

# **Python: Interfaces Gráficas com Tk**

Claudio Esperança

# Interfaces Gráficas

- Também chamadas de Graphical User Interfaces (GUI)
- Usadas em aplicações modernas que requerem uma interação constante com o usuário
  - Maior usabilidade e naturalidade do que interfaces textuais
- Aplicação apresenta uma ou mais janelas com elementos gráficos que servem para comandar ações, especificar parâmetros, desenhar e exibir gráficos, etc
- Bibliotecas (*toolkits*) para construção de interfaces como
  - Qt
  - Gtk
  - wxWindows
  - Tk

# Interfaces Gráficas em Python

- Python possui camadas de portabilidade (*bindings*) para várias bibliotecas de construção de interfaces. Ex.:
  - PyQt (Qt)
  - PyGtk (Gtk)
  - wxPython (wxWindows)
  - Tkinter (Tk)
- Multiplataforma (MS-Windows, Unix/Linux, OSX)

# Tk

- Toolkit originalmente criado para utilização com a linguagem script Tcl
- Bastante leve, portátil e robusto
- Um tanto obsoleto frente a outros toolkits mais modernos como Qt ou Gtk
- Camada Tkinter normalmente distribuída com o Python
  - Inicia um processo Tcl que toma conta dos elementos de interface
  - Classes e funções do Tkinter se comunicam com o interpretador Tcl para especificar aspecto e comportamento da interface

# Usando Tkinter

- Importar o módulo Tkinter
  - `from Tkinter import *`
- Elementos de interface (*widgets*) correspondem a objetos de diversas classes. Por exemplo:
  - Frame (Área retangular)
  - Button (botão)
  - Label (rótulo)
  - Text (caixa de texto)
  - Canvas (caixa de desenho)
- Posição e tamanho dos elementos controlados por gerentes de geometria
  - Pack (mais comum), Place, Grid

# Usando Tkinter (2)

- Para criar um widget, tem-se que informar o widget-pai (parâmetro *master*) onde geometricamente deverá ser encaixado e as opções de configuração para o widget. Ex.:  

```
w = Button(pai, text="Cancelar", command=cancelar)
```
- Tk já define por default uma janela principal
  - `master=None` (default) indica que o widget será filho da janela principal
  - Outras janelas pode ser criadas criando objetos da classe `Toplevel`
- A função `mainloop` tem que ser invocada para que a aplicação entre no modo de tratamento de eventos

# Exemplo

```
from Tkinter import *

class Application(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.msg = Label(self, text="Hello World")
        self.msg.pack ()
        self.bye = Button (self, text="Bye", command=self.quit)
        self.bye.pack ()
        self.pack()

app = Application()
mainloop()
```

# Exemplo

```
from Tkinter import *
```

```
class Application(Frame):
```

```
    def __init__(self, master=None):
```

```
        Frame.__init__(self, master)
```

```
        self.msg = Label(self, text="Hello World")
```

```
        self.msg.pack ()
```

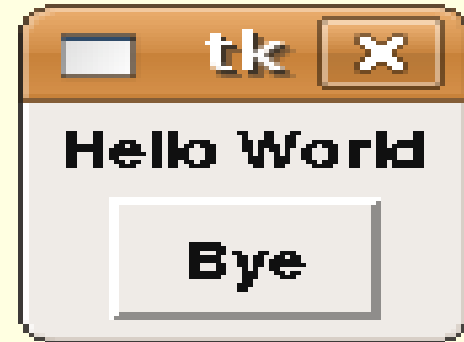
```
        self.bye = Button (self, text="Bye", command=self.quit)
```

```
        self.bye.pack ()
```

```
        self.pack()
```

```
app = Application()
```

```
mainloop()
```





# Exemplo

```
from Tkinter import *
```

Elemento principal derivado de Frame

```
class Application(Frame):
```

```
    def __init__(self, master=None):
```

Construtor da classe base

```
        Frame.__init__(self, master)
```

```
        self.msg = Label(self, text="Hello World")
```

```
        self.msg.pack ()
```

```
        self.bye = Button (self, text="Bye", command=self.quit)
```

```
        self.bye.pack ()
```

```
        self.pack()
```

Janela tem um rótulo e um botão

```
app = Application()
```

Interface é instanciada

```
mainloop()
```

Laço de tratamento de eventos é iniciado

# Classes de componentes

- `Button` Um botão simples usado para executar um comando
- `Canvas` Provê facilidades de gráficos estruturados
- `Checkbox` Representa uma variável que pode ter dois valores distintos (tipicamente um valor booleano). Clicando no botão alterna-se entre os valores
- `Entry` Um campo para entrada de uma linha de texto
- `Frame` Usado como agrupador de widgets
- `Label` Mostra um texto ou uma imagem
- `Listbox` Mostra uma lista de alternativas. Pode ser configurado para ter comportamento de `checkboxbutton` ou `radiobutton`

# Classes de componentes (cont.)

- `Menu` Um painel de menu. Implementa menus de janela, pulldowns e popups
- `Message` Similar ao widget `Label`, mas tem mais facilidade para mostrar texto quebrado em linhas
- `Radiobutton` Representa um possível valor de uma variável que tem um de muitos valores. Clicando o botão, a variável assume aquele valor
- `Scale` Permite especificar um valor numérico através de um ponteiro em uma escala linear
- `Scrollbar` Barra de rolamento para widgets que têm superfície útil variável (`Text`, `Canvas`, `Entry`, `Listbox`)
- `Text` Exibe e permite editar texto formatado. Também suporta imagens e janelas embutidas
- `Toplevel` Uma janela separada

# A Classe Tk

- É a que define uma janela principal e o interpretador Tcl
- Em geral, nunca precisa ser instanciada
  - É instanciada automaticamente quando um widget filho é criado
- Pode ser instanciada explicitamente
- Possui vários métodos, entre os quais
  - `title` (*string*) Especifica o título da janela
  - `geometry`(*string*) Especifica tamanho e posição da janela
    - String tem a forma *largura* $\times$ *altura*+*x*+*y*

# Exemplo

```
from Tkinter import *
```

```
class Application(Frame):
```

```
    def __init__(self, master=None):
```

```
        Frame.__init__(self, master)
```

```
        self.msg = Label(self, text="Hello World")
```

```
        self.msg.pack ()
```

```
        self.bye = Button (self, text="Bye", command=self.quit)
```

```
        self.bye.pack ()
```

```
        self.pack()
```

```
app = Application()
```

```
app.master.title("Exemplo")
```

```
app.master.geometry("200x200+100+100")
```

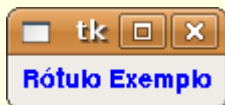
```
mainloop()
```

# Opções de *Widgets*

- *Widgets* (elementos de interface) têm opções com nomenclatura unificada. Ex.:
  - `text` Texto mostrado no elemento
  - `background` cor de fundo
  - `foreground` cor do texto
  - `font` fonte do texto
  - `relief` relevo da borda ('flat', 'raised', 'ridge', 'sunken', 'groove')
- Opções são especificadas
  - No construtor
  - Através do método `configure`

# Exemplo

```
from Tkinter import *  
top = Frame() ; top.pack()  
rotulo = Label (top, text="Rótulo Exemplo",  
                foreground="blue")  
rotulo.pack ()  
rotulo.configure(relief="ridge", font="Arial 24 bold",  
                border=5, background="yellow")
```



# O método `configure`

- Usado com pares do tipo *opção=valor*, modifica os valores dos atributos
- Usado com uma string “*nomeopção*” retorna a configuração da opção com esse nome
  - A configuração é uma tupla com 5 valores
    - nome do atributo
    - nome do atributo no banco de dados (X11)
    - nome da classe no banco de dados (X11)
    - objeto que representa a opção
    - **valor corrente da opção**
- Se `configure` é usado sem argumentos, retorna um dicionário com todas as opções
- Pode-se obter diretamente o valor de uma opção usando o método `cget`



# Exemplo

```
>>> rotulo.configure(relief="ridge")
>>> rotulo.configure("relief")
('relief', 'relief', 'Relief', <index object at
 0x85f9530>, 'ridge')
>>> rotulo.configure()["relief"]
('relief', 'relief', 'Relief', <index object at
 0x85f9530>, 'ridge')
>>> rotulo.configure("relief")[4]
'ridge'
>>> rotulo.cget("relief")
'ridge'
```

# Gerenciando geometrias

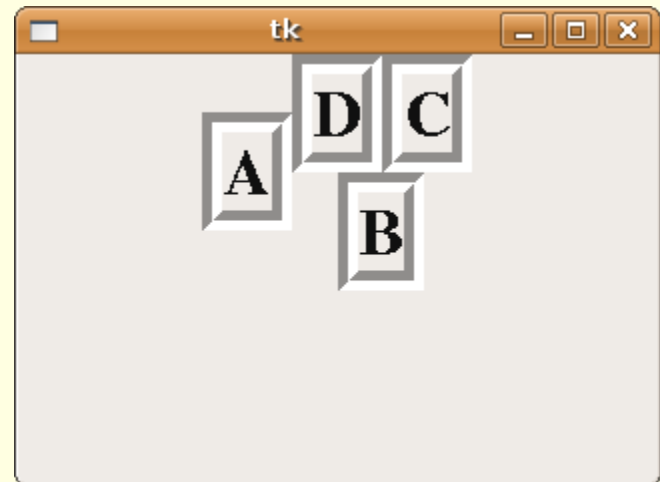
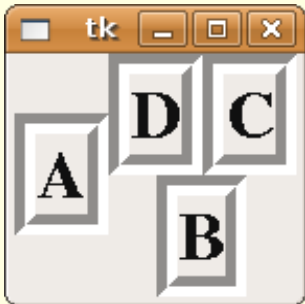
- Todos os elementos de interface ocupam uma área retangular na janela
- A posição e tamanho de cada elemento é determinada por um gerenciador de geometria
  - O elemento não “aparece” enquanto não for informado ao gerenciador
- A geometria resultante depende de
  - Propriedades dos elementos (tamanho mínimo, tamanho da moldura, etc)
  - Opções do gerenciador
  - Algoritmo usado pelo gerenciador
- O gerenciador mais usado em Tk é o **pack**

# Usando o *pack*

- Para informar que um elemento deve ser gerenciado pelo pack, use o método `pack` (*opções*)
- O pack considera o espaço do elemento “pai” como uma cavidade a ser preenchida pelos elementos filhos
- O algoritmo usado pelo pack consiste em empacotar os filhos de um elemento “pai” segundo o lado (`side`) especificado
  - Os lados possíveis são 'top', 'left', 'right' e 'bottom'
  - Deve-se imaginar que sempre que um elemento filho escolhe um lado, a cavidade disponível fica restrita ao lado oposto

# Exemplo

```
from Tkinter import *
top = Frame() ; top.pack()
a = Label (top, text="A") ; a.pack (side="left")
b = Label (top, text="B") ; b.pack (side="bottom")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", border=10,
                    font="Times 24 bold")
top.mainloop()
```



# Redimensionamento

- Por default, o pack não redimensiona os filhos quando o pai é redimensionado
- Duas opções controlam o redimensionamento dos filhos
  - `expand` (booleano)
    - Se verdadeiro, indica que o filho deve tomar toda a cavidade disponível no pai
    - Caso contrário, toma apenas o espaço necessário (default)
  - `fill ('none', 'x', 'y' ou 'both')`
    - Indica como o desenho do elemento irá preencher o espaço alocado
    - 'x' / 'y' indica que irá preencher a largura / altura
    - 'both' indica preenchimento de todo o espaço
    - 'none' indica que apenas o espaço necessário será ocupado (default)

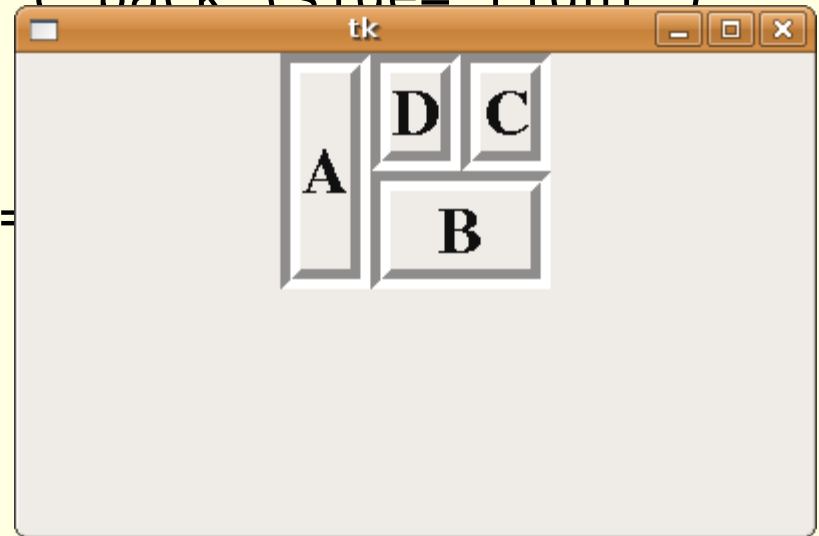
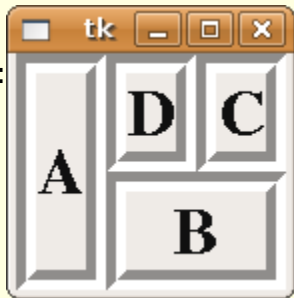
# Exemplo

```
from Tkinter import *
top = Frame() ; top.pack()
a = Label (top, text="A") ; a.pack (side="left",
    fill="y")
b = Label (top, text="B") ; b.pack (side="bottom",
    fill="x")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", border=10,
        font="Times 24 bold")
top.mainloop()
```

# Exemplo

```
from Tkinter import *
top = Frame() ; top.pack()
a = Label (top, text="A") ; a.pack (side="left",
    fill="y")
b = Label (top, text="B") ; b.pack (side="bottom",
    fill="x")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ;
for widget in (a,b,c,d):
```

```
    widget.configure(relief=RAISED, font=
    font.Font(size=24, bold=True))
top.mainloop()
```



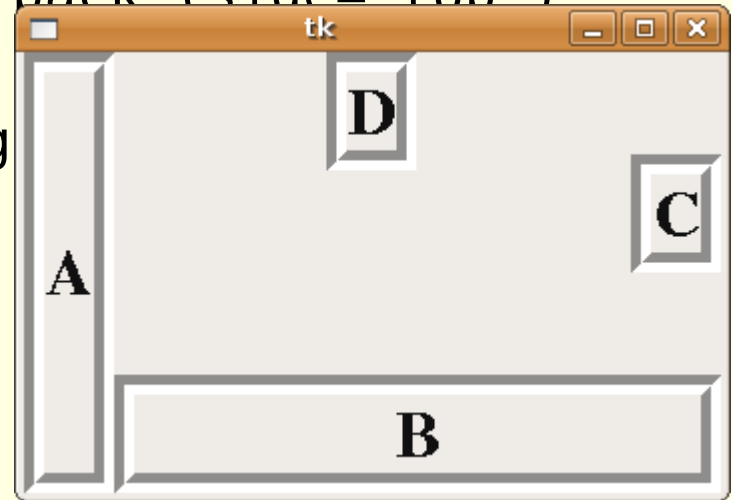
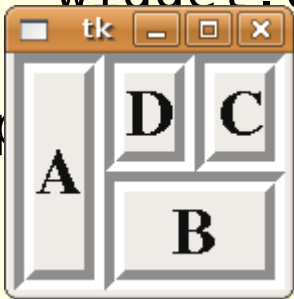
# Exemplo

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
a = Label (top, text="A") ; a.pack
    (side="left", fill="y")
b = Label (top, text="B") ; b.pack
    (side="bottom", fill="x")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", border=10,
        font="Times 24 bold")
top.mainloop()
```



# Exemplo

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
a = Label (top, text="A") ; a.pack
    (side="left",fill="y")
b = Label (top, text="B") ; b.pack
    (side="bottom",fill="x")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove",font=
        ("Helvetica", 24, bold))
top.mainloop()
```

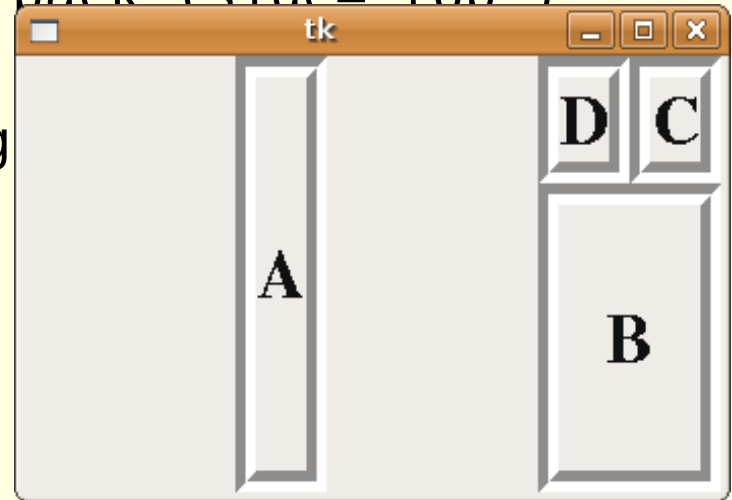
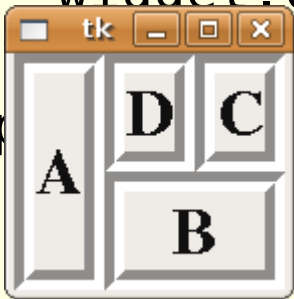


# Exemplo

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
a = Label (top, text="A") ; a.pack
  (side="left", expand=True, fill="y")
b = Label (top, text="B") ; b.pack
  (side="bottom", expand=True, fill="both")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", border=10,
    font="Times 24 bold")
top.mainloop()
```

# Exemplo

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
a = Label (top, text="A") ; a.pack
    (side="left", expand=True, fill="y")
b = Label (top, text="B") ; b.pack
    (side="bottom", expand=True, fill="both")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", font=
        ("Helvetica", 24, "bold"))
top.mainloop()
```



# Usando frames

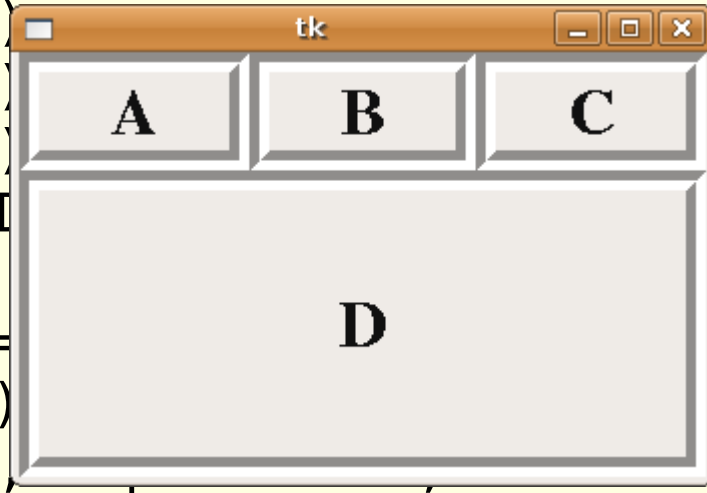
- Frames podem ser usados para auxiliar no layout dos elementos com pack. Ex.:

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
f = Frame (top); f.pack (fill='x')
a = Label (f, text="A")
b = Label (f, text="B")
c = Label (f, text="C")
d = Label (top, text="D")
for w in (a,b,c,d):
    w.configure(relief="groove", border=10,
    font="Times 24 bold")
    w.pack(side="left", expand=True, fill="both")
top.mainloop()
```

# Usando frames

- Frames podem ser usados para auxiliar no layout dos elementos com pack. Ex.:

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
f = Frame (top); f.pack (fill='x')
a = Label (f, text="A")
b = Label (f, text="B")
c = Label (f, text="C")
d = Label (top, text="D")
(a,b,c,d).configure(relief=RAISED,
font="Times 24 bold")
w.pack(side="left", fill="both")
top.mainloop()
```



# Programação com eventos

- Diferente da programação convencional
- O programa não está sob controle 100% do tempo
  - Programa entrega controle ao sistema
  - Em Tk: método(função) `mainloop`
- Interação gera eventos. Ex:
  - Acionamento de um menu ou de um botão
  - Mouse arrastado sobre uma janela
  - Uma caixa de texto teve seu valor alterado
- O tratamento de um evento é feito por uma rotina *“Callback”*

# ***A opção `command`***

- Muitos componentes do Tk suportam a opção `command` que indica uma função a ser invocada sempre que o widget é acionado
- Tipicamente, a função (ou método) usado obtém valores de outros widgets para realizar alguma operação

# Exemplo

```
from Tkinter import *

def inc():
    n=int(rotulo.configure("text")[4])+1
    rotulo.configure(text=str(n))

b = Button(text="Incrementa",command=inc)
b.pack()
rotulo = Label(text="0")
rotulo.pack()
mainloop()
```



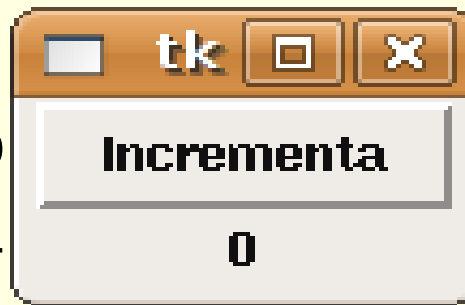
# Exemplo

```
from Tkinter import *
```

```
def inc():
```

```
    n=int(rotulo.get())+1
```

```
    rotulo.config(text=str(n))
```



```
b = Button(text="Incrementa",command=inc)
```

```
b.pack()
```

```
rotulo = Label(text="0")
```

```
rotulo.pack()
```

```
mainloop()
```

# Exemplo

```
from Tkinter import *
```

```
def inc():
```

```
    n=int(rotulo.get())+1
```

```
    rotulo.config(text=str(n))
```

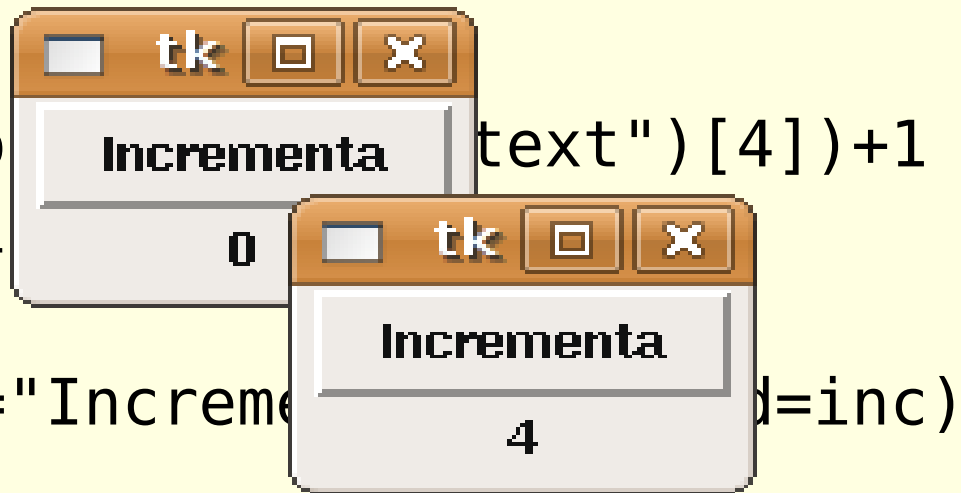
```
b = Button(text="Incrementa", command=inc)
```

```
b.pack()
```

```
rotulo = Label(text="0")
```

```
rotulo.pack()
```

```
mainloop()
```



# Eventos e *Bind*

- Widgets que não dispõem da opção `command` também podem receber eventos e responder a eles
- O método `bind` permite especificar um padrão de eventos ao qual o widget será sensível e uma rotina callback para tratá-lo

`bind(padrão,rotina)`

- *padrão* é uma string que descreve quais eventos a rotina irá tratar
- *rotina* é uma função ou método com exatamente um parâmetro: o evento que deve ser tratado

# Exemplo

```
from Tkinter import *

def clica (e):
    txt = "Mouse clicado em\n%d,%d"%(e.x,e.y)
    r.configure(text=txt)

r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clica)
mainloop()
```

# Exemplo

```
from Tkinter import *  
  
def clicca (e)  
    txt = "Mostrando a posição do clique: " + "%(e.x,e.y)"  
    r.config(text=txt)  
  
r = Label()  
r.pack(expand=1)  
r.master.geometry("300x300")  
r.bind("<Button-1", clicca)  
mainloop()
```



# Exemplo

```
from Tkinter import *
```

```
def clica (e)
```

```
    txt = "Mo
```

```
    r.configu
```

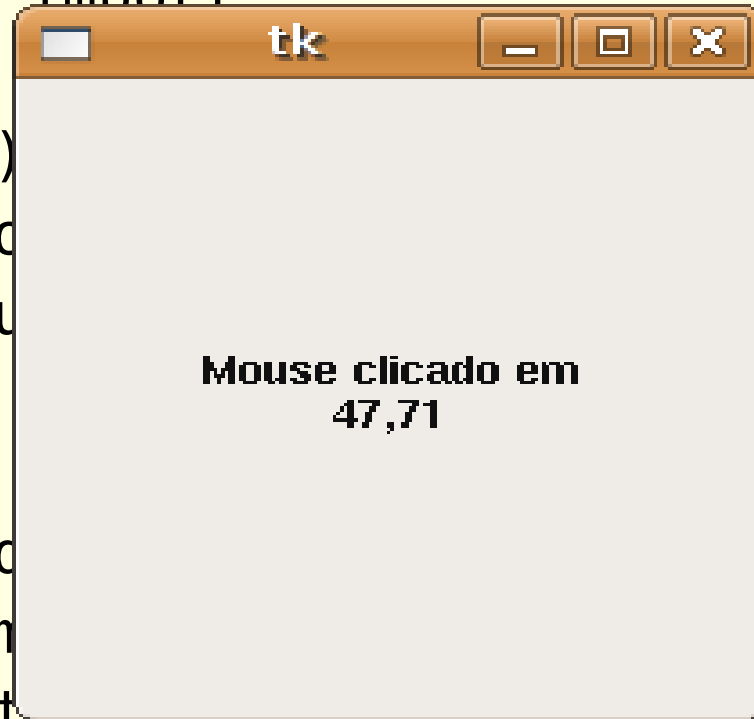
```
r = Label()
```

```
r.pack(expand
```

```
r.master.geom
```

```
r.bind("<Button-1", clica)
```

```
mainloop()
```



```
"%(e.x,e.y)
```

# Campos do objeto evento

- `x, y` : posição do mouse com relação ao canto superior esquerdo do widget
- `x_root, y_root`: posição do mouse com relação ao canto superior esquerdo da tela
- `char`: caractere digitado (eventos de teclado)
- `keysym`: representação simbólica da tecla
- `keycode`: representação numérica da tecla
- `num`: número do botão – 1/2/3=Esquerdo/Meio/Direito – (eventos de mouse)
- `widget`: o objeto que gerou o evento
- `width, height`: largura e altura do widget (evento Configure)

# Padrões de evento (mouse)

- `<Button-i>` para  $i = 1,2,3$ : botão  $i$  do mouse pressionado sobre o widget
- `<Motion>`: mouse arrastado sobre o widget
- `<Bi-Motion>`: mouse arrastado sobre o widget com o botão  $i$  pressionado
- `<ButtonRelease-i>`: botão  $i$  do mouse solto sobre o widget
- `<Double-Button-i>`: botão  $i$  do mouse clicado duas vezes em seguida
- `<Enter>`: O mouse entrou na área do widget
- `<Leave>`: O mouse saiu da área do widget



# Padrões de evento (teclado)


- *character* : O *character* foi digitado sobre o widget
- <Key>: Algum character foi digitado sobre o widget
- <Return>: Tecla *enter* foi digitada
- <Tab>, <F1>, <Up>...: A tecla correspondente foi digitada
- <Shift-Tab>, <Alt-F1>, <Ctrl-Up>...: Tecla com modificador
- Para os eventos serem gerados, é preciso que o *foco* de teclado esteja sobre o widget
  - Depende do sistema de janelas
  - O foco para um widget pode ser forçado usando o método `focus`

# Exemplo

```
from Tkinter import *
def clicca (e):
    txt = "Mouse clicado em\n%d,%d"%(e.x,e.y)
    r.configure(text=txt)
    r.focus()
def tecla(e):
    txt="Keysym=%s\nKeycode=%s\nChar=%s"\
        %(e.keysym,e.keycode,e.char)
    r.configure(text=txt)
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clicca)
r.bind("<Key>", tecla)
```

# Exemplo

```
from Tkinter import *
def clicca (e):
    txt = "Mouse"
    r.configure(
    r.focus()
def tecla(e):
    txt="Keysym="
    %(e.key
    r.configure(
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clicca)
r.bind("<Key>", tecla)
```



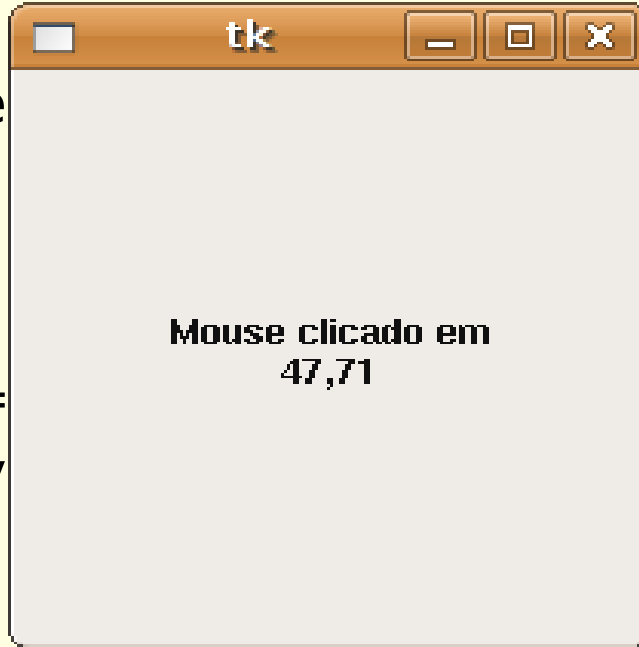
(e.x,e.y)

=%s "\

)

# Exemplo

```
from Tkinter import *
def clicca (e):
    txt = "Mouse clicado em %s" % (e.x,e.y)
    r.configure(text=txt)
    r.focus()
def tecla(e):
    txt="Keysym=%s" % (e.keysym)
    r.configure(text=txt)
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clicca)
r.bind("<Key>", tecla)
```




(e.x,e.y)

=%s "\  
)

# Exemplo

```
from Tkinter import *
def clicca (e):
    txt = "Mouse"
    r.configure(
    r.focus()
def tecla(e):
    txt="Keysym="
    %(e.key
    r.configure(
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clicca)
r.bind("<Key>", tecla)
```



(e.x,e.y)


Keysym=a  
Keycode=38  
Char=a

=%s "\

)

# Exemplo

```
from Tkinter import *
def clicca (e):
    txt = "Mouse"
    r.configure(
    r.focus()
def tecla(e):
    txt="Keysym="
    %(e.key
    r.configure(
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clicca)
r.bind("<Key>", tecla)
```



(e.x,e.y)

Keysym=Alt\_L  
Keycode=64  
Char=

= %s \\

)

# Menus

- Podem ser associados a uma janela (menus toplevel), pulldown, popup e em cascata a partir de outro menu
- Todos são instâncias da classe Menu
- Um menu é composto de itens que podem ser
  - `command` quando pressionado executa uma callback
  - `checkbox` parecido com `command`, mas tem um valor booleano associado
  - `radiobutton` como `command`, mas representa um de vários estados mutuamente exclusivos
  - `cascade` ativa um outro menu em cascata
- Para adicionar um item a um menu, use métodos da forma `add ("tipo", opções)` ou `add_tipo(opções)`

# Menu de janela (toplevel)

- É tipicamente exibido horizontalmente no topo da janela
  - Aspecto depende do sistema operacional
- Se um outro menu é associado como item *cascade*, ele é tratado como *pulldown*, isto é, é exibido sob o item do menu de janela
- Assim como outros menus, não necessita ter sua geometria gerenciada (e.g., pack ou grid)
- Para associar a uma janela, usa-se a opção *menu* do objeto janela.



# Exemplo

```
from Tkinter import *
def abrir(): print "abrir"
def salvar(): print "salvar"
def ajuda() : print "ajuda"
top=Tk()
principal=Menu(top)
arquivo=Menu(principal)
arquivo.add_command(label="Abrir",command=abrir)
arquivo.add_command(label="Salvar",command=salvar)
principal.add_cascade(label="Arquivo",menu=arquivo)
principal.add_command(label="Ajuda",command=ajuda)
top.configure(menu=principal)
```

# Exemplo

```
from Tkinter import *
```

```
def abrir(): print "abrir"
```

```
def salvar(): pr
```

```
def ajuda() : pr
```

```
top=Tk()
```

```
principal=Menu(t
```

```
arquivo=Menu(pri
```

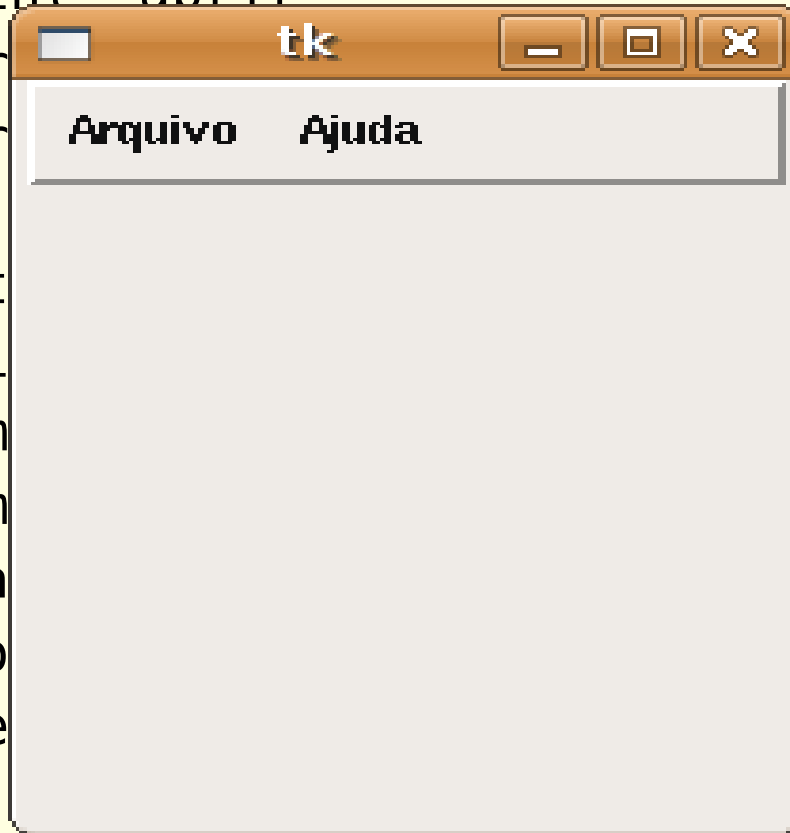
```
arquivo.add_comm
```

```
arquivo.add_comm
```

```
principal.add_ca
```

```
principal.add_co
```

```
top.configure(me
```



```
nd=abrir)
```

```
nd=salvar)
```

```
nu=arquivo)
```

```
and=ajuda)
```

# Exemplo

```
from Tkinter import *
```

```
def abrir(): print "abrir"
```

```
def salvar(): pr
```

```
def ajuda() : pr
```

```
top=Tk()
```

```
principal=Menu(t
```

```
arquivo=Menu(pri
```

```
arquivo.add_comm
```

```
arquivo.add_comm
```

```
principal.add_ca
```

```
principal.add_co
```

```
top.configure(me
```



```
nd=abrir)
```

```
nd=salvar)
```

```
nu=arquivo)
```

```
and=ajuda)
```

# Menus Popup

- Um menu popup é aquele que é exibido numa janela independente
- Para que o menu seja exibido, é preciso invocar o método `post`:  
`post (x, y)`
  - onde `x` e `y` são as coordenadas do canto superior esquerdo do menu com relação ao canto superior esquerdo da tela

# Exemplo

```
from Tkinter import *

def alo(): print "Alo!"

root = Tk()
menu = Menu(root, tearoff=0)
menu.add_command(label="Alo 1", command=alo)
menu.add_command(label="Alo 2", command=alo)

def popup(e): menu.post(e.x_root, e.y_root)

frame = Frame(root, width=200, height=200)
frame.pack()
frame.bind("<Button-3>", popup)
mainloop()
```

# Exemplo

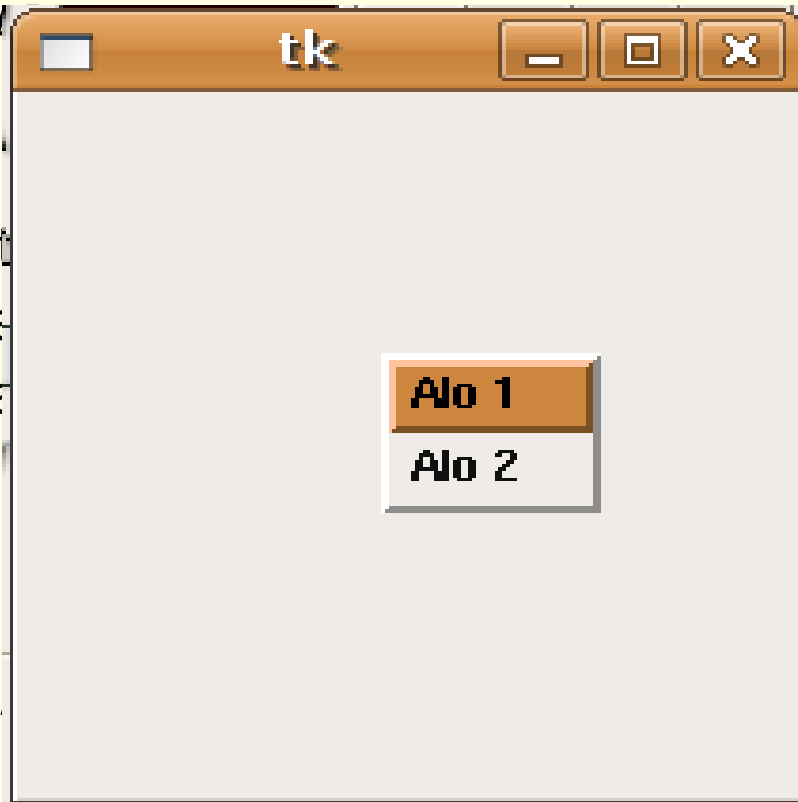
```
from Tkinter import *

def alo(): print "Alo"

root = Tk()
menu = Menu(root, tearoff=0)
menu.add_command(label="Alo 1", command=alo)
menu.add_command(label="Alo 2", command=alo)

def popup(e): menu.post(e.x_root, e.y_root)

frame = Frame(root)
frame.pack()
frame.bind("<Button-3>", popup)
mainloop()
```



# Variáveis

- Tk é controlado por um interpretador Tcl (e não diretamente pelo python)
- Em alguns casos, deseja-se usar usar variáveis na interface
  - Por exemplo, é possível especificar que o texto exibido em um `Label` é o valor de uma variável (e não uma constante)
    - Nesse caso, usa-se a opção `textvar` ao invés de `text`
- Variáveis Tcl são expostas à aplicação Python através das classes `StringVar`, `IntVar` e `DoubleVar`
  - O construtor é da forma `StringVar(master)` onde `master` é uma janela ou widget
- Instâncias dessas classes possuem os métodos `get` e `set` que podem ser usados para acessar os valores armazenados no interpretador Tcl

# Exemplo

```
from Tkinter import *

root = Tk()
soma = DoubleVar(root)
parcela = DoubleVar(root)
def aritmetica (e):
    soma.set(soma.get()+parcela.get())

lsoma = Label(textvar=soma)
eparcela = Entry(textvar=parcela)
eparcela.bind("<Return>", aritmetica)
lsoma.pack()
eparcela.pack()
```



# Exemplo

```
from Tkinter import *
```

```
root = Tk()
```

```
soma = DoubleVar()
```

```
parcela = DoubleVar()
```

```
def aritmetica():
```

```
    soma.set(soma.get()+parcela.get())
```

```
lsoma = Label(textvar=soma)
```

```
eparcela = Entry(textvar=parcela)
```

```
eparcela.bind("<Return>", aritmetica)
```

```
lsoma.pack()
```

```
eparcela.pack()
```



# Exemplo

```
from Tkinter import *
```

```
root = Tk()
```

```
soma = DoubleVar()
```

```
parcela = DoubleVar()
```

```
def aritmetica():
```

```
    soma.set(soma.get()+parcela.get())
```

```
lsoma = Label(textvar=soma)
```

```
eparcela = Entry(textvar=parcela)
```

```
eparcela.bind("<Return>", aritmetica)
```

```
lsoma.pack()
```

```
eparcela.pack()
```



# Exemplo

```
from Tkinter import *
```

```
root = Tk()
```

```
soma = DoubleVar()
```

```
parcela = DoubleVar()
```

```
def aritmetica():
```

```
    soma.set(soma.get()+parcela.get())
```

```
lsoma = Label(textvar=soma)
```

```
eparcela = Entry(textvar=parcela)
```

```
eparcela.bind("<Return>", aritmetica)
```

```
lsoma.pack()
```

```
eparcela.pack()
```



# Checkbuttons

- Checkbutton Representa uma variável que pode ter dois valores distintos (tipicamente um valor booleano). Clicando no botão alterna-se entre os valores
- A callback especificada pela opção `command` é chamada sempre que a variável muda de valor
- Estado é armazenado pela variável Tcl especificada pela opção `variable`
- Se a variável é inteira, o valor correspondente ao checkbutton “desligado” / “ligado” é 0/1
- É possível usar um checkbutton com uma variável string
  - Nesse caso, os valores correspondentes a “desligado” / “ligado” são especificados com as opções `offvalue` e `onvalue`

# Exemplo

```
from Tkinter import *

root = Tk()
v1 = IntVar(root)
v2 = StringVar(root)

def exibe():
    l.config (text="v1=%d,v2=%s"%(v1.get(),v2.get()))

c1 = Checkbutton (text="V1", var=v1, command=exibe)
c2 = Checkbutton (text="V2", var=v2, command=exibe,\
                 onvalue="Sim", offvalue="Nao")

l = Label()
for w in (c1,c2,l):w.pack()
exibe()
```

# Exemplo

```
from Tkinter import *
```

```
root = Tk()
```

```
v1 = IntVar(root)
```

```
v2 = StringVar(root)
```

```
def exibe():
```

```
    l.config (text=v1.get(),v2.get())
```

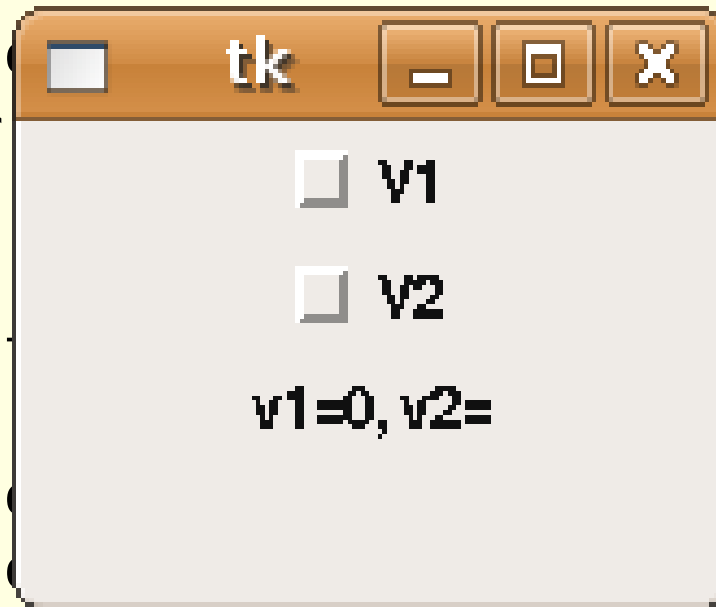
```
c1 = Checkbutton(root, text="v1",
```

```
c2 = Checkbutton(root, text="v2",
```

```
l = Label(root)
```

```
for w in (c1,c2,l):w.pack()
```

```
exibe()
```



```
        v1.get(),v2.get())
```

```
        command=exibe)
```

```
        command=exibe,\
```

```
        onvalue="Sim", offvalue="Nao")
```

# Exemplo

```
from Tkinter import *
```

```
root = Tk()
```

```
v1 = IntVar(root)
```

```
v2 = StringVar(root)
```

```
def exibe():
```

```
    l.config (text=v1.get(),v2.get())
```

```
c1 = Checkbutton(root, text="v1",
```

```
c2 = Checkbutton(root, text="v2",
```

```
l = Label(root)
```

```
for w in (c1,c2,l):w.pack()
```

```
exibe()
```



```
        v1.get(),v2.get())
```

```
        command=exibe)
```

```
        command=exibe,\
```

```
        onvalue="Sim", offvalue="Nao")
```

# Exemplo

```
from Tkinter import *
```

```
root = Tk()
```

```
v1 = IntVar(root)
```

```
v2 = StringVar(root)
```

```
def exibe():
```

```
    l.config (text=v1.get(),v2.get())
```

```
c1 = Checkbutton(root, text="v1",
```

```
c2 = Checkbutton(root, text="v2",
```

```
l = Label(root)
```

```
for w in (c1,c2,l):w.pack()
```

```
exibe()
```



```
        v1.get(),v2.get())
```

```
        command=exibe)
```

```
        command=exibe,\
```

```
        onvalue="Sim", offvalue="Nao")
```



# Radiobuttons

- Radiobutton representa um possível valor de uma variável que tem um de muitos valores. Clicando o botão, a variável assume aquele valor
- A variável é especificada com a opção `variable` e o valor associado com a opção `value`
- Os radiobuttons que se referem à mesma variável funcionam em conjunto
  - Ex.: ligar um faz com que outro seja desligado
- Um radiobutton é mostrado com um indicador ao lado
  - Pode-se desabilitar o indicador usando a opção `indicatoron=False`
  - Nesse caso, é mostrado como um botão normal

# Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar(root)
cor.set("black")
l = Label(background=cor.get())
l.pack(fill='both',expand=True)
def pinta(): l.configure(background=cor.get())
for txt,val in (("preto","black"),\
                ("vermelho","red"),\
                ("azul","blue"), ("verde","green")):
    Radiobutton(text=txt,value=val,variable=cor,\
                command=pinta).pack(anchor=W)
mainloop()
```

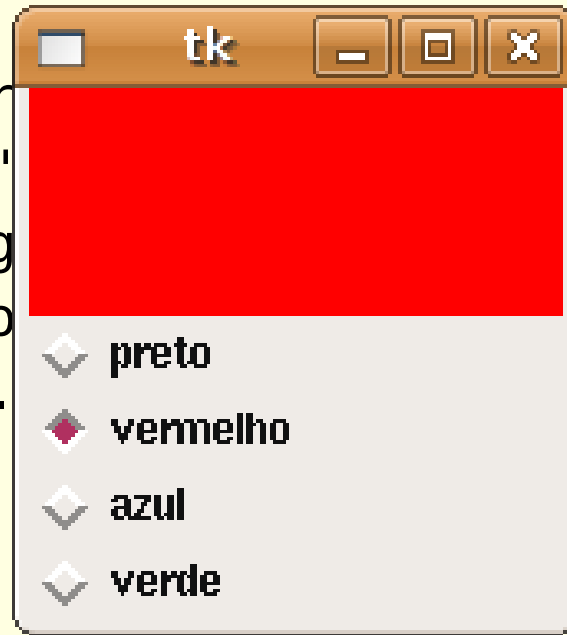
# Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar()
cor.set("black")
l = Label(background="black")
l.pack(fill='both')
def pinta(): l.config(background=cor.get())
for txt, val in [("preto", "black"), ("vermelho", "red"), ("azul", "blue"), ("verde", "green")]:
    Radiobutton(text=txt, value=val, variable=cor, \
                command=pinta).pack(anchor=W)
mainloop()
```



# Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar()
cor.set("black")
l = Label(background="black")
l.pack(fill='both')
def pinta(): l.config(background=cor.get())
for txt,val in [("preto", "black"), ("vermelho", "red"), ("azul", "blue"), ("verde", "green")]:
    Radiobutton(text=txt, value=val, variable=cor, \
                command=pinta).pack(anchor=W)
mainloop()
```



# Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar()
cor.set("black")
l = Label(background="black")
l.pack(fill='both')
def pinta(): l.config(background=cor.get())
for txt, val in [("preto", "black"), ("vermelho", "red"), ("azul", "blue"), ("verde", "green")]:
    Radiobutton(text=txt, value=val, variable=cor, \
                command=pinta).pack(anchor=W)
mainloop()
```

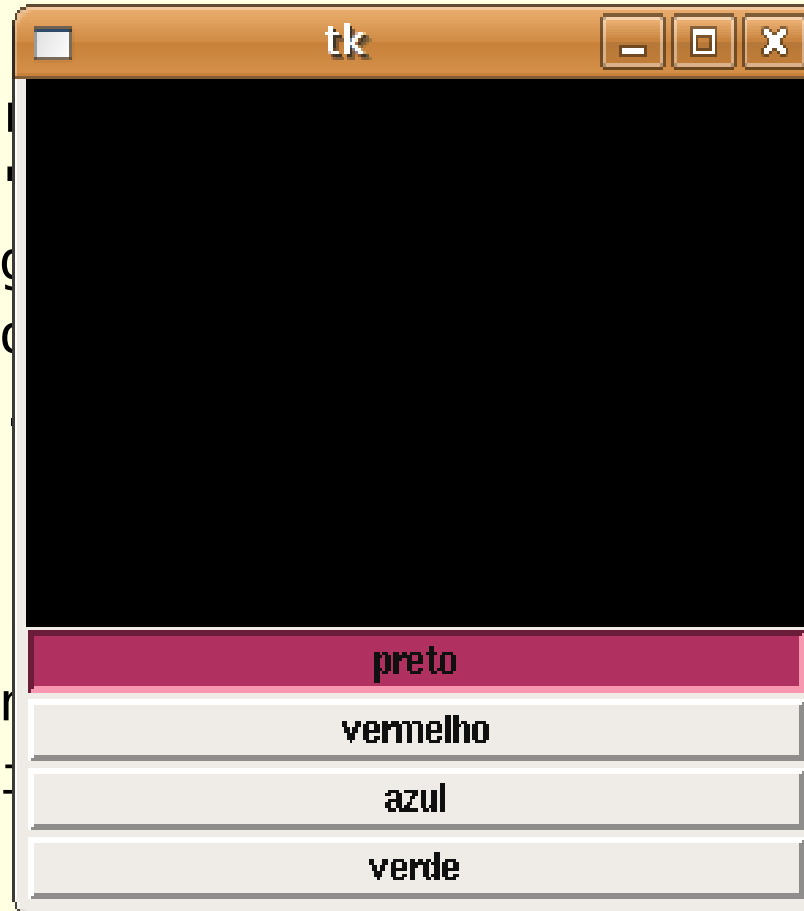


# Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar(root)
cor.set("black")
l = Label(background=cor.get())
l.pack(fill='both',expand=True)
def pinta(): l.configure(background=cor.get())
for txt,val in (("preto","black"),\
                ("vermelho","red"),\
                ("azul","blue"), ("verde","green")):
    Radiobutton(text=txt,value=val,variable=cor,\
                command=pinta,indicatoron=False).pack(fill='x')
mainloop()
```

# Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar()
cor.set("black")
l = Label(background=cor)
l.pack(fill='both')
def pinta(): l.config(background=cor.get())
for txt, val in
```



```
Radiobutton(
    command=p
)
mainloop()
```

```
.get())
("green")):
able=cor,\
ack(fill='x')
```

# Entry

- Um `Entry` permite entrada/edição de uma linha de texto
- O texto associado ao `Entry` é normalmente armazenado numa variável indicada pela opção `textvariable`
  - Se não indicada, é usada uma variável interna cujo valor pode ser obtido usando o método `get()`
- Há diversos métodos para manipular diretamente o texto
  - Usam o conceito de índices (não confundir com os índices usado pelo Python)
  - Por exemplo, o índice `INSERT` indica a posição do texto onde o cursor de inserção se encontra, `0` a posição antes do primeiro caractere e `END` a posição ao final do texto



# Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT, "*")
def limpa(): e.delete(INSERT, END)
e=Entry(font="Arial 24")
i=Button(text="Insere*", command=insere)
l=Button(text="Limpa", command=limpa)
e.pack()
for w in (i,l): w.pack(side='left')
mainloop()
```

# Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT, "*")
def limpa(): e.delete(INSERT, END)
e=Ent
i=But
l=But
e.pac
for w
mainloop()
```



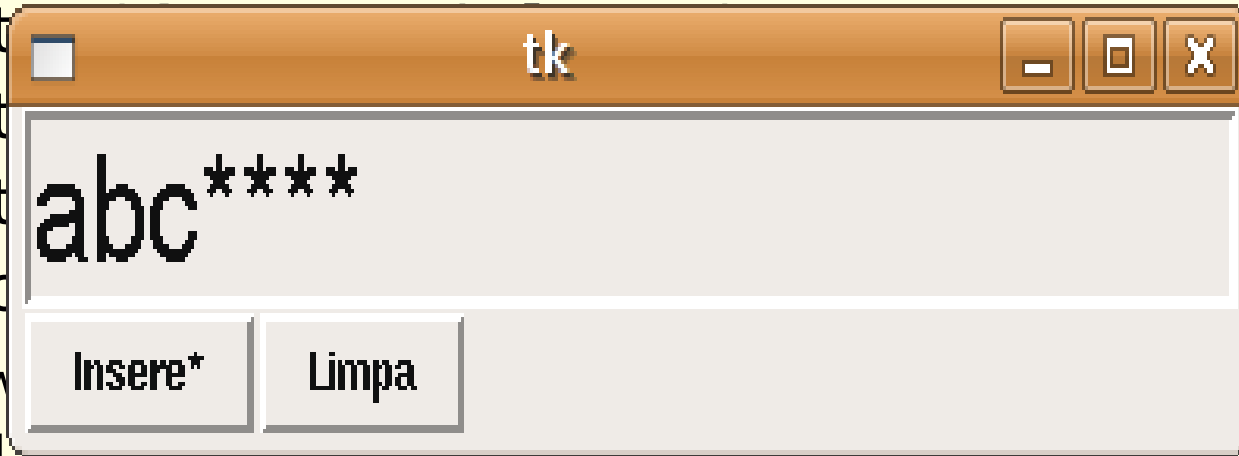
# Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT, "*")
def limpa(): e.delete(INSERT, END)
e=Ent
i=But
l=But
e.pac
for w
mainloop()
```



# Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT, "*")
def limpa(): e.delete(INSERT, END)
e=Ent
i=But
l=But
e.pac
for w
mainloop()
```



# Canvas

- Permite a exibição e edição de gráficos estruturados 2D
- Elementos gráficos (itens) são introduzidos usando métodos da forma `create_tipo (...)`, onde *tipo* pode ser
  - `arc` arco de círculo
  - `bitmap` imagem binária
  - `image` imagem colorida
  - `line` linha poligonal
  - `oval` círculos e elipses
  - `polygon` polígonos
  - `rectangle` retângulo
  - `text` texto
  - `window` um widget tk

# Exemplo

```
from Tkinter import *
c = Canvas()
c.pack()
o = c.create_oval(1,1,200,100,outline="blue",\
                 width=5,fill="red")
widget = Button(text="Tk Canvas")
w = c.create_window(10,120,window=widget,anchor=W)
l = c.create_line(100,0,120,30,50,60,100,120,\
                 fill="black",width=2)
r = c.create_rectangle(40,150,100,200,fill="white")
img = PhotoImage(file="python.gif")
i = c.create_image (150,150,image=img,anchor=NW)
a = c.create_arc (150,90,250,190,start=30,extent=60,\
                 outline="green",fill="orange")
t = c.create_text(200,35,text="Texto\nTexto",
                 font="Arial 22")
```

# Exemplo

```
from Tkinter import *
```

```
c = Canvas()
```

```
c.pack()
```

```
o = c.create
```

```
width
```

```
widget = E
```

```
w = c.create
```

```
l = c.create
```

```
fill
```

```
r = c.create
```

```
img = Photo
```

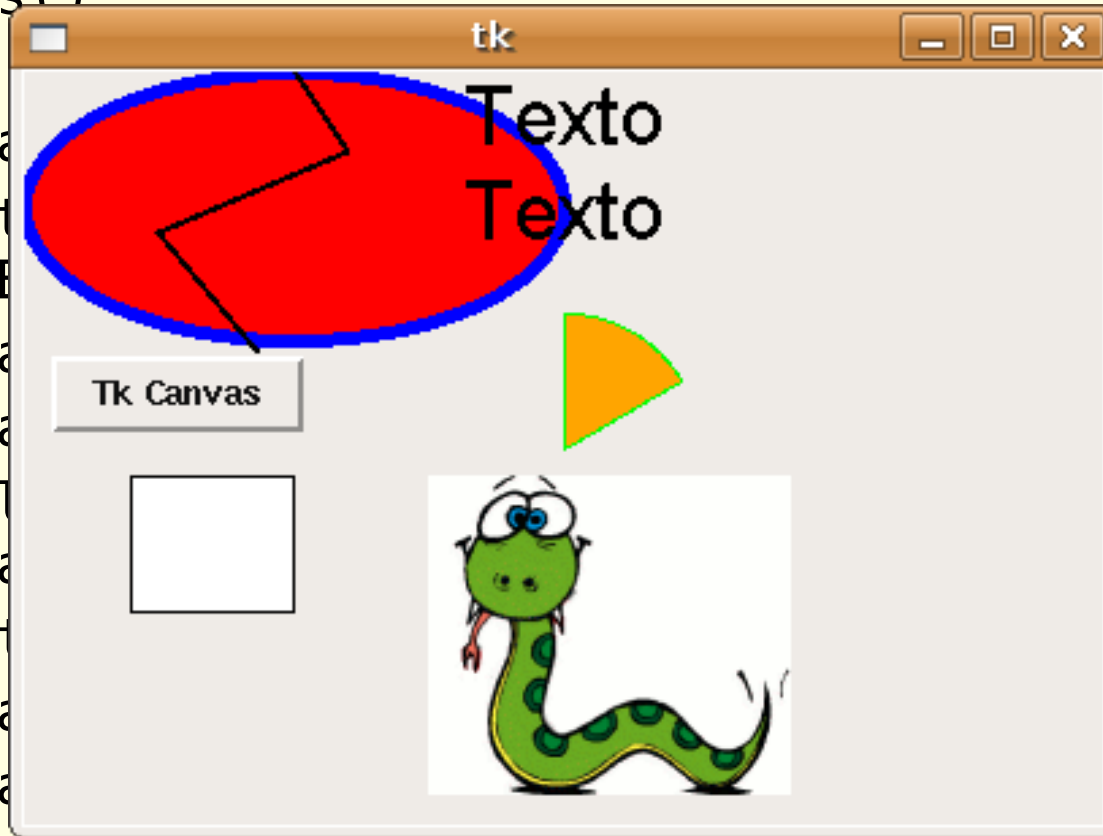
```
i = c.create
```

```
a = c.create
```

```
outline="green",fill="orange")
```

```
t = c.create_text(200,35,text="Texto\nTexto",
```

```
font="Arial 22")
```



```
,\
```

```
hor=W)
```

```
,\
```

```
white")
```

```
=NW)
```

```
tent=60,\
```

# Coordenadas de Itens

- Todos os métodos `create_item` têm como primeiros argumentos um par de coordenadas  $x,y$  do item
  - Os itens `oval` e `rectangle` requerem mais um par de coordenadas para delimitar a extensão (caixa envolvente)
  - Os itens `line` e `polygon` podem ser seguidos por outros pares de coordenadas que especificam demais vértices
- As coordenadas referem-se a um sistema de coordenadas próprio que pode ser diferente do da janela
  - A área do canvas que deve ser mostrada na janela pode ser modificada pela opção  
`scrollarea=(xmin,ymin,xmax,ymax)`
  - Para obter as coordenadas do canvas dadas as coordenadas da janela usa-se os métodos `canvasx(x)` e `canvasy(y)`



# Identificação de Itens

- Todo item de um canvas tem um identificador numérico que é retornado pelo método `create_item`
- Pode-se também associar tags (etiquetas) a itens
  - Usa-se a opção `tags=tags` onde `tags` pode ser uma string ou uma tupla com várias strings
  - Uma mesma etiqueta pode ser associada a mais de um item
- O identificador `ALL` refere-se a todos os itens do canvas
- O identificador `CURRENT` refere-se ao item do canvas sob o cursor do mouse
  - Usado em callbacks de canvas para alterar propriedades dos itens clicados

# Métodos de Canvas

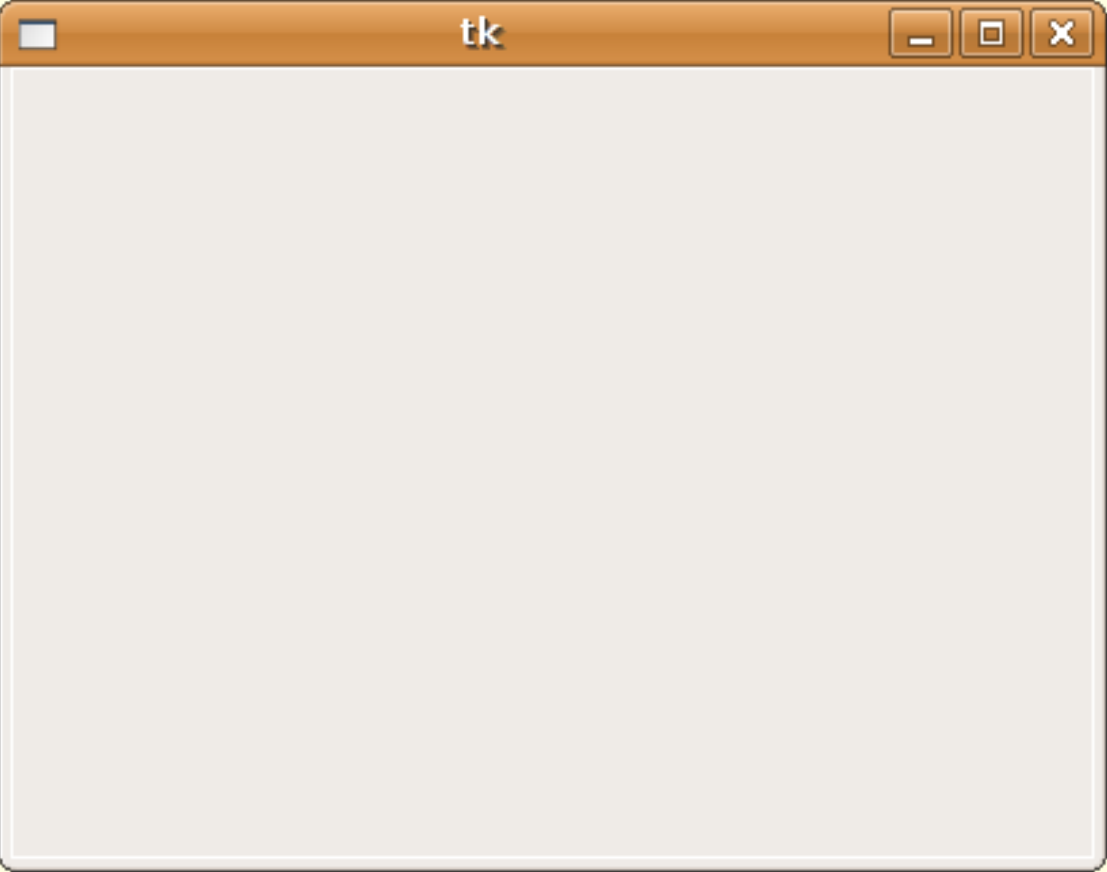
- `itemconfig` (*itemOuTag*, ...) altera opções do(s) item(s)
- `tag_bind`(*itemOuTag*, *padrão*, *callback*) associa uma *callback* a um *padrão* de eventos sobre o(s) item(s)
- `delete`(*itemOuTag*) remove o(s) item(s)
- `move`(*itemOuTag*, *dx*, *dy*) translada o(s) item(s)
- `coords`(*itemOuTag*, *x1*, *x2*, ..., *xN*, *yN*) altera as coordenadas do(s) item(s)
- `coords`(*item*) retorna as coordenadas do item
- `bbox`(*itemOuTag*) retorna uma tupla com a caixa envolvente dos itens
- `itemcget`(*item*, *opção*) retorna o valor da *opção* dada do *item*

# Exemplo

```
from Tkinter import *
c = Canvas()
c.pack()
def novalinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    c.create_line(x,y,x,y,tags="corrente")
def estendelinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    coords = c.coords("corrente") + [x,y]
    c.coords("corrente",*coords)
def fechalinha(e): c.itemconfig("corrente",tags=())
c.bind("<Button-1>", novalinha)
c.bind("<B1-Motion>", estendelinha)
c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```

# Exemplo

```
from Tkinter import *
c = Canvas
c.pack()
def novali
    x,y =
    c.crea
def estend
    x,y =
    coords
    c.coor
def fechalinha
    tags=()
c.bind("<B
c.bind("<B
c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```

A screenshot of a Tkinter window titled "tk". The window has a standard Mac OS-style title bar with a red close button, a yellow maximize button, and a green minimize button. The main content area of the window is a large, empty white rectangle, representing a canvas. The window is positioned in the center of the slide, overlapping the Python code on the left.

# Exemplo

```
from Tkinter import *
```

```
c = Canvas
```

```
c.pack()
```

```
def novali
```

```
    x,y =
```

```
    c.crea
```

```
def estend
```

```
    x,y =
```

```
    coords
```

```
    c.coor
```

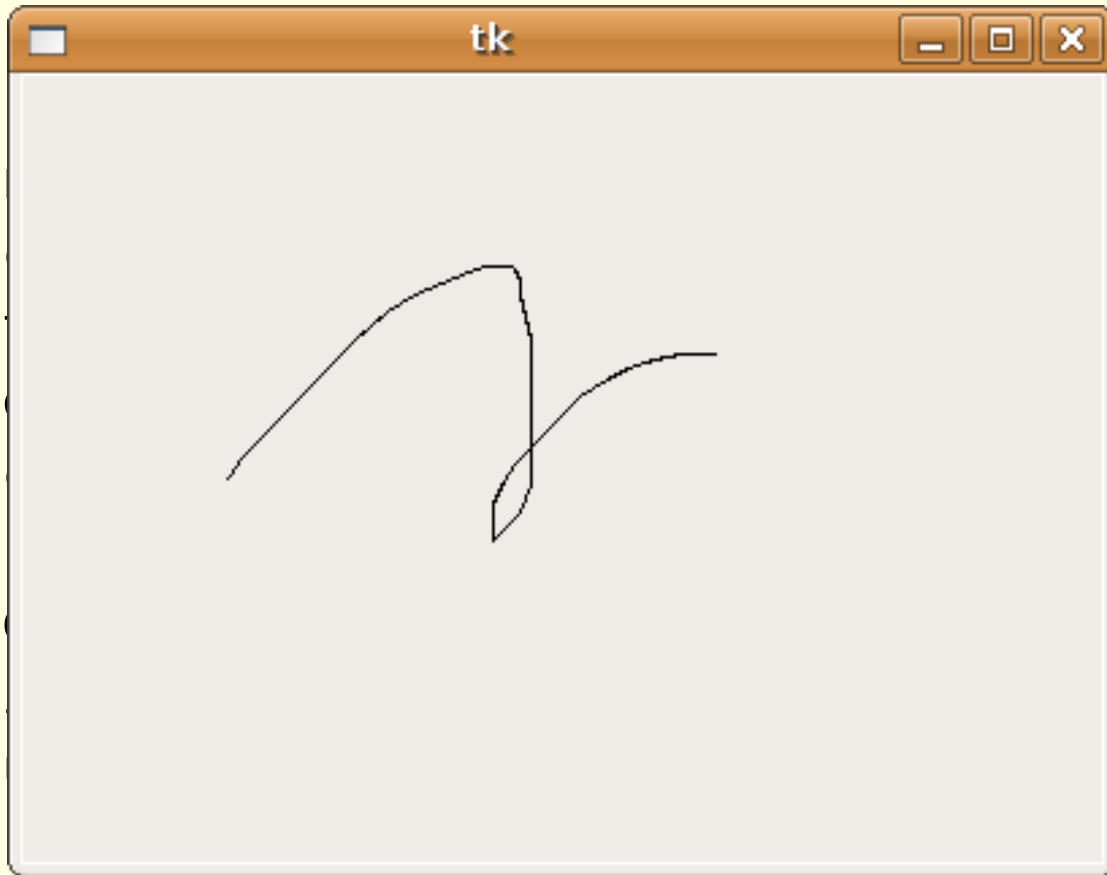
```
def fechal
```

```
c.bind("<B
```

```
c.bind("<B
```

```
c.bind("<ButtonRelease-1>", fechalinha)
```

```
c.pack()
```



```
ags=())
```

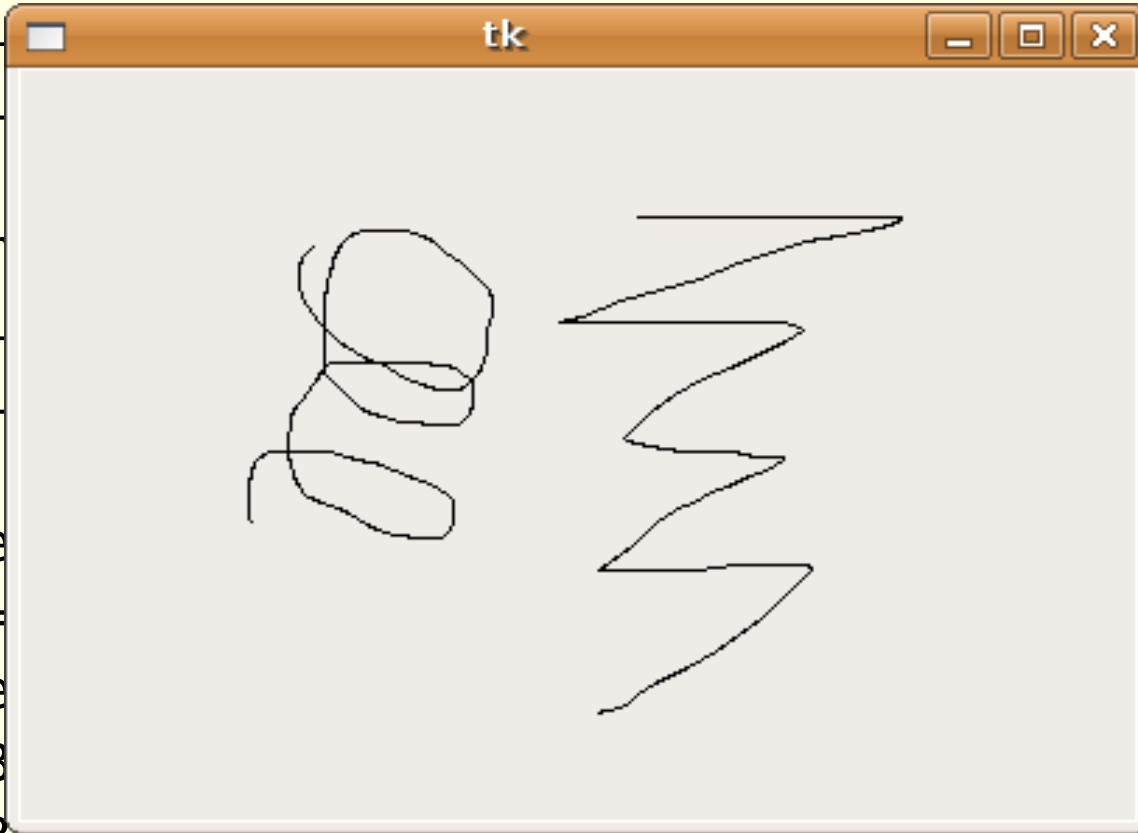


# Exemplo

```
...
def selescionalinha(e):
    global x0,y0
    x0,y0 = c.canvasx(e.x), c.canvasy(e.y)
    c.itemconfig(CURRENT, tags="sel")
def movelinha (e):
    global x0,y0
    x1,y1 = c.canvasx(e.x), c.canvasy(e.y)
    c.move("sel",x1-x0,y1-y0)
    x0,y0=x1,y1
def deselescionalinha(e): c.itemconfig("sel", tags=())
c.bind("<Button-3>", selescionalinha)
c.bind("<B3-Motion>", movelinha)
c.bind("<ButtonRelease-3>", deselescionalinha)
```

# Exemplo

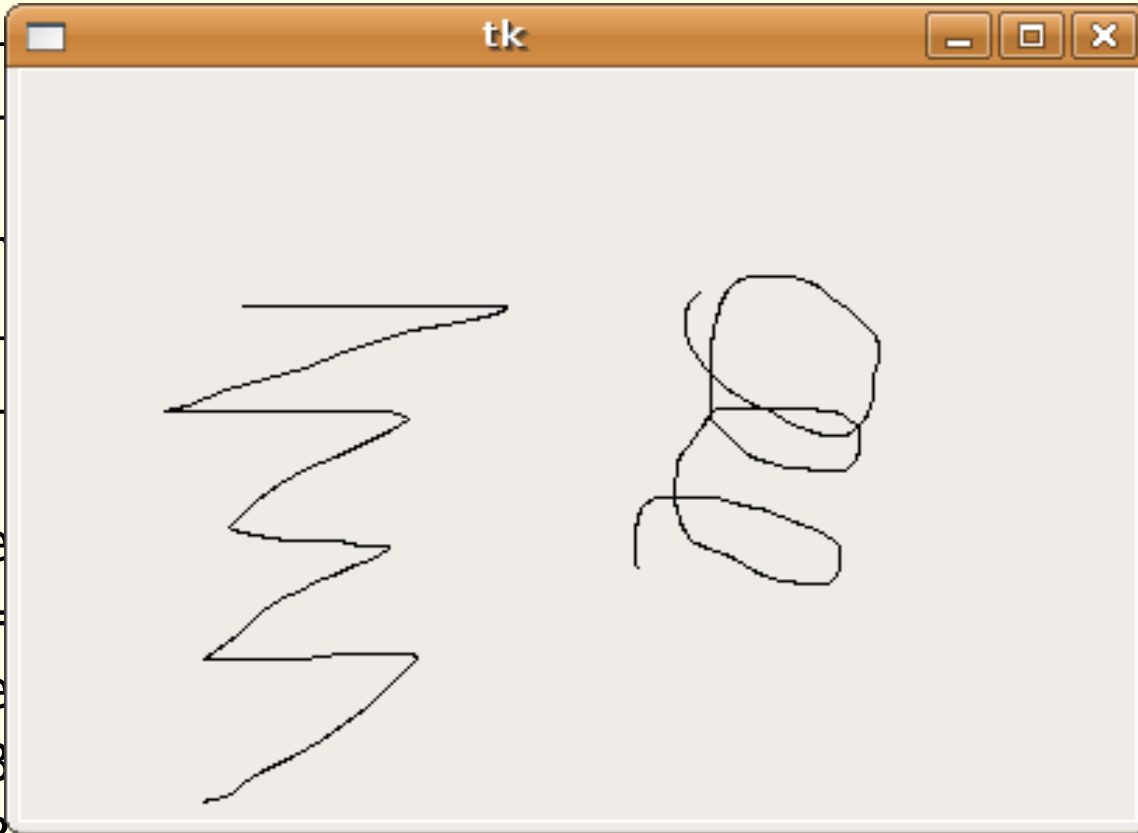
```
...
def seleci
    global
    x0,y0
    c.item
def moveli
    global
    x1,y1
    c.move
    x0,y0=
def desele
    c.bind("<B
    c.bind("<BS-motion> ", movelinha)
    c.bind("<ButtonRelease-3>", deseleccionalinha)
tags=()
```





# Exemplo

```
...
def seleci
    global
    x0,y0
    c.item
def moveli
    global
    x1,y1
    c.move
    x0,y0=
def desele
    c.bind("<B
    c.bind("<BS-motion> ", movelinha)
    c.bind("<ButtonRelease-3>", deseleccionalinha)
tags=()
```



# Scrollbar

- Barras de rolamento são usadas com outros widgets com área útil maior do que pode ser exibida na janela (Canvas, Text, Listbox, Entry)
- Uma barra de rolamento horizontal (vertical) funciona chamando o método `xview` (`yview`) do widget associado
  - Isto é feito configurando a opção `command` da barra
- Por outro lado, sempre que a visão do widget muda, a barra de rolamento precisa ser atualizada
  - Isto é feito configurando a opção `xscrollcommand` (ou `yscrollcommand`) do widget ao método `set` da barra

# Exemplo

```
from Tkinter import *
lb = Listbox()
lb.pack(side=LEFT, expand=True, fill="both")
sb = Scrollbar()
sb.pack(side=RIGHT, fill="y")
sb.configure(command=lb.yview)
lb.configure(yscrollcommand=sb.set)
for i in range(100):
    lb.insert(END, i)
```

# Exemplo

```
from Tkinter import *
lb = Listbox
lb.pack(side=TOP, fill="both")
sb = Scrollbar
sb.pack(side=RIGHT, fill="y")
sb.configure(command=lb.yview)
lb.configure(yscrollcommand=sb.set)
for i in range(10):
    lb.insert(0, i)
```

