=Tk()
##mudar título da janela
janela.title('1ª janela')
#definir dimensões da janela em pixeis
janela.geometry('1250x1175+200+200') ##larg x alt + Pos_x + Pos_y
##definir background
janela["bg"] = "black"  ##podemos usar a chave "bg" ou "background"
janela.mainloop()
```

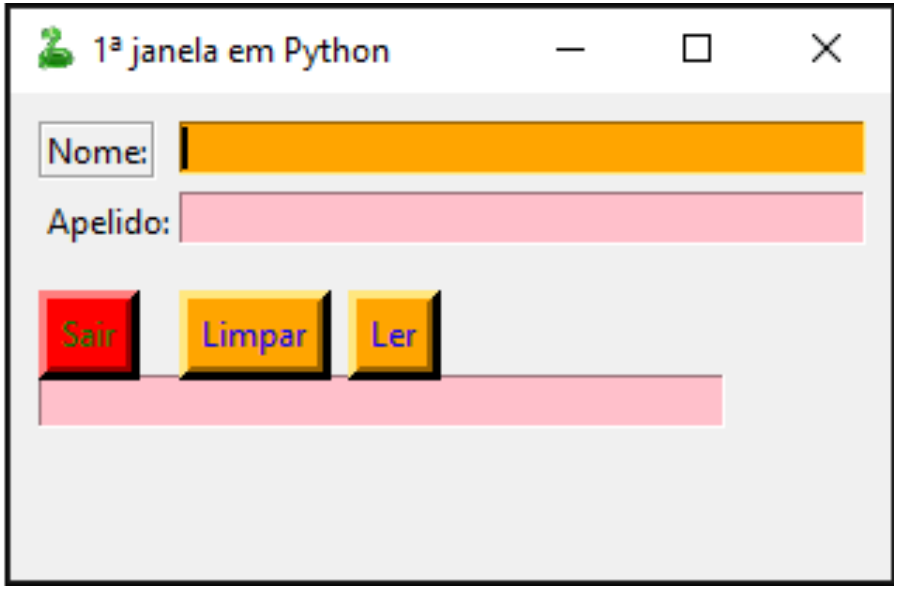
Janela – alteração do ícone

- janela.iconbitmap("nome_ficheiro.ico")
  - O ficheiro deverá ter o formato .ico
  - Se indicarmos apenas o nome do ficheiro então este deverá estar na mesma localização do script, caso contrário, deveremos indicar o caminho relativo para ele.

Centrar uma janela no monitor

```
janela = Tk()
largecra = janela.winfo_screenwidth() #obtém largura do ecrã
altecra = janela.winfo_screenheight() #obtém altura do ecrã
## centro = largecra/2 x altecra/2
larg = 800   #larg/2
alt = 500    #alt/2

#(x,y) pixel canto superior da janela (origem do objeto gráfico)
posx = largecra/2 - larg/2 #à metade da largura do monitor subtrai metade largura da janela
posy = altecra/2 - alt/2    #à metade da altura do monitor subtrai metade altura da janela

janela.geometry("largxalt+posx+posy") #dimensão em pixéis
#janela.geometry("%dx%d+%d+%d" % (larg,alt,posx,posy))
```

Objetos simples



- Rótulos de texto (Label)
  - label1 = Label(janela, text="Nome:", relief=GROOVE)
- Caixas de texto (Entry)
  - entry1 = Entry(janela, width=40,bg="orange",fg="Blue")
- Botões (Button)
  - blimpa = Button(janela, text="Limpar", command=limpar, bg="orange", fg="blue", bd=5)

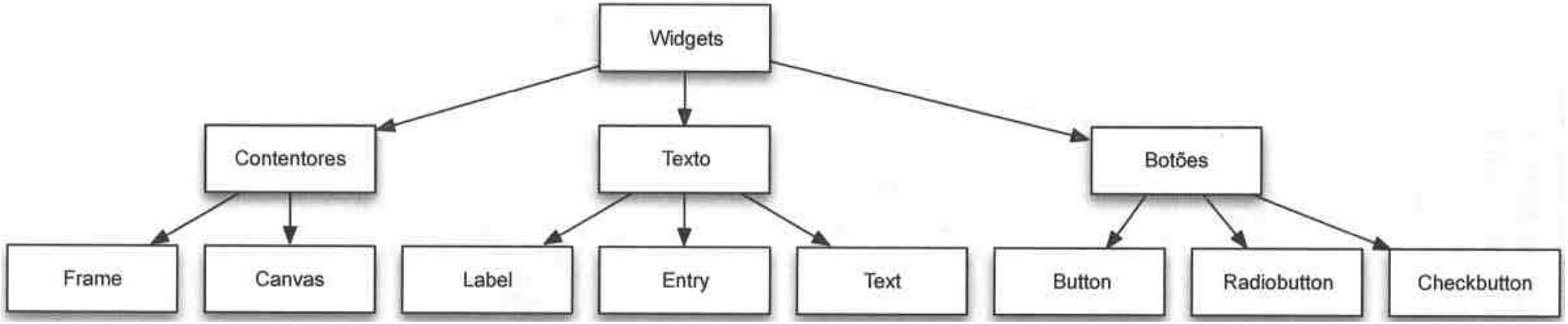
Colocação dos objetos

- Pack(opções)
  - expand - Quando definido como True, a janela expande-se para preencher qualquer espaço.
  - fill - determina se a janela preenche qualquer espaço: NONE (padrão), X (preencher apenas horizontalmente), Y (preencher apenas verticalmente) ou BOTH (preencher horizontalmente e verticalmente)
  - side - Determina qual lado a ser posicionado: “top” (padrão), “bottom”, “left” ou “right”.

Colocação dos objetos

- Grid(linha, coluna)
  - Funciona como uma tabela. Ajusta a dimensão das colunas e linhas
- Place(pos_x, pos_y)
  - O objeto é posicionado nas coordenadas especificadas. Pode sobrepor objetos ocultando outros.

Componentes de uma janela



Frame

- Frame funciona como um contentor que irá conter objetos.
- Primeiro passo, será criar uma Frame na janela que criamos anteriormente:

```
from tkinter import *
janela=Tk()
janela.title('1ª janela')
janela.geometry('250x175')

tela=Frame(janela)  #cria a frame
tela.grid() #o método grid() posiciona a frame na janela

janela.mainloop()
```

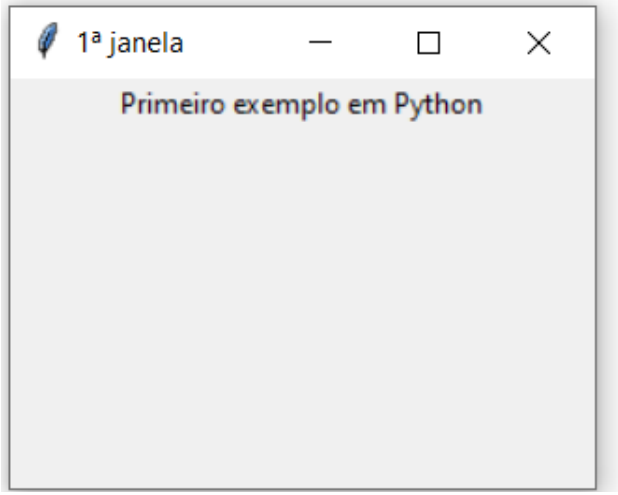
Frame

- Se pretendemos utilizar Frames para conter outros objetos, então esta deve ser criada em primeiro lugar, em seguida criar os objetos e posicioná-los na Frame e por fim posicionar a Frame:

```
from tkinter import *
janela=Tk()
frame1 = Frame(janela) # frame1 é filho de janela
#criar widgets
lnome = Label(frame1, text = "Nome: ")
enome = Entry(frame1)
#depois posicionamos objetos na frame
lnome.grid(row = 0,column = 0, stick=W) #stick ou sticky "encosta" o objeto à esquerda (Oeste)
enome.grid(row = 0,column = 1, stick=W)
#posicionar a Frame
frame1.grid()
janela.mainloop()
```

Label

- Uma label funciona como uma etiqueta ou rótulo. Assim, ao associar uma label ao contentor Frame devemos indicar qual o texto que pretendemos.

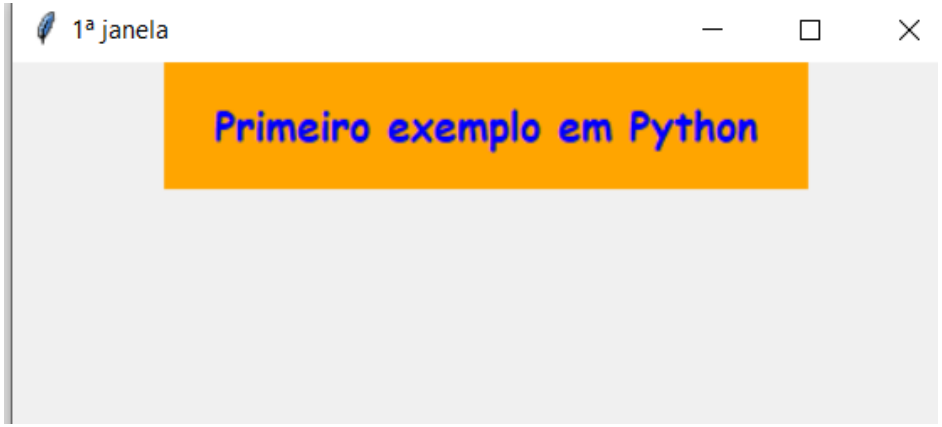



```
from tkinter import *

janela=Tk()

tela=Frame(janela)
tela.pack()

etiqueta=Label(tela, text="Primeiro exemplo em Python")
etiqueta.pack()

janela.mainloop()
```

Label - fontes

- Podemos definir o tipo de letra, a fonte e cores de background (bg) foreground(fg), altura e largura,.... Utilizamos o método configure()

```
etiqueta=Label(tela, text="Primeiro exemplo em Python")
fonte=("Comic Sans MS",14,"bold")
etiqueta.configure(font=fonte, width=25, height=2, bg="orange")
etiqueta.configure(fg="Blue")
etiqueta.pack()
```



Label – redimensionamento e relevo

- Se redimensionarmos a janela a nossa label está no estado imutável, ou seja, não se altera, não acompanha a "redimensão" da janela.
- Para o fazermos temos de recorrer às opções expand(YES ou NO) e fill(X,Y ou BOTH – expande em função do eixo do X, Y ou ambos).

etiqueta.pack(expand = YES, fill=BOTH)

- O relevo (relief) numa label pode ser dos seguintes tipo:
  - flat, solid, sunken, ridge, raised, groove

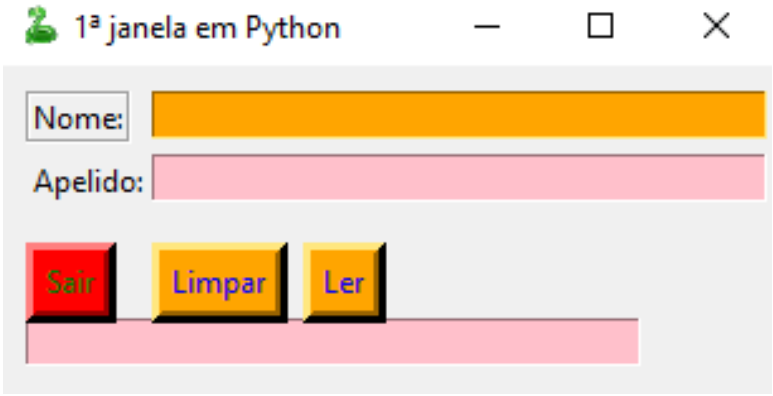
Label – redefinir as propriedades

- Por vezes, depois de termos a label apresentada graficamente, temos necessidade de redefinir as suas propriedades.
- O objeto Label, tem como representação interna o tipo "dictionary", onde o tipo de propriedade é a "key" e o valor é o "value"
- Assim, por exemplo, para alterar o texto de uma label:
  - label1["text"] = "Novo texto"
  - label1["relief"] = "sunken"

Entry – caixa de texto

- Caixa de texto que permite a inserção de dados por parte do utilizador.
- Podemos aplicar diversos tipos de configurações, tais como, o comprimento da caixa de texto, a cor de fundo e cor de texto ou ainda configurar uma fonte específica

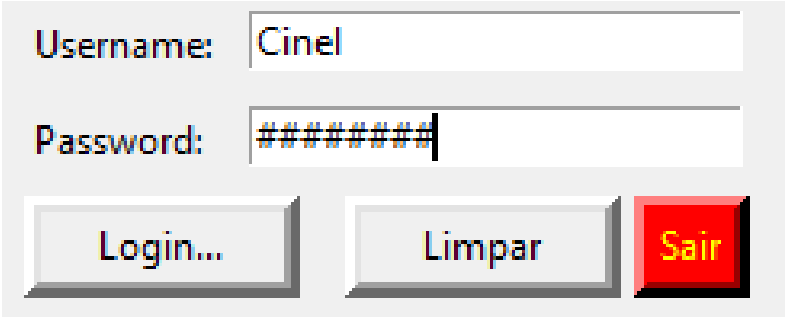
```
etexto=Entry(janela, width=40,bg="pink",fg="Blue")
lnome = Label(janela, text="Nome:", relief=GROOVE)
enome = Entry(janela, width=40,bg="orange",fg="Blue")
lapelido=Label(janela, text="Apelido:")
eapelido = Entry(janela, width=40,bg="pink",fg="Blue")
```



Entry – escondendo os carateres

- Por vezes, no caso de uma caixa de texto que represente uma password, não queremos que os carateres dessa password sejam mostrados à medida que o utilizador a escreve.
- Assim, devemos recorrer à key "show" indicando o carater que pretendemos utilizar para mascarar o carater real

```
lUser = Label(janela, text = "Username:")
eUser = Entry(janela, width=25)
lSenha = Label(janela, text = "Password:")
eSenha = Entry(janela, width=25, show="#")
bLer = Button(janela,text="Login...",bd=5,width=10)
bLimpar =Button(janela,text="Limpar",bd=5,width=10)
bSair = Button(janela,text="Sair",bd=5,fg='Yellow',bg='Red')
```



Entry – leitura e escrita

- Para obter os dados escritos numa ENTRY podemos recorrer ao método GET
  - nome = entry1.get()
- Quando pretendemos inserir algo numa ENTRY, utilizamos o método INSERT, indicando a partir de que posição pretendemos inserir os texto e qual o valor a inserir
  - entry1.insert(0,"texto")
- Para apagar o conteúdo de uma ENTRY podemos utilizar o método DELETE, indicando o início e o fim das posições
  - entry1.delete(0, END)

Estado de uma Entry

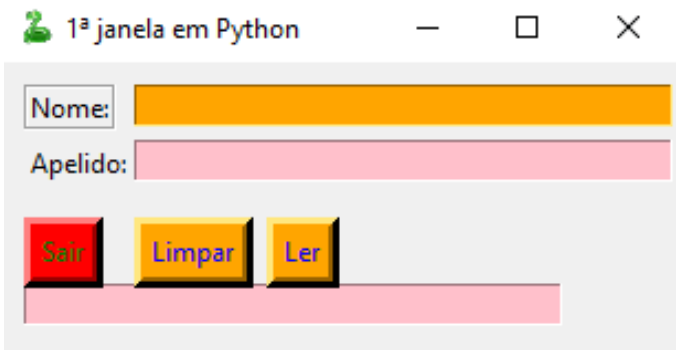
- Uma ENTRY poderá apresentar 1 dos 3 estados seguintes:
  - normal (Editável)
  - disabled (Não editável. Se pretende editar terá que mudar para estado normal)
  - readonly (Não editável. Se pretende editar terá que mudar para estado normal. Neste estado consegue-se ler o conteúdo)

Button

- O objeto Button permite-nos desencadear uma série de ações através do recurso a funções em python, graças à sua key "command"

```
bsair = Button(janela, text="Sair", command=abandono,
                    bg="red",fg="Green", bd=5)
```

```
def abandono():
    janela.destroy()
```



Radiobutton

- Este widget implementa um botão de escolha de uma única opção entre várias disponíveis.
- Para implementar esta funcionalidade, cada grupo de botões de opção deve estar associado à mesma variável e cada um dos botões deve simbolizar um único valor. Pode-se usar a tecla "Tab" para alternar de um botão radio para outro.

valor = IntVar() #variável dinâmica

r1 = Radiobutton(janela, text="Portugal", variable=**valor**,

value=**1**) #"value" irá ser atribuído à variável "valor"

Radiobutton - exemplo

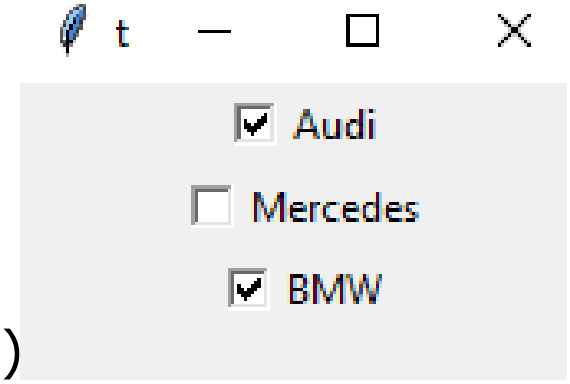

```
# Variável que define o grupo - Inteiro dinâmico
escolhacor=IntVar()
r1=Radiobutton(janela, text="Amarelo", variable = escolhacor,
               value = 0, command = cor) #value tem que ser nº pois definimos IntVar()
r2=Radiobutton(janela, text="Azul", variable = escolhacor,
               value = 1, command = cor)
r3=Radiobutton(janela, text="Preto", variable = escolhacor,
               value = 2, command = cor)
r4=Radiobutton(janela, text="Branco", variable = escolhacor,
               value = 3, command = cor)
```

Radiobutton - exemplo

```
def cor():
    cores=["Amarelo", "Azul", "Preto", "Branco"]
    x = escolhacor.get() #obtém o valor atribuído à variável dinâmica

    print(f'VALOR do RADIO = {x}')
    print(cores[x])
```

Checkbutton

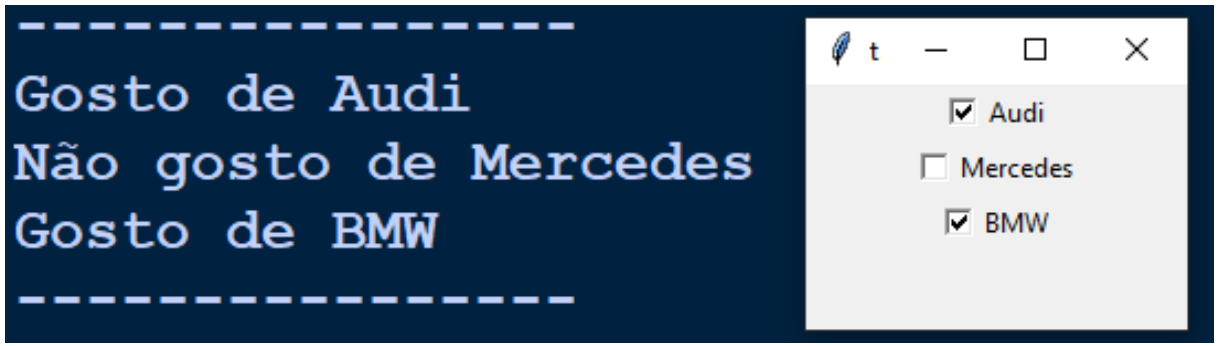
- Este widget implementa um botão de múltipla escolha de várias opções disponíveis.
- Para implementar esta funcionalidade, cada botão de opção deve estar associado a uma única variável. Pode-se usar a tecla "Tab" para alternar de um botão check para outro.




```
vcheck1 = IntVar() #variável dinâmica
vcheck2 = IntVar() #variável dinâmica
chk1 = Checkbutton(janela, text="Audi", variable=vcheck1)
chk2 = Checkbutton(janela, text="Mercedes", variable=vcheck2)
```

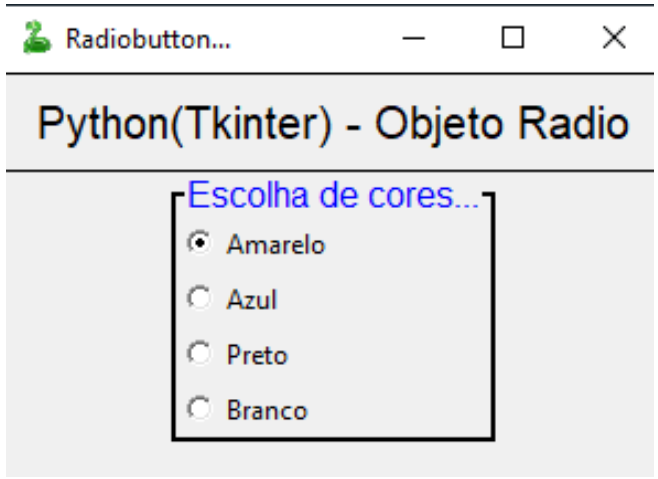
Checkbutton - exemplo

```
def escolha():
    ##Valores da checkbox é 1(ativo) ou 0(inativo)
    marca1=vcheck1.get()
    marca2=vcheck2.get()
    if(marca1):
        print("Gosto de Audi")
    else:
        print("Não gosto de Audi")
    if(marca2):
        print("Gosto de Mercedes")
    else:
        print("Não gosto de Mercedes")
```



Caixa de separador (LabelFrame)

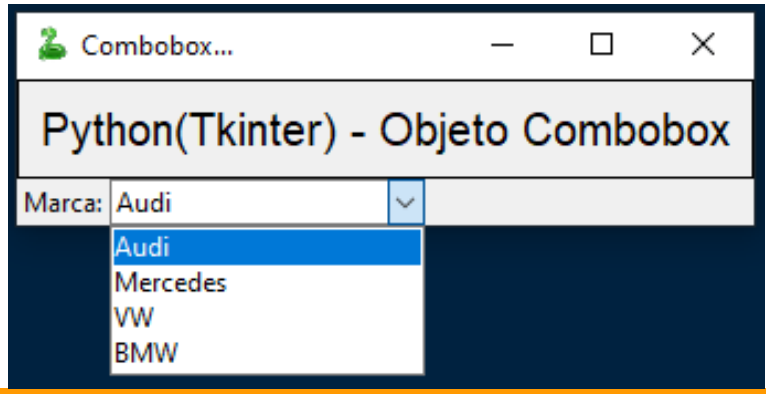
```
lbcores = LabelFrame(janela, text="Escolha de cores...", borderwidth=2,
                             relief="solid", fg="Blue",font="Arial 12").pack()
escolhacor=IntVar()
r1=Radiobutton(lbcores, text="Amarelo", variable = escolhacor, value = 0)
r2=Radiobutton(lbcores, text="Azul", variable = escolhacor, value = 1)
r3=Radiobutton(lbcores, text="Preto", variable = escolhacor, value = 2)
r4=Radiobutton(lbcores, text="Branco", variable = escolhacor, value = 3)
r1.pack(anchor=W)
r2.pack(anchor=W)
r3.pack(anchor=W)
r4.pack(anchor=W)
```



Combobox – caixa de combinação

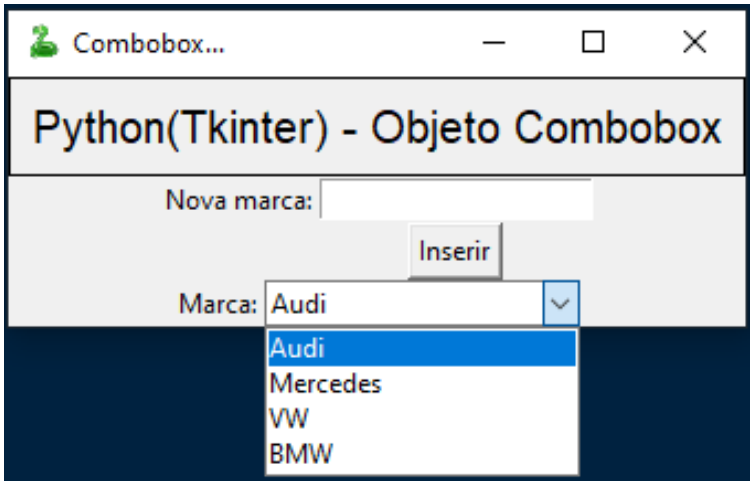
- Este objeto para ser carregado teremos que forçar a biblioteca ttk
  - from tkinter import ttk #from tkinter.ttk import *
- Existe a key "values" que será uma lista de valores a apresentar na caixa de combinação.

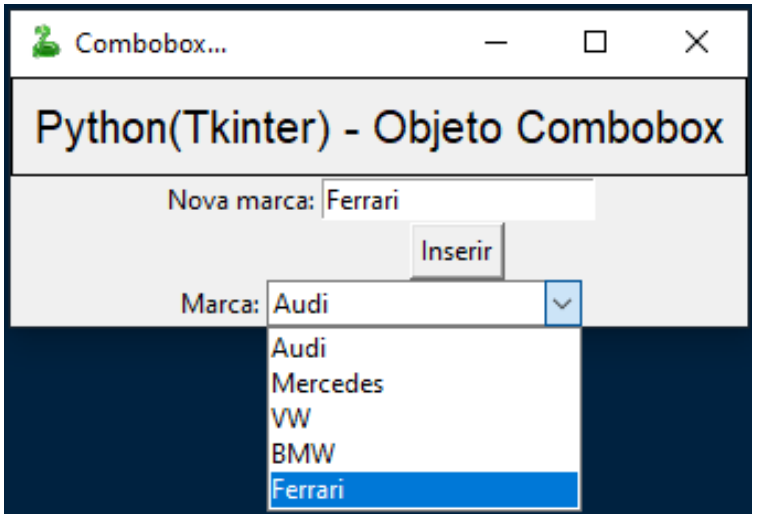
```
lmarcas=['Audi','Mercedes','VW','BMW'] #criar lista com valores
combo1 = ttk.Combobox(janela, values=lmarcas)
combo1.set("Audi") #preenche a caixa de combinação com o valor
                   #caso contrário a combobox estaria vazia
```



Combobox – adicionar elementos

- Ao optar por inserir um novo elemento numa combobox, teremos de fazer um "refresh", ou seja, reconfigurar a key "values"





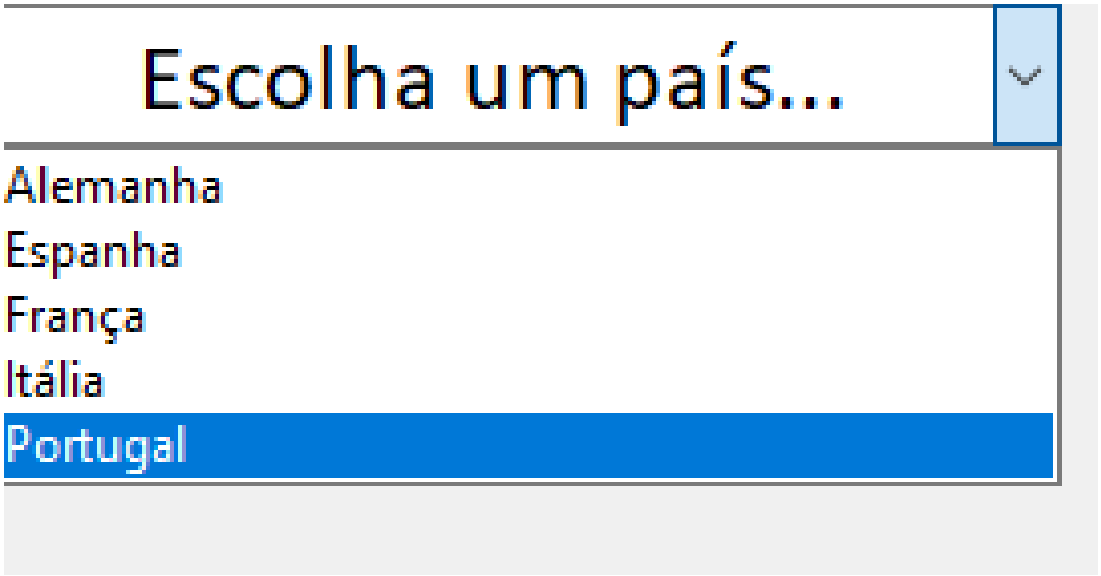

```
lmarcas=['Audi','Mercedes','VW','BMW']

def inserir():
    novo = entry1.get()
    lmarcas.append(novo)
    combo1.configure(values=lmarcas)
```

Combobox - estados

- Disable – Não é possível sequer escolher nova opção
- Readonly – Não é possível acrescentar texto apesar de deixar escolher opções

combo1[“justify”]=“CENTER”

combo1["state"]="readonly"



Combobox – reagir à mudança

- Por vezes poderá existir a necessidade de executar qualquer evento quando o valor da combobox é alterado.
- Teremos de utilizar ao método BIND:

```
nome_combo.bind("<<ComboboxSelected>>", funcao)
```

```
def funcao(evento):
    novo_valor = nome_combo.get()
    print(f"O valor da combobox foi alterado para: {novo_valor}")
```

Listbox

- O objetivo deste widget é permitir criação de uma lista de vários elementos. Podemos inserir elemento 1 de cada vez ou até mesmo inserir um conjunto de vários tópicos para a lista através de um ciclo (loop).

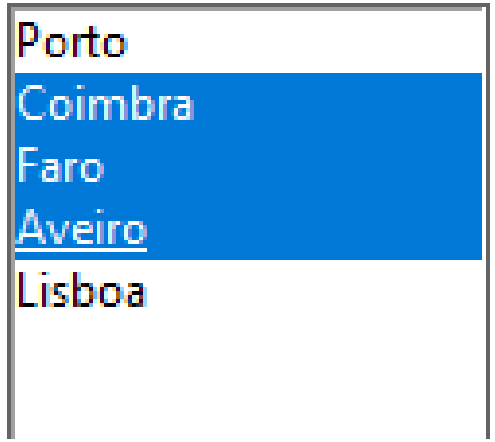
```
lb1 = Listbox(janela).pack()
lista = ['Batata','Cenoura','Repolho']
for x in lista:
    lb1.insert(END, x) #posiciona no fim da lista o elemento "x"
```

Listbox – várias seleções

Numa listbox é possível selecionar vários items.

Para tal, teremos que indicar a propriedade EXTENDED na key "selectmode"

```
lstbox1 = Listbox(janela, selectmode = EXTENDED)
```



Mais opções: https://www.tutorialspoint.com/python/tk_listbox.htm

Listbox - remover elementos

A opção curselection() devolve um tuplo com os índices selecionados.

```
def remover():
    pos = lb1.curselection() #tuplo com índices
    lb1.delete(pos[0], pos[-1])
```

pos[0] representa o início da seleção

pos[-1] representa o fim da seleção

Messagebox

- from tkinter.messagebox import *
- Biblioteca para criar caixas de mensagens de tipos diferenciados:

Aviso: showwarning(title="titulo", message="Aviso")

Erro:showerror(title="titulo", message="Erro")

Informação: showinfo(title="titulo", message="Informação... dramática")

Questionar: temos 5 tipos distintos para obter respostas do tipo Sim/Não/Cancelar

Messagebox

Questionar:

askquestion(title="questionar", message="questionar?")

askokcancel(title="okcancelar", message="ok ou cancelar")

askyesno(title="simnao", message="sim ou nao")

askyesnocancel(title="simnaocancelar", message="sim ou nao ou cancelar")

askretrycancel(title="Retry..." , message="Tentar novamente")

Bind

- Este método permite reagir a um determinado evento/ação.
- Por exemplo, para bloquear a inserção de um carater numa dada entry, podemos recorrer à ação "break" e não deixa inserir na entry:

```
cxtexto = Entry(janela, width=15)
cxtexto.bind("<Key>", lambda evento: "break")
```

Link para opções do BIND: https://www.python-course.eu/tkinter_events_binds.php

Nova janela sem dependência

- Por exemplo, quando um botão chama uma nova janela:

```
def novajan():
        janela2 = Tk()
        janela2.geometry("250x250")
        botao2 = Button(janela2, text="OFF", command=sair)
        botao2.pack()
        janela2.focus_force() #ATIVA O FOCUS NA JANELA 2
        janela.attributes('-disabled',True) #COLOCA 1ª JANELA DISABLED
        janela2.mainloop()
```

Nova janela sem dependência

- Por exemplo, quando um botão chama uma nova janela:

```
def novajan():
    def sair():
        janela.attributes('-disabled',False) #DESATIVA O DISABLE DA 1ª JANELA
        janela.focus_force() #FORÇA FOCUS NA 1ª JANELA
        janela2.destroy() #DESTROI A 2ª JANELA
    janela2 = Tk()
        ....
    janela2.mainloop()
```

Toplevel

- Permite chamar outras janelas a partir da janela principal. Por exemplo, quando um botão chama uma nova janela

```
janela = Tk()
b1 = Button(janela, text=">>>", command=proximo).pack()


def proximo():
    def atras():
        janela.deiconify()
        j2.destroy()
    j2 = Toplevel(janela)
    janela.iconify()
    bant = Button(j2, text="Voltar…", command=atras).pack()
```

Toplevel – ocultar/mostrar janelas

- Quando invocamos outra janela, por vezes torna-se necessário ocultar a janela anterior.

  janela.iconify()

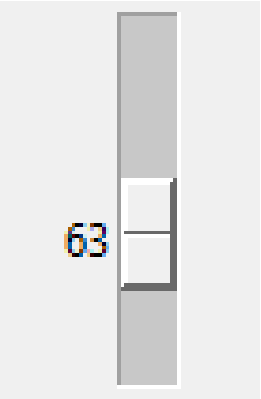
- Para voltar a mostrar a janela escondida:

  janela.deiconify()

Scale

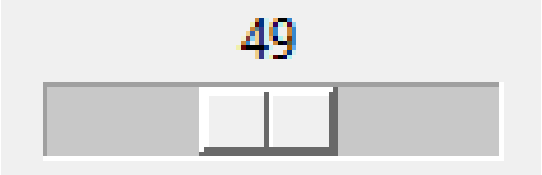
- escala1 = Scale(janela, from_=0, to=100)

varia os valores entre 0 e 100 com disposição vertical



- escala2 = Scale(janela, from_=0, to=100, orient=HORIZONTAL)

varia os valores entre 0 e 100 com disposição horizontal



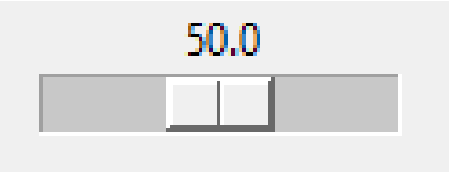
from_ para diferenciar do from reservado

Quando se importa bibliotecas

Scale – leitura, atribuição e evolução

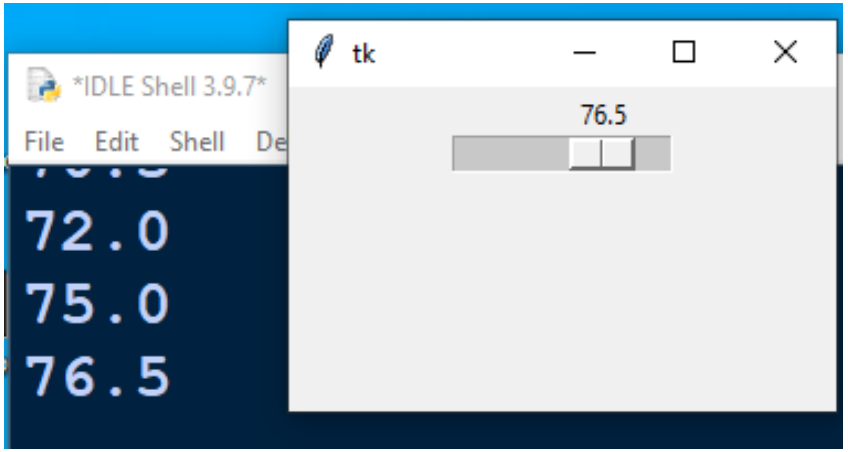
- escala1 = Scale(janela, from_=0, to=100, resolution=0.5, command=ler)

varia os valores entre 0 e 100 com disposição vertical e evolui de 0.5 em 0.5 e passa o valor para a função "ler"



- escala.set(50) # atribui o valor 50 à escala
- x = escala.get() # permite obter o valor definido na escala
- A função "ler" irá receber um valor automático, enviado pelo valor da própria escala

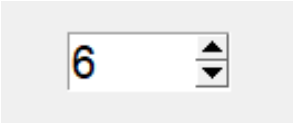
```
def ler(valor):
    x=escala.get()
    print(x)
```



Spinbox

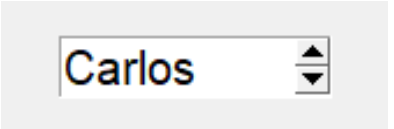
- Entry com botões de incremento e decremento. Permite validar os campos a mostrar à semelhança da combobox

```
spin1 = Spinbox(janela, from_=0, to=10,
                font=fonte, width=5)
```



- Spinbox aceita também uma lista de valores do tipo string:

```
spin2 = Spinbox(janela, font=font,
                values=('Ana','Bruno','Carlos','David'),
                wrap=True) #permite dar a volta à coleção
```



- Key WRAP permite rodar os valores da lista em ciclo

Spinbox

- A leitura de um valor do Spinbox consiste em utilizar o método get():
  - nome = spin2.get()

- É também possível mostrar valores dando um certo número para incremento:

```
spin1 = Spinbox(janela, from_=0, to=10,
                font=fonte, width=5, increment=2)
```

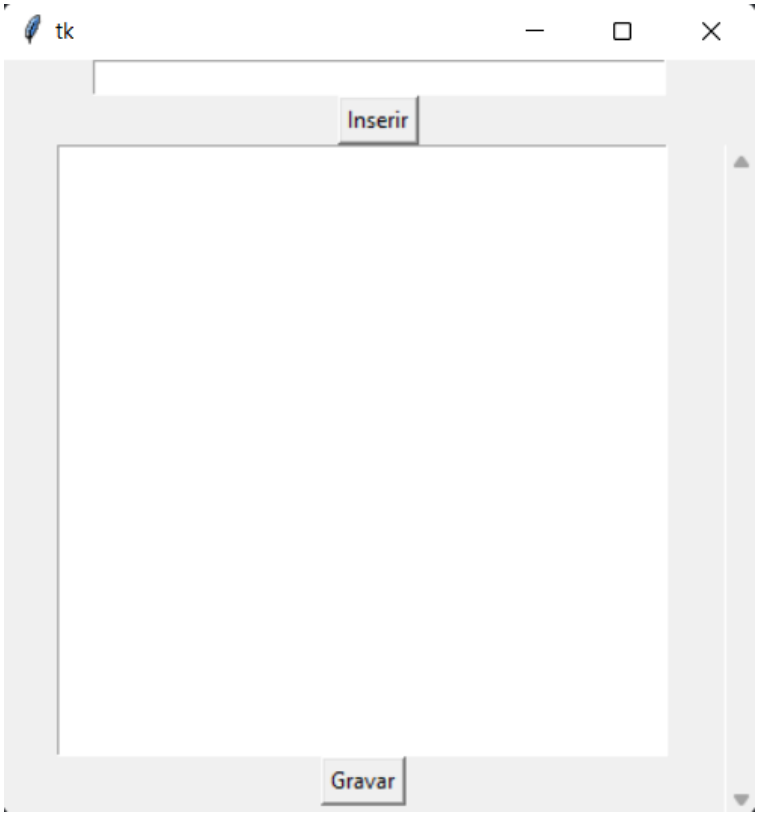
Assim, os valores a serem mostrados na Spinbox serão de 2 em 2

Text

Áreas de texto fornecem mecanismos que permitem editar um texto de várias linhas e formatá-lo da forma que pretendemos. Apresenta uma certa semelhança ao bloco de notas (notepad).

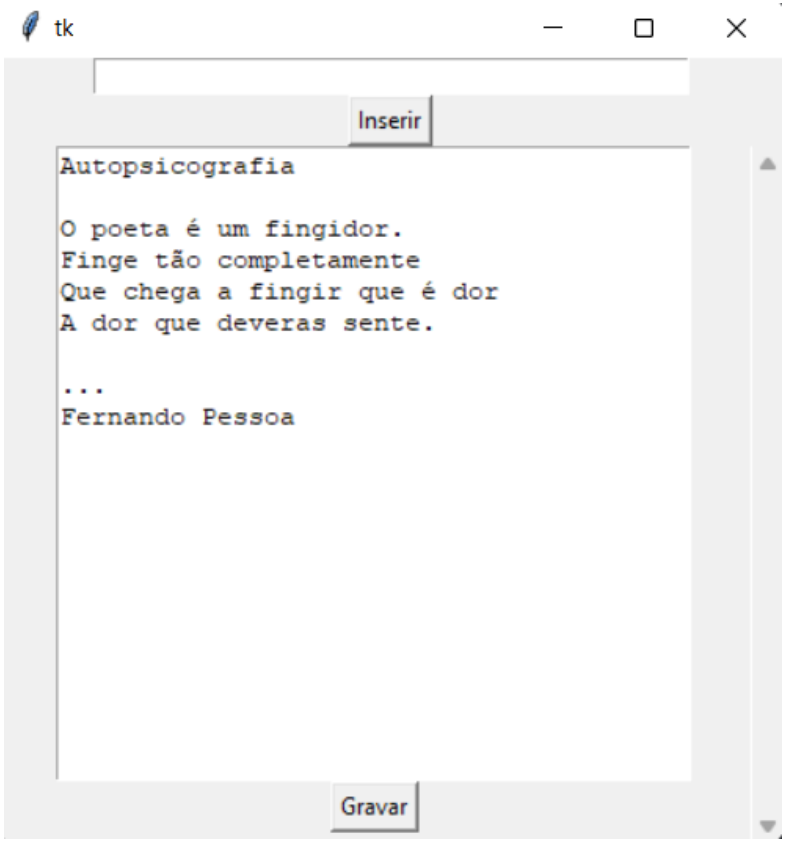
Pode-se também usar inserção de imagens.

https://www.tutorialspoint.com/python3/tk_text.htm



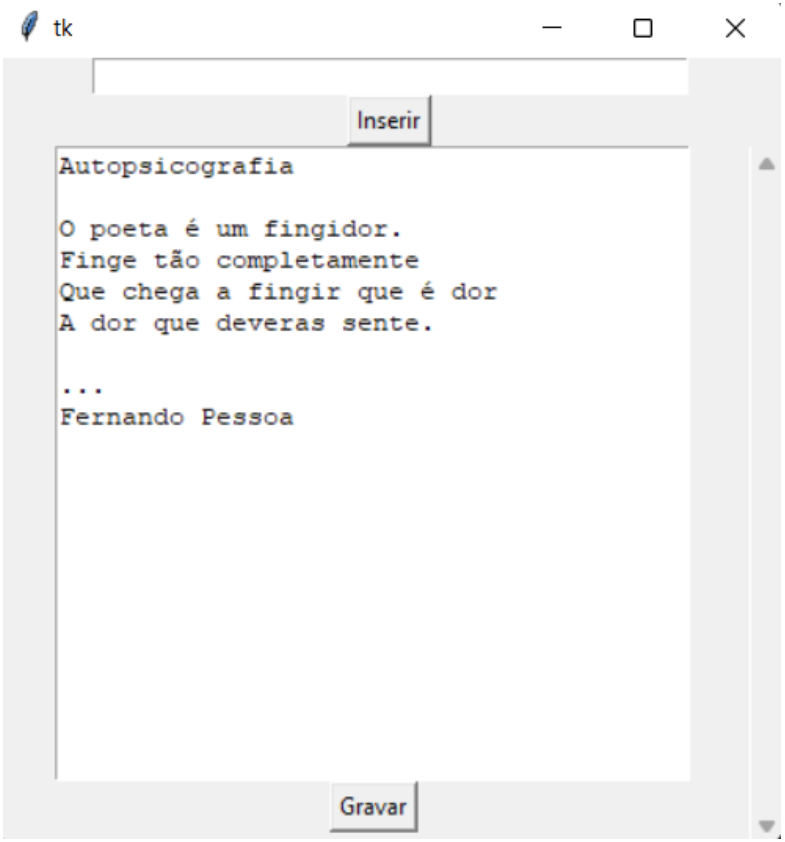
Text - escrita

Para inserir texto utilizaremos o método Insert. Para adicionar conteúdo no final da Text, utilizamos a expressão END.




```
def inseretexto():
    resultado = "Autopsicografia\n\nO poeta é um
fingidor.\nFinge tão completamente\nQue chega a fingir que é
dor\nA dor que deveras sente.\n\n...\nFernando Pessoa"
    areatexto.insert(END, resultado+'\n')
```

Text - leitura

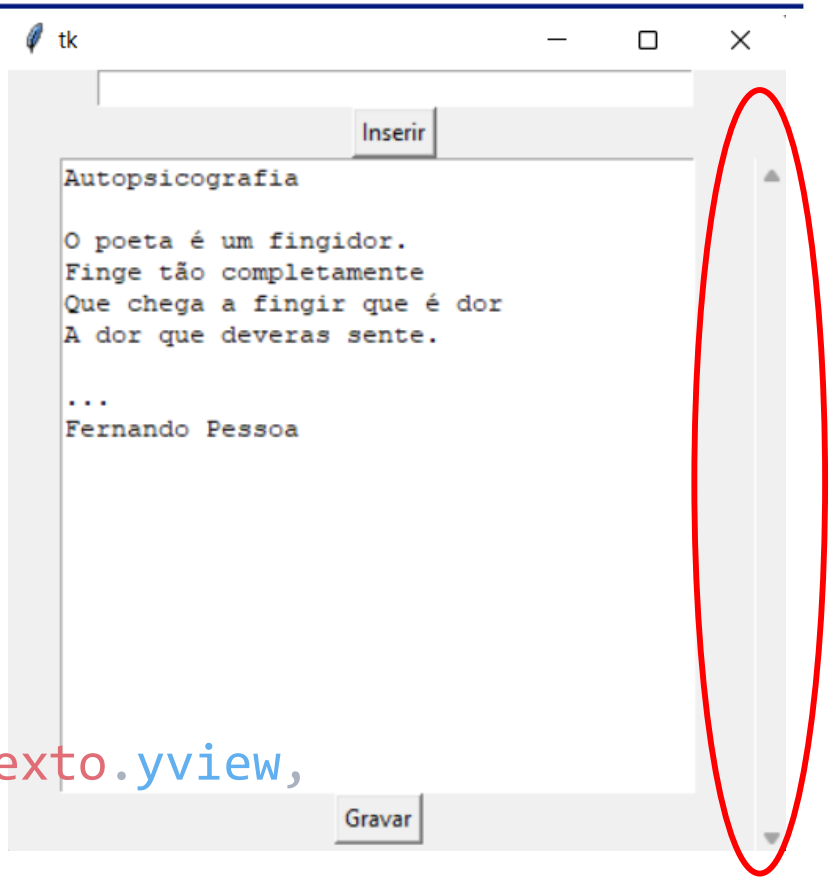


Na Text, para obter o conteúdo, utilizaremos o método get mas teremos que indicar qual a linha e a coluna do cursor (1ª linha – índice 1 e coluna índice 0)

```
def gravar():
    fich = open("lertexto.txt", "w")
    texto = areatexto.get(1.0,END) #lê da linha 1 e carater zero ate ao fim
    fich.write(texto)
    areatexto.delete(1.0,END)
    fich.close()
```

Scroll

Supondo que temos o objeto areatexto criado, pretendemos associar uma scrollbar vertical.



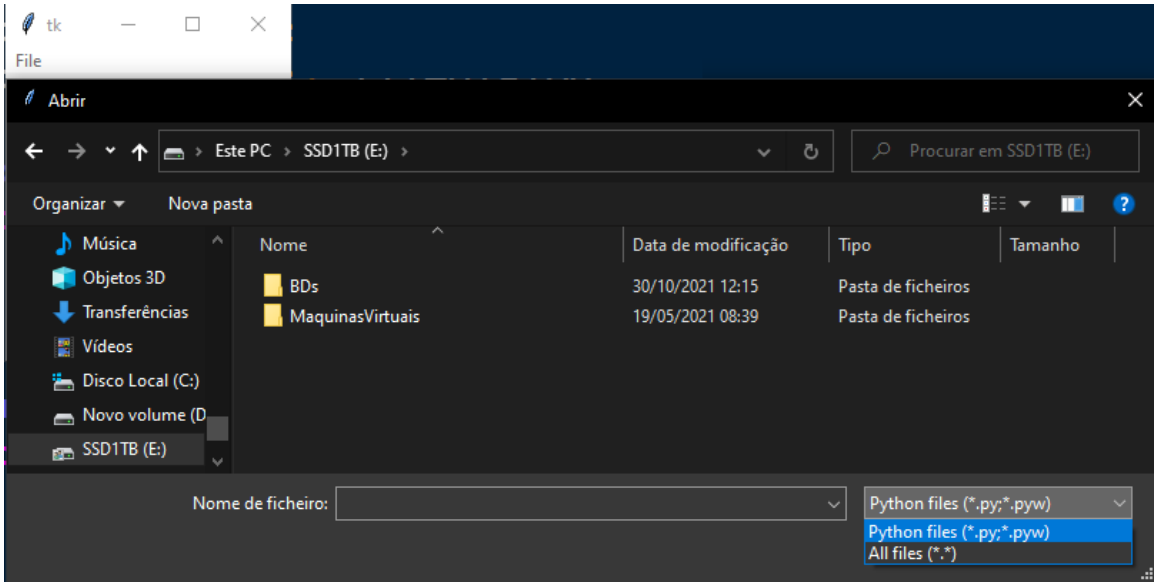

```
#Criação do objeto scrollbar:
scroll = Scrollbar(janela, command=areatexto.yview,
orient="vertical")
areatexto.configure(yscrollcommand=scroll.set) #associa scrollbar
ao scroll da área de texto
scroll.pack(side=RIGHT, fill=Y) #estica scroll ao tamanho da área
de texto
```

Menu

https://www.tutorialspoint.com/python3/tk_menu.htm

```
janela = Tk()
menubar = Menu(janela) #na janela cria área para conter Menu
#constituição dos items do menu
filemenu = Menu(menubar) #tearoff=0 -> prende o menu à janela
filemenu.add_command(label="Abrir", command=open)
filemenu.add_command(label="Guardar", command=save)
filemenu.add_command(label="Sair", command=exit)

menubar.add_cascade(label="Ficheiro", menu=filemenu) #agrupa items
do menu ao "Ficheiro"
janela.config(menu=menubar) #dispor a área do menu na janela
```

Funções do Menu




```
from tkinter import filedialog

def open():
    filedialog.askopenfilename(initialdir = "/", title = 'Abrir ficheiro',
            filetypes = (("Python files","*.py;*.pyw"),("All files","*.*")))
def save():
    filedialog.asksaveasfilename(initialdir = "/",title = 'Guardar como',
            filetypes = (("Python files","*.py;*.pyw"),("All files","*.*")))
```

Menu – Filedialog (3 funções base)

Função	Parâmetros	Descrição
.askopenfilename	Diretório, Título, Extensão	Para **abrir** ficheiro, a caixa de diálogo solicita ao Sistema Operativo (SO) a janela de seleção de um ficheiro existente.
.asksaveasfilename	Diretório, Título, Extensão	Para **guardar** ficheiro, a caixa de diálogo solicita ao SO a criação ou substituição de um ficheiro.
.askdirectory	Nenhum	Para **abrir diretório**

Imagens

- Biblioteca PIL (https://www.tutorialspoint.com/how-do-i-use-pil-with-tkinter)
- Sem biblioteca….

```
imagem = PhotoImage(file = 'c:/imagens/foto1.png')
botao = Button(janela, image=imagem)
```