

# **Python: Interfaces Gráficas com Tk**

UFRJ

# 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(master=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 podem ser criadas instanciando-se 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=master.destroy)
```

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

```
        self.pack()
```

```
if __name__ == '__main__':
```

```
    master = Tk()
```

```
    app = Application(master)
```

```
    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=master.destroy)
```

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

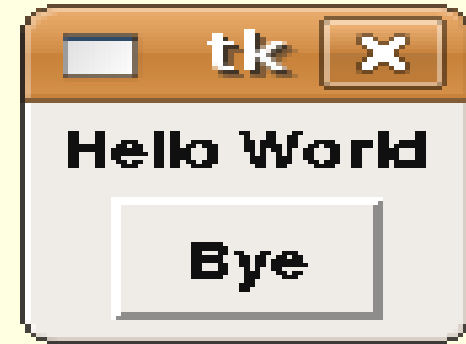
```
        self.pack()
```

```
if __name__ == '__main__':
```

```
    master = Tk()
```

```
    app = Application(master)
```

```
    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=master.destroy)
```

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

```
        self.pack()
```

Janela tem um  
rótulo e um botão

```
if __name__ == '__main__':
```

```
    master = Tk()
```

```
    app = Application(master)
```

```
    mainloop()
```

Interface é  
instanciada

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
- **Checkbutton** - 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 checkbutton 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 *larguraXaltura+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.master.destroy)
        self.bye.pack ()
        self.pack()

if __name__ == '__main__':
    master = Tk()
    master.title("Exemplo")
    master.geometry("640x480+200+200")
    app = Application(master)
    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

```
def exemplo2():  
    master=Tk()  
    top = Frame(master)  
    top.pack()  
    rotulo = Label(top, text="Rótulo Exemplo",  
foreground="blue")  
    rotulo.pack()  
    rotulo.configure(relief="ridge", font="Arial 24 bold",  
border=5,  
background="yellow")  
    mainloop()  
  
if __name__ == '__main__':  
    exemplo2()
```



# 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**

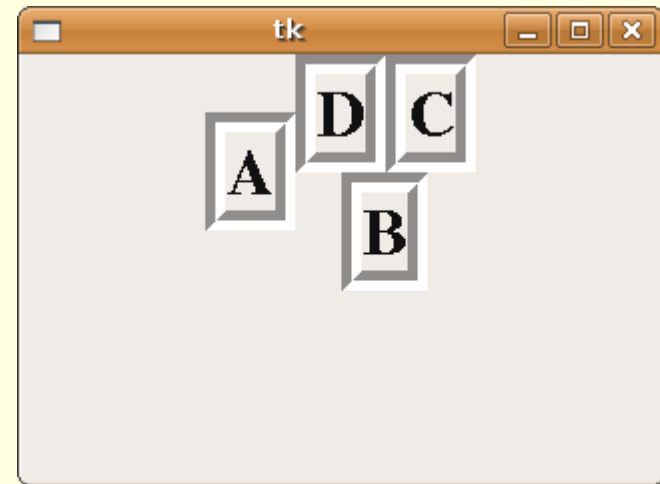
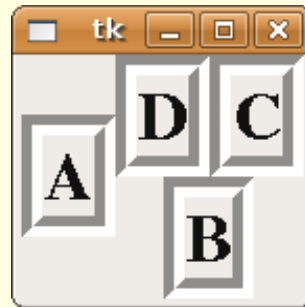
# Usando o *pack*

- 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

```
def exemplo3():
    master=Tk()
    top = Frame(master) ; 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")
    mainloop()

if __name__ == '__main__':
    exemplo3()
```



# 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

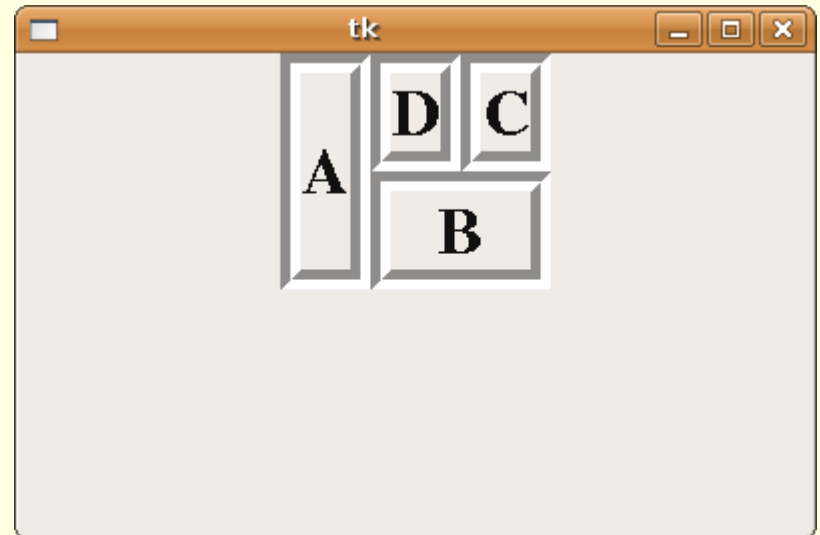
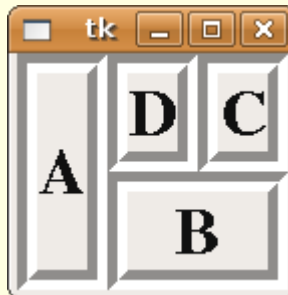
```
def exemplo4():
    master=Tk()
    top = Frame(master) ; top.pack()
    a = Label (top, text="A") ; a.pack (side="left", fill='y')
    b = Label (top, text="B") ; b.pack (side="bottom", fill='x')
    c = Label (top, text="C") ; c.pack (side="right")
    d = Label (top, text="D") ; d.pack (side="top")
    for widget in (a,b,c,d):
        widget.configure(relief="groove", border=10, font="Times 24
bold")
    mainloop()

if __name__ == '__main__':
    exemplo4()
```

# Exemplo

```
def exemplo4():
    master=Tk()
    top = Frame(master) ; 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")
    mainloop()

if __name__ == '__main__':
    exemplo4()
```



# Exemplo

```
def exemplo5():
    master=Tk()
    top = Frame(master) ; 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")
    mainloop()

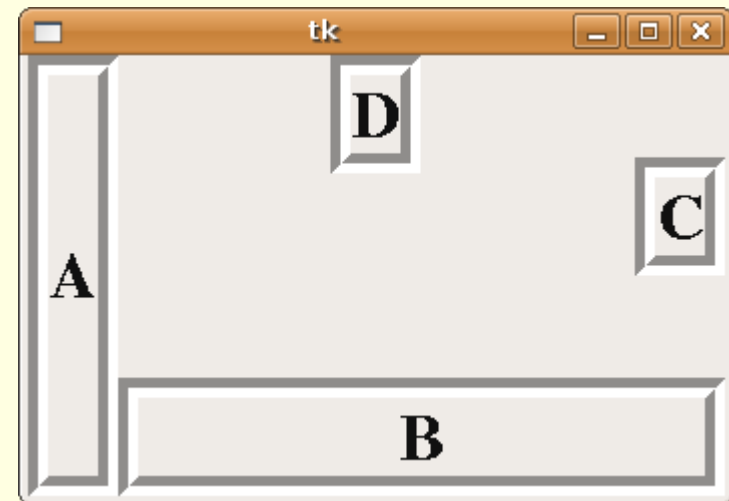
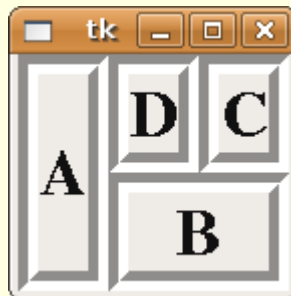
if __name__ == '__main__':
    exemplo5()
```



# Exemplo

```
def exemplo5():
    master=Tk()
    top = Frame(master) ; 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")
    mainloop()

if __name__ == '__main__':
    exemplo5()
```



# Exemplo

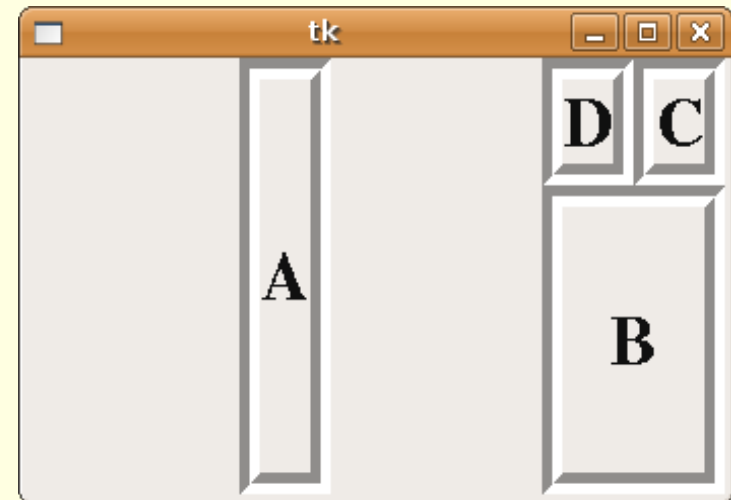
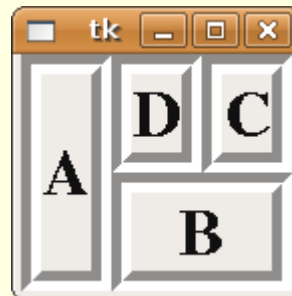
```
def exemplo6():
    master=Tk()
    top = Frame(master) ; top.pack(fill='both', expand=True)
    a = Label (top, text="A") ; a.pack (side="left", expand=True,
fill='y')
    b = Label (top, text="B") ; b.pack (side="bottom", expand=True,
fill='both')
    c = Label (top, text="C") ; c.pack (side="right")
    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")
    mainloop()

if __name__ == '__main__':
    exemplo6()
```

# Exemplo

```
def exemplo6():
    master=Tk()
    top = Frame(master) ; 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")
    mainloop()

if __name__ == '__main__':
    exemplo6()
```



# Usando frames

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

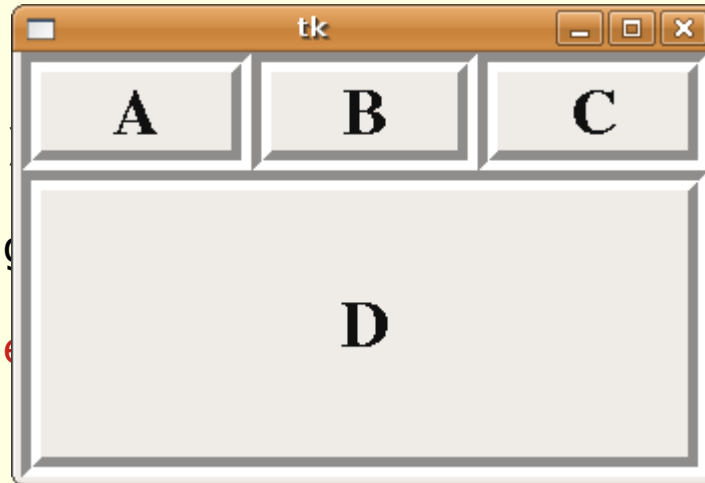
```
def exemplo7():
    master=Tk()
    top = Frame(master) ; top.pack(fill='both', expand=True)
    f = Frame (top); f.pack (fill='x')
    a = Label (f, text="A")
    b = Label (f, text="B")
    c = Label (f, text="C")
    d = Label (top, text="D")
    for w in (a,b,c,d):
        w.configure(relief="groove", border=10, font="Times 24
bold")
        w.pack(side="left", expand=True, fill="both")
    mainloop()

if __name__ == '__main__':
    exemplo7()
```

# Usando frames

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

```
def exemplo7():  
    master=Tk()  
    top = Frame(master) ; 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")  
    d.pack(side="left", expand=True)  
    mainloop()
```



mes 24

```
if __name__ == '__main__':  
    exemplo7()
```

# 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

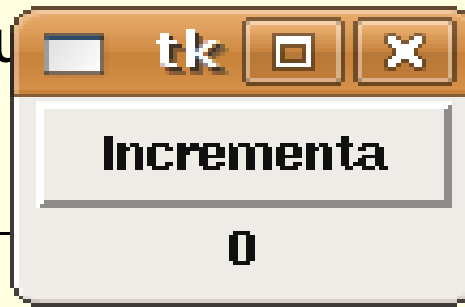
```
def inc():
    n=int(rotulo.cget("text"))+1
    rotulo.configure(text=str(n))

if __name__ == '__main__':
    master=Tk()
    b = Button(master,
text="Incrementa",command=inc)
    b.pack()
    rotulo = Label(master, text="0")
    rotulo.pack()
    mainloop()
```



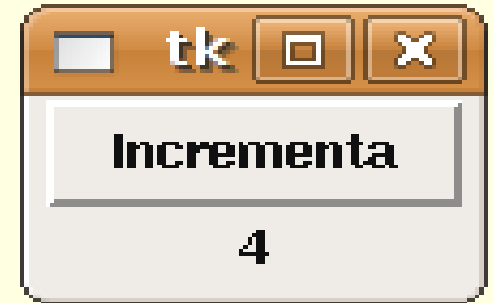
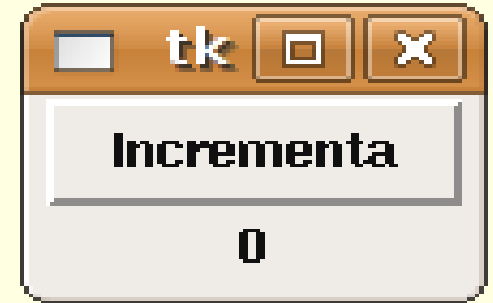
# Exemplo

```
def inc():  
    n=int(rotulo.cget("text"))+1  
    rotulo.configure(text=n))  
  
if __name__ == '__main__':  
    master=Tk()  
    b = Button(master,  
text="Incrementa",command=inc)  
    b.pack()  
    rotulo = Label(master, text="0")  
    rotulo.pack()  
    mainloop()
```



# Exemplo

```
def inc():  
    n=int(rotulo.cget("text"))+1  
    rotulo.configure(text=str(n))  
  
if __name__ == '__main__':  
    master=Tk()  
    b = Button(master,  
text="Incrementa",command=inc)  
    b.pack()  
    rotulo = Label(master, 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

```
def clicca (e):  
    txt = "Mouse clicado em\n%d,%d"%(e.x,e.y)  
    e.widget.configure(text=txt)  
  
if __name__ == '__main__':  
    master=Tk()  
    r = Label(master)  
    r.pack(expand=True, fill="both")  
    master.geometry("200x200")  
    r.bind("<Button-1>", clicca)  
    mainloop()
```

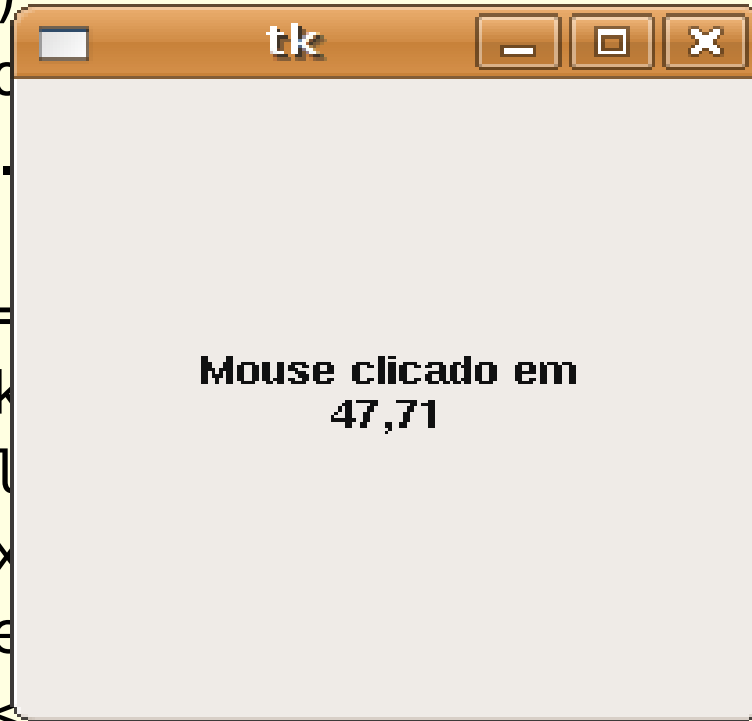
# Exemplo

```
def clicar (e):  
    txt = "Mostrando a posição: " + "%(e.x,e.y)"  
    e.widget.config(text=txt)  
  
if __name__ == '__main__':  
    master = Tk()  
    r = Label(master, text="Clique aqui")  
    r.pack(expand=1)  
    master.geometry("300x100")  
    r.bind("<Button-1", clicar)  
    mainloop()
```



# Exemplo

```
def clicar (e):  
    txt = "Mouse clicado em  
    e.widget.x, e.widget.y"  
    label.config(text=txt)  
    label.place(x=e.x, y=e.y)  
  
if __name__ == '__main__':  
    master = Tk()  
    label = Label(master, text="")  
    label.pack(expand=True)  
    master.geometry("300x300")  
    label.bind("<Button-1", clicar)  
    mainloop()
```



# 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

```
def clica (e):
    txt = "Mouse clicado em\n {0},{1}".format(e.x,e.y)
    e.widget.configure(text=txt)
    e.widget.focus()

def tecla(e):
    txt="Keysym={0}\nKeycode={1}\nChar={2}".format(e.keysym,e.keycode,e.char)
    e.widget.configure(text=txt)

if __name__ == '__main__':
    master=Tk()
    r = Label(master)
    r.pack(expand=True, fill="both")
    master.geometry("200x200")
    r.bind("<Button-1>", clica)
    r.bind("<Key>", tecla)
    mainloop()
```

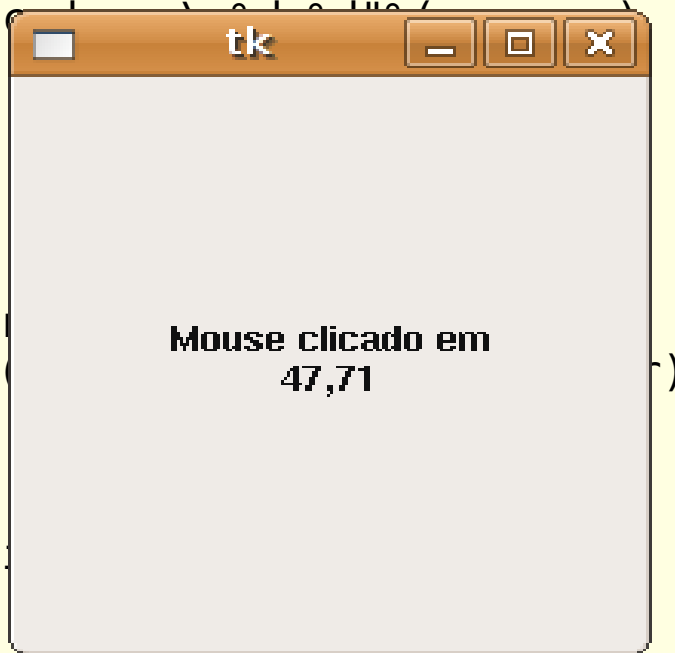
# Exemplo

```
def clicca (e):  
    txt = "Mouse clicado em: " + e.widget.config(cursor) + " no ponto " + e.widget.config(cursorx) + ", " + e.widget.config(cursory) + "  
    e.widget.config(cursor="cross")  
    e.widget.focus()  
  
def tecla(e):  
    txt="Keysym={0}\nChar={2}".format(e.widget.config(cursor), e.widget.config(cursorx), e.widget.config(cursory))  
    e.widget.config(cursor="cross")  
  
if __name__ == '__main__':  
    master=Tk()  
    r = Label(master)  
    r.pack(expand=True, fill="both")  
    master.geometry("200x200")  
    r.bind("<Button-1>", clicca)  
    r.bind("<Key>", tecla)  
    mainloop()
```



# Exemplo

```
def clicca (e):  
    txt = "Mouse clicado em " + str(x) + " e " + str(y) + "  
    e.widget.config(text=txt)  
    e.widget.focus()  
  
def tecla(e):  
    txt="Keysym={0}\nCaracter={1}".format(event.keysym, event.char)  
    e.widget.config(text=txt)  
  
if __name__ == '__main__':  
    master=Tk()  
    r = Label(master)  
    r.pack(expand=True, fill="both")  
    master.geometry("200x200")  
    r.bind("<Button-1>", clicca)  
    r.bind("<Key>", tecla)  
    mainloop()
```



# Exemplo

```
def clicar(e):  
    txt = "Mouse clicado em: " + str(e.x) + "x" + str(e.y)  
    e.widget.config(text=txt)  
    e.widget.focus()  
  
def tecla(e):  
    txt="Keysym={0}\nKeycode={1}\nChar={2}".format(e.keysym, e.keycode, e.char)  
    e.widget.config(text=txt)  
  
if __name__ == '__main__':  
    master=Tk()  
    r = Label(master)  
    r.pack(expand=True, fill="both")  
    master.geometry("200x200")  
    r.bind("<Button-1>", clicar)  
    r.bind("<Key>", tecla)  
    mainloop()
```



# Exemplo

```
def clicar(e):  
    txt = "Mouse clicado em: " + str(e.x) + "x" + str(e.y)  
    e.widget.config(text=txt)  
    e.widget.focus()  
  
def tecla(e):  
    txt="Keysym={0}\nKeycode={1}\nChar={2}".format(e.keysym, e.keycode, e.char)  
    e.widget.config(text=txt)  
  
if __name__ == '__main__':  
    master=Tk()  
    r = Label(master)  
    r.pack(expand=True, fill="both")  
    master.geometry("200x200")  
    r.bind("<Button-1>", clicar)  
    r.bind("<Key>", tecla)  
    mainloop()
```



# 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 (oplevel)

- 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

```
def abrir():  
    print("abrir")  
def salvar():  
    print("salvar")  
def ajuda():  
    print("ajuda")  
def exemplo_menu():  
    master=Tk()  
    principal=Menu(master)  
    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)  
    master.configure(menu=principal)  
    mainloop()  
if __name__ == '__main__':  
    exemplo_menu()
```

# Exemplo

```
def abrir():  
    print("abrir")  
def salvar():  
    print("salvar")  
def ajuda():  
    print("ajuda")  
def exemplo_menu():  
    master=Tk()  
    principal=Menu(m  
    arquivo=Menu(pri  
    arquivo.add_comm  
    arquivo.add_comm  
    principal.add_ca  
    principal.add_co  
    master.configure  
    mainloop()  
if __name__ == '__main__':  
    exemplo_menu()
```



# Exemplo

```
def abrir():  
    print("abrir")  
def salvar():  
    print("salvar")  
def ajuda():  
    print("ajuda")  
def exemplo_menu():  
    master=Tk()  
    principal=Menu(m  
    arquivo=Menu(pri  
    arquivo.add_comm  
    arquivo.add_comm  
    principal.add_ca  
    principal.add_co  
    master.configure  
    mainloop()  
  
if __name__ == '__main__':  
    exemplo_menu()
```



# Menus Popup

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

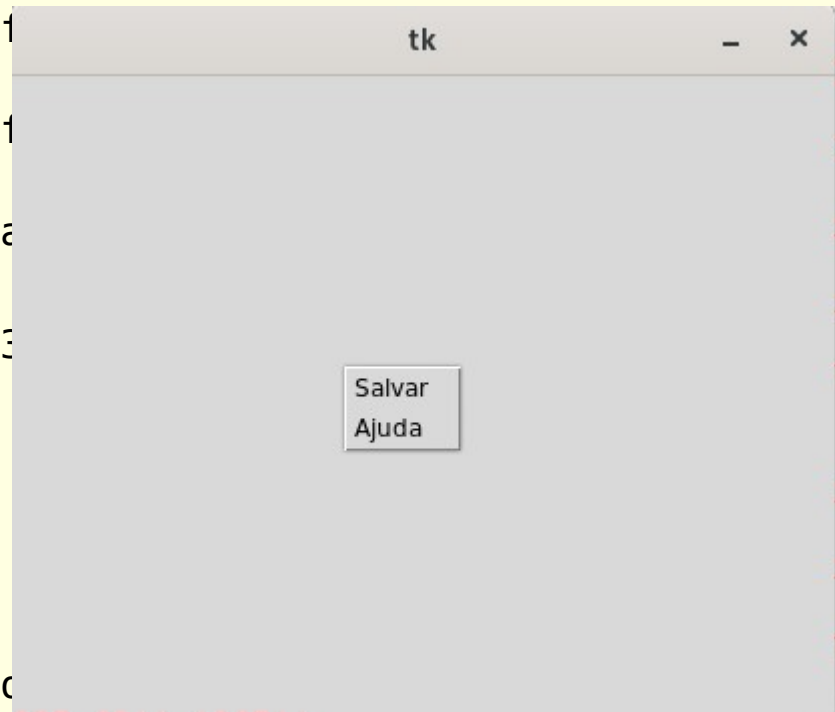
# Exemplo

```
class exemplo_popup(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.menu = Menu(master, tearoff=0)
        self.menu.add_command(self.master, label="Salvar",
command=self.salvar)
        self.menu.add_command(self.master, label="Ajuda",
command=self.ajuda)
        self.frame = Frame(self.master, width=200, height=200)
        self.frame.pack()
        self.frame.bind("<Button-3>", self.popup)
    def salvar(self,):
        print("salvar")
    def ajuda(self,):
        print("ajuda")
    def popup(self, e):
        self.menu.tk_popup(e.x_root, e.y_root)
if __name__ == '__main__':
    master=Tk()
    app = exemplo_popup(master)
    mainloop()
```

# Exemplo

```
class exemplo_popup(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.menu = Menu(master, tearoff=0)
        self.menu.add_command(self)
        command=self.salvar)
        self.menu.add_command(self)
        command=self.ajuda)
        self.frame = Frame(self.ma
        self.frame.pack()
        self.frame.bind("<Button-3
    def salvar(self,):
        print("salvar")
    def ajuda(self,):
        print("ajuda")
    def popup(self, e):
        self.menu.tk_popup(e.x_roo

if __name__ == '__main__':
    master=Tk()
    app = exemplo_popup(master)
    mainloop()
```



# 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

```
class exemplo_entry(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.label = Label(self.master, text='Email').pack()
        self.e = Entry(self.master, font="Arial 24")
        self.i = Button(self.master, text="Insere @",command=self.insere)
        self.l = Button(self.master, text="Limpa",command=self.limpa)
        self.e.pack()
        for w in (self.i,self.l):
            w.pack(side='left')
    def insere(self):
        self.e.insert(INSERT,"@")
    def limpa(self):
        self.e.delete(0,END)

if __name__ == '__main__':
    master=Tk()
    master.geometry("400x300")
    app = exemplo_entry(master)
    mainloop()
```



# Exemplo

```
class exemplo_entry(Frame):  
    def __init__(self, master=None):  
        Frame.__init__(self, master)  
        self.label = Label(self.master, text='Email').pack()  
        self.e = Entry(self.master, font="Arial 24")
```



```
        self.insere()  
        self.limpa()
```

```
def
```

```
def
```

```
        self.e.delete(0,END)
```

```
if __name__ == '__main__':  
    master=Tk()  
    master.geometry("400x300")  
    app = exemplo_entry(master)  
    mainloop()
```

# Exemplo

```
class exemplo_entry(Frame):  
    def __init__(self, master=None):  
        Frame.__init__(self, master)  
        self.label = Label(self.master, text='Email').pack()  
        self.e = Entry(self.master, font="Arial 24")
```



```
f.insere)  
impa)
```

```
def
```

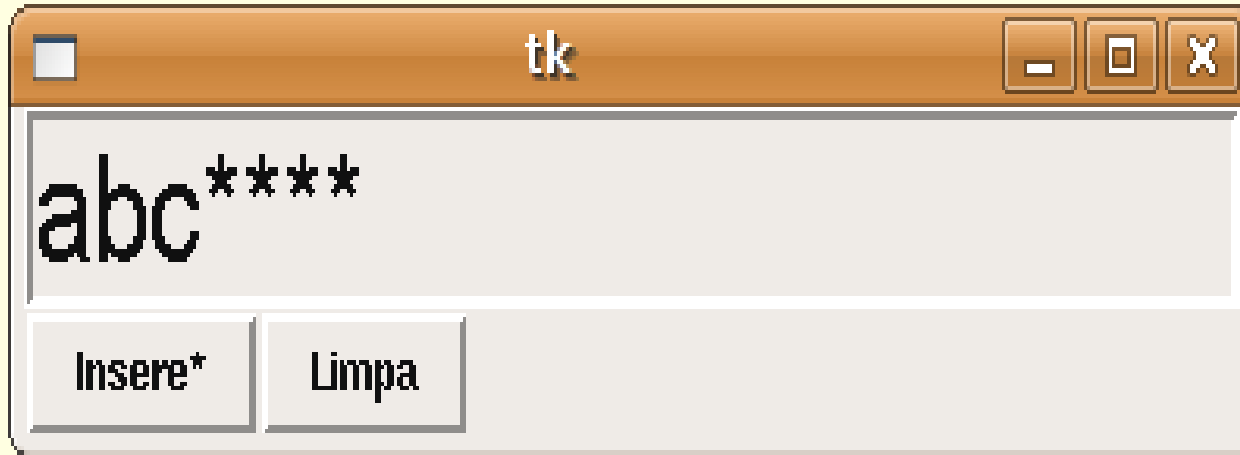
```
def
```

```
self.e.delete(0,END)
```

```
if __name__ == '__main__':  
    master=Tk()  
    master.geometry("400x300")  
    app = exemplo_entry(master)  
    mainloop()
```

# Exemplo

```
class exemplo_entry(Frame):  
    def __init__(self, master=None):  
        Frame.__init__(self, master)  
        self.label = Label(self.master, text='Email').pack()  
        self.e = Entry(self.master, font="Arial 24")
```



```
f.insere)  
impa)
```

```
def
```

```
def
```

```
self.e.delete(0,END)
```

```
if __name__ == '__main__':  
    master=Tk()  
    master.geometry("400x300")  
    app = exemplo_entry(master)  
    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

```
class exemplo_var(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.soma = DoubleVar(self.master)
        self.parcela = DoubleVar(self.master)
        self.lsoma = Label(self.master, textvar=self.soma)
        self.eparcela = Entry(self.master, textvar=self.parcela)
        self.eparcela.bind("<Return>", self.aritmetica)
        self.eparcela.bind("<KP_Enter>", self.aritmetica)
        self.lsoma.pack()
        self.eparcela.pack()

    def aritmetica(self, e):
        self.soma.set(self.soma.get() + self.parcela.get())

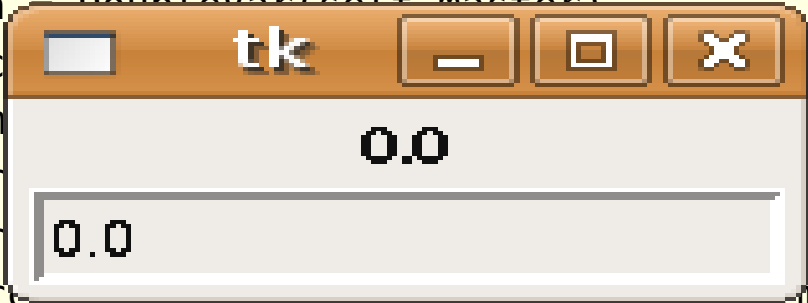
if __name__ == '__main__':
    master=Tk()
    app = exemplo_var(master)
    mainloop()
```

# Exemplo

```
class exemplo_var(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.soma = DoubleVar(self, master)
        self.parcela = DoubleVar(self, master)
        self.lsoma = Label(self, text=self.soma.get())
        self.eparcela = Entry(self, text=self.parcela.get())
        self.eparcela.bind('<Return>', self.aritmetica)
        self.lsoma.pack()
        self.eparcela.pack()

    def aritmetica(self, e):
        self.soma.set(self.soma.get() + self.parcela.get())

if __name__ == '__main__':
    master=Tk()
    app = exemplo_var(master)
    mainloop()
```

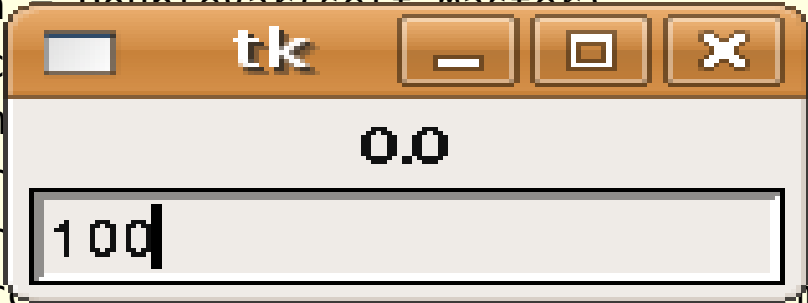


# Exemplo

```
class exemplo_var(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.soma = DoubleVar(self, master)
        self.parcela = DoubleVar(self, master)
        self.lsoma = Label(self, text=self.soma.get())
        self.eparcela = Entry(self, text=self.parcela.get())
        self.eparcela.bind('<Return>', self.aritmetica)
        self.lsoma.pack()
        self.eparcela.pack()

    def aritmetica(self, e):
        self.soma.set(self.soma.get() + self.parcela.get())

if __name__ == '__main__':
    master=Tk()
    app = exemplo_var(master)
    mainloop()
```




# Exemplo

```
class exemplo_var(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.soma = DoubleVar(self, master)
        self.parcela = DoubleVar(self, master)
        self.lsoma = Label(self, text=self.soma.get())
        self.eparcela = Entry(self, text=self.parcela.get())
        self.eparcela.bind('<Return>', self.aritmetica)
        self.lsoma.pack()
        self.eparcela.pack()

    def aritmetica(self, e):
        self.soma.set(self.soma.get() + self.parcela.get())

if __name__ == '__main__':
    master=Tk()
    app = exemplo_var(master)
    mainloop()
```





# 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

```
class exemplo_check(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.v1 = IntVar(self.master, value=0)
        self.v2 = StringVar(self.master, value="Nao")
        self.c1 = Checkbutton (self.master, text="V1", var=self.v1, \
                                command=self.exibe)
        self.c2 = Checkbutton (self.master, text="V2", var=self.v2, \
                                command=self.exibe, onvalue="Sim", \
                                offvalue="Nao")

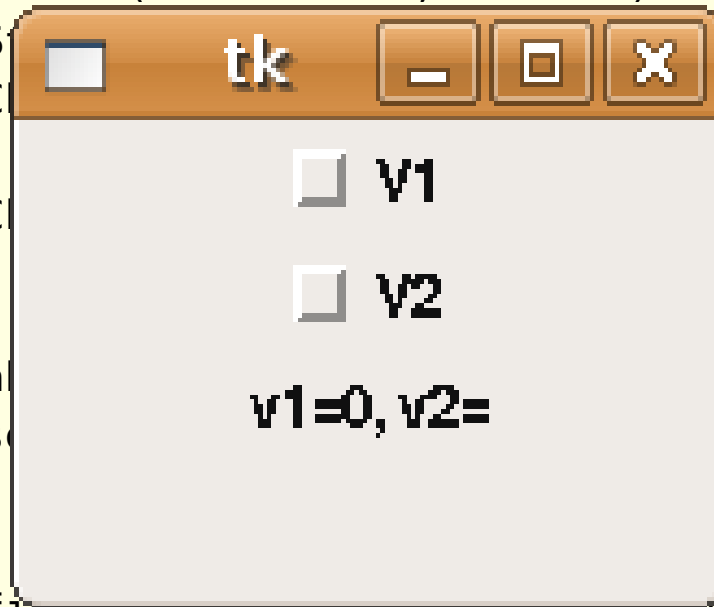
        self.l = Label(self.master)
        for w in (self.c1, self.c2, self.l):
            w.pack()
    def exibe(self):
        self.l.config(text="v1={0}, v2={1}".format(self.v1.get(), \
                                                    self.v2.get()))

if __name__ == '__main__':
    master = Tk()
    master.geometry("400x300")
    app = exemplo_check(master)
    mainloop()
```

# Exemplo

```
class exemplo_check(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.v1 = IntVar(self.master, value=0)
        self.v2 = StringVar(self.master, value="")
        self.c1 = Checkbutton(self, text="V1", var=self.v1, \
                                command=self.exibe)
        self.c2 = Checkbutton(self, text="V2", var=self.v2, \
                                command=self.exibe)
        self.l = Label(self, text="v1=0, v2=")
        for w in (self.c1, self.c2, self.l):
            w.pack()
        def exibe(self):
            self.l.config(text="v1={0}, v2={1}".format(self.v1.get(), \
                                                         self.v2.get()))

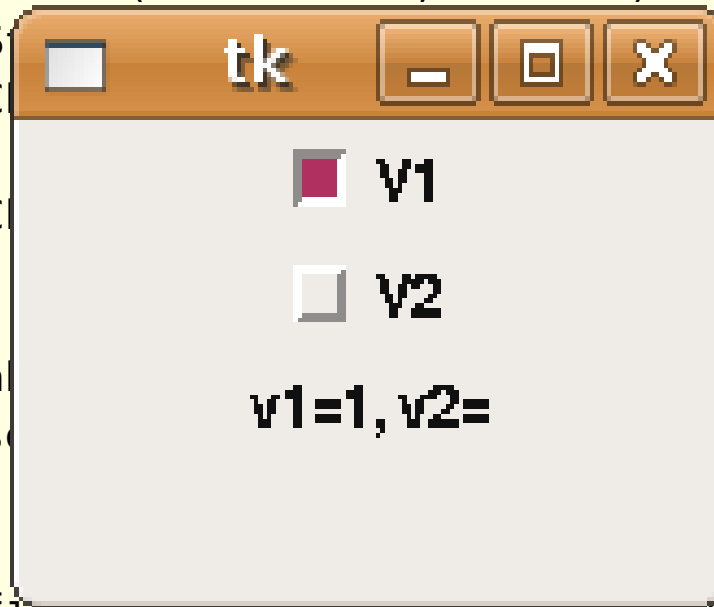
if __name__ == '__main__':
    master=Tk()
    master.geometry("400x300")
    app = exemplo_check(master)
    mainloop()
```



# Exemplo

```
class exemplo_check(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.v1 = IntVar(self.master, value=0)
        self.v2 = StringVar(self.master, value="")
        self.c1 = Checkbutton(self, text="V1", var=self.v1, \
                                command=self.exibe)
        self.c2 = Checkbutton(self, text="V2", var=self.v2, \
                                command=self.exibe)
        self.l = Label(self, text="")
        for w in (self.c1, self.c2, self.l):
            w.pack()
        def exibe(self):
            self.l.config(text="v1={0}, v2={1}".format(self.v1.get(), \
                                                         self.v2.get()))
        self.exibe()

if __name__ == '__main__':
    master=Tk()
    master.geometry("400x300")
    app = exemplo_check(master)
    mainloop()
```



# Exemplo

```
class exemplo_check(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.v1 = IntVar(self.master, value=0)
        self.v2 = StringVar(self.master, value="")
        self.c1 = Checkbutton(self, text="V1", var=self.v1, \
                                command=self.exibe)
        self.c2 = Checkbutton(self, text="V2", var=self.v2, \
                                command=self.exibe)

        self.l = Label(self, text="")
        for w in (self.c1, self.c2, self.l):
            w.pack()

    def exibe(self):
        self.l.config(text="v1={0}, v2={1}".format(self.v1.get(), \
                                                    self.v2.get()))

if __name__ == '__main__':
    master=Tk()
    master.geometry("400x300")
    app = exemplo_check(master)
    mainloop()
```



# 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

```
class exemplo_radio(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.cor = StringVar(self.master, value="black")
        self.l = Label(self.master, background=self.cor.get())
        self.l.pack(fill='both', expand=True)
        self.cores = [("preto", "black"), ("vermelho", "red"), \
                      ("azul", "blue"), ("verde", "green")]
        for txt, val in self.cores:
            Radiobutton(text=txt, value=val, variable=self.cor, \
                        command=self.pinta).pack(anchor=W)

    def pinta(self):
        self.l.configure(background=self.cor.get())

if __name__ == '__main__':
    master=Tk()
    master.geometry("400x300")
    app = exemplo_radio(master)
    mainloop()
```

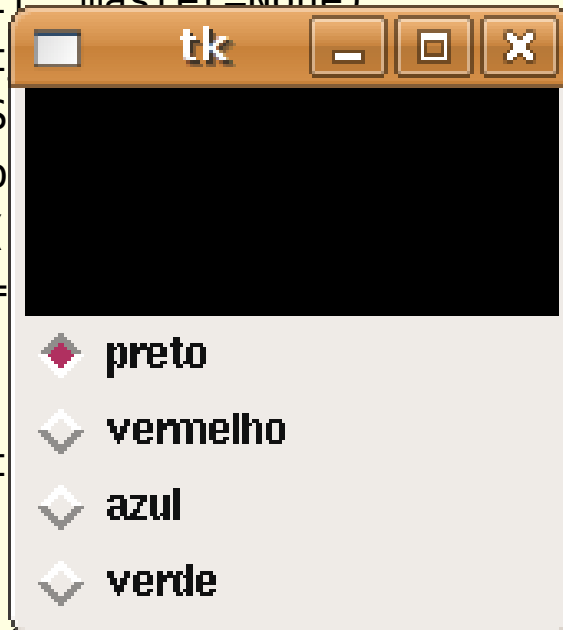
# Exemplo

```
class exemplo_radio(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.cor = S
        self.l = Lab
        self.l.pack(
        self.cores =

        for txt,val
            Radiobut

    def pinta(self):
        self.l.configure(background=self.cor.get())

if __name__ == '__main__':
    master=Tk()
    master.geometry("400x300")
    app = exemplo_radio(master)
    mainloop()
```





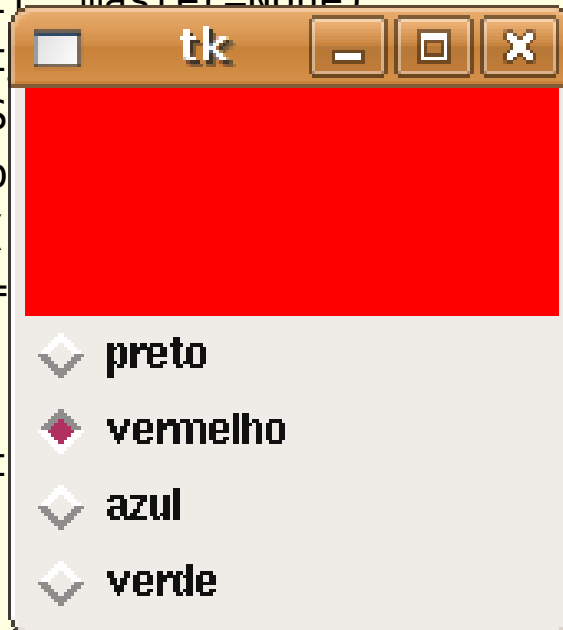
# Exemplo

```
class exemplo_radio(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.cor = S
        self.l = Lab
        self.l.pack(
        self.cores =

        for txt,val
            Radiobut

    def pinta(self):
        self.l.configure(background=self.cor.get())

if __name__ == '__main__':
    master=Tk()
    master.geometry("400x300")
    app = exemplo_radio(master)
    mainloop()
```



# Exemplo

```
class exemplo_radio(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.cor = S
        self.l = Lab
        self.l.pack(
        self.cores =

        for txt,val
            Radiobut

    def pinta(self):
        self.l.configure(background=self.cor.get())

if __name__ == '__main__':
    master=Tk()
    master.geometry("400x300")
    app = exemplo_radio(master)
    mainloop()
```



# Exemplo

```
class exemplo_radio(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.cor = StringVar(self.master, value="black")
        self.l = Label(self.master, background=self.cor.get())
        self.l.pack(fill='both', expand=True)
        self.cores = [("preto", "black"), ("vermelho", "red"), \
                      ("azul", "blue"), ("verde", "green")]
        for txt, val in self.cores:
            Radiobutton(text=txt, value=val, variable=self.cor, \
                        command=self.pinta).pack(anchor=W)

    def pinta(self):
        self.l.configure(background=self.cor.get())

if __name__ == '__main__':
    master=Tk()
    master.geometry("400x300")
    app = exemplo_radio(master)
    mainloop()
```

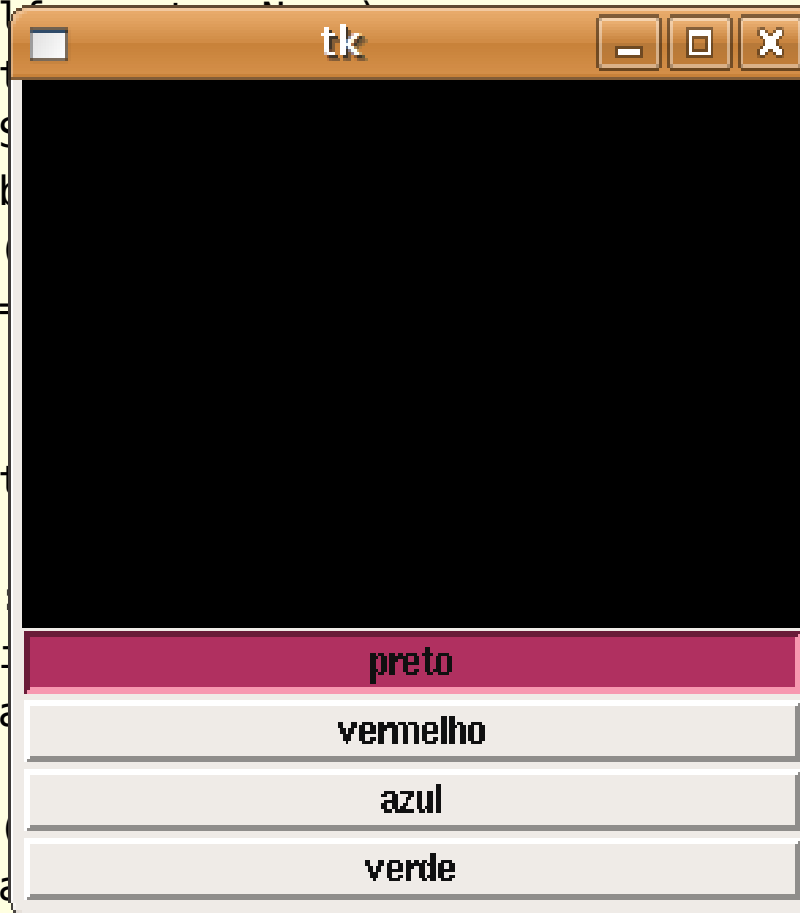
# Exemplo

```
class exemplo_radio(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.cor = 'preto'
        self.l = Label(self, text="Cor: ")
        self.l.pack()
        self.cores = ['preto', 'vermelho', 'azul', 'verde']

        for txt, val in zip(self.cores, self.cores):
            Radiobutton(self, text=txt, value=val, command=self.pinta)

        def pinta(self):
            self.l.config(text=f"Cor: {self.cor}")

if __name__ == '__main__':
    master = Tk()
    master.geometry("300x200")
    app = exemplo_radio(master)
    app.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 PIL import Image, ImageTk
class exemplo_canvas(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.c = Canvas(self.master)
        self.c.pack(expand=True, fill='both')
        self.c.create_oval(1,1,200,100,outline="blue", width=5,fill="red")
        self.button = Button(text="Tk Canvas")
        self.c.create_window(10,120,window=self.button,anchor=W)
        self.c.create_line(100,0,120,30,50,60,100,120,\
                           fill="black",width=2)
        self.c.create_rectangle(40,150,100,200,fill="white")
        img = Image.open("canvas1.jpg").resize((300,300), Image.ANTIALIAS)
        self.img = ImageTk.PhotoImage(img)
        self.c.create_image(200,200,image=self.img,anchor=NW)
        self.c.create_arc (150,90,250,190,start=30,extent=60,\
                           outline="green",fill="orange")
        self.c.create_text(200,35,text="Texto\nTexto", font="Arial 22")
if __name__ == '__main__':
    master=Tk()
    master.geometry("640x480")
    app = exemplo_canvas(master)
    mainloop()
```

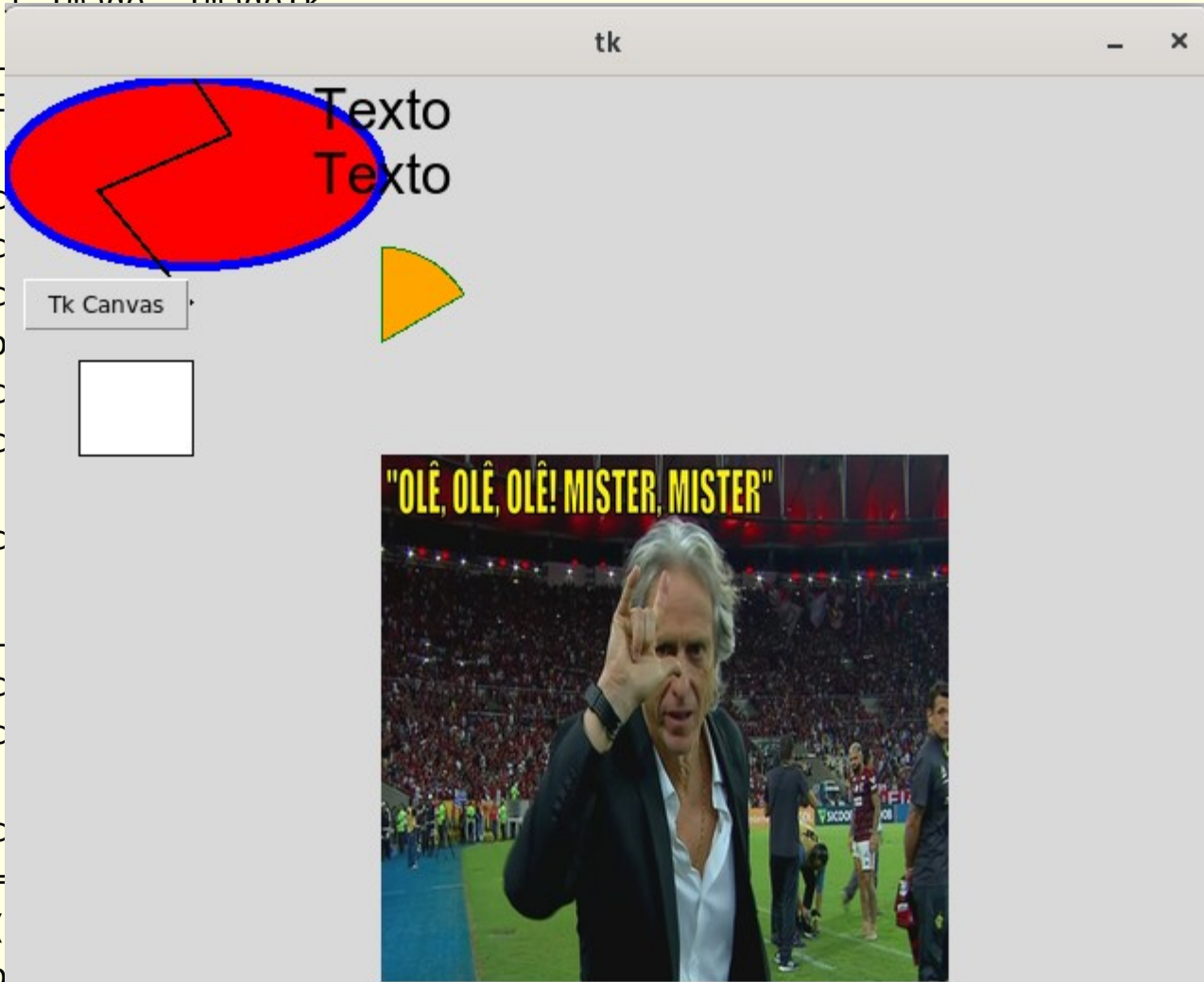
# Exemplo

```
from PIL import Image, ImageTk
class exemplo_canvas:
    def __init__(self, master):
        self.canvas = Canvas(master)
        self.canvas.pack()

        img = Image.open('image.png')
        self.image = ImageTk.PhotoImage(img)
        self.canvas.create_image(0, 0, anchor='nw', image=self.image)

        self.canvas.mainloop()

if __name__ == '__main__':
    master=Tk()
    master.geometry('600x600')
    app = exemplo_canvas(master)
    mainloop()
```



# 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
  - O parâmetro `scrollregion` configura as coordenadas `x0,y0` e `x1,y1` até onde o canvas pode ser rolado
  - 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

```
class exemplo_canvas2(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.c = Canvas(self.master)
        self.c.pack(expand=True, fill='both')
        self.c.bind("<Button-1>", self.novalinha)
        self.c.bind("<B1-Motion>", self.estendelinha)
        self.c.bind("<ButtonRelease-1>", self.fechalinha)
        self.b = Button(self.master, text="limpar", command=self.limpa)
        self.b.pack()

    def novalinha(self,e):
        x,y = self.c.canvasx(e.x), self.c.canvasy(e.y)
        self.c.create_line(x,y,x,y,tags="corrente")

    def estendelinha(self,e):
        x,y = self.c.canvasx(e.x), self.c.canvasy(e.y)
        coords = self.c.coords("corrente") + [x,y]
        self.c.coords("corrente", coords)

    def fechalinha(self,e):
        self.c.itemconfig("corrente",tags=())

    def limpa(self):
        self.c.delete(ALL)

if __name__ == '__main__':
    master=Tk()
    master.geometry("640x480")
    app = exemplo_canvas2(master)
    mainloop()
```

# Exemplo

```
class exemplo_canvas2(Frame):  
    def __init__(self, master=None):  
        Frame.__init__(self, master)
```

```
        self.c = Ca
```

```
        self.c.pack
```

```
        self.c.bind
```

```
        self.c.bind
```

```
        self.c.bind
```

```
        self.b = Bu
```

```
        self.b.pack
```

```
    def novalinha(s
```

```
        x,y = self.
```

```
        self.c.crea
```

```
    def estendelinh
```

```
        x,y = self.
```

```
        coords = se
```

```
        self.c.coor
```

```
    def fechalinha(
```

```
        self.c.item
```

```
    def limpa(self)
```

```
        self.c.dele
```

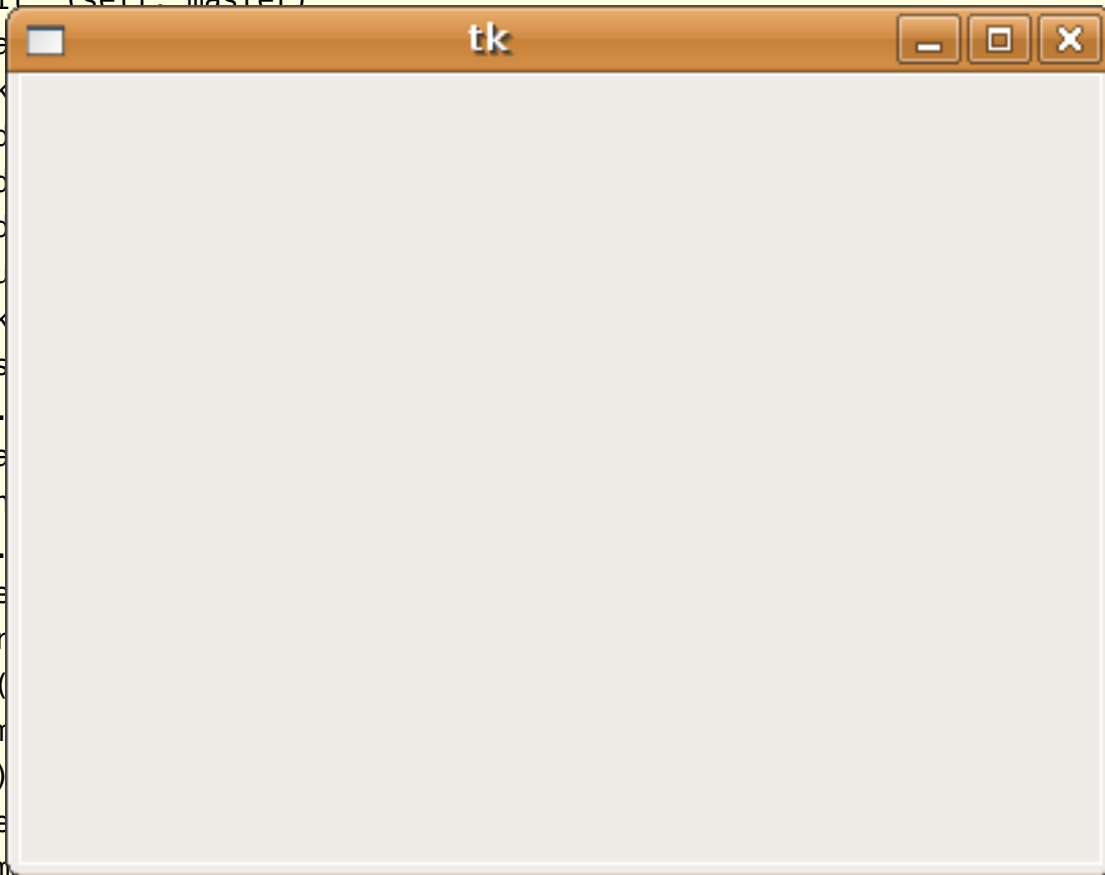
```
if __name__ == '__m
```

```
    master=Tk()
```

```
    master.geometry("640x480")
```

```
    app = exemplo_canvas2(master)
```

```
    mainloop()
```



# Exemplo

```
class exemplo_canvas2(Frame):  
    def __init__(self, master=None):  
        Frame.__init__(self, master)
```

```
        self.c = Ca
```

```
        self.c.pack
```

```
        self.c.bind
```

```
        self.c.bind
```

```
        self.c.bind
```

```
        self.b = Bu
```

```
        self.b.pack
```

```
    def novalinha(s
```

```
        x,y = self.
```

```
        self.c.crea
```

```
    def estendelinh
```

```
        x,y = self.
```

```
        coords = se
```

```
        self.c.coor
```

```
    def fechalinha(
```

```
        self.c.item
```

```
    def limpa(self)
```

```
        self.c.dele
```

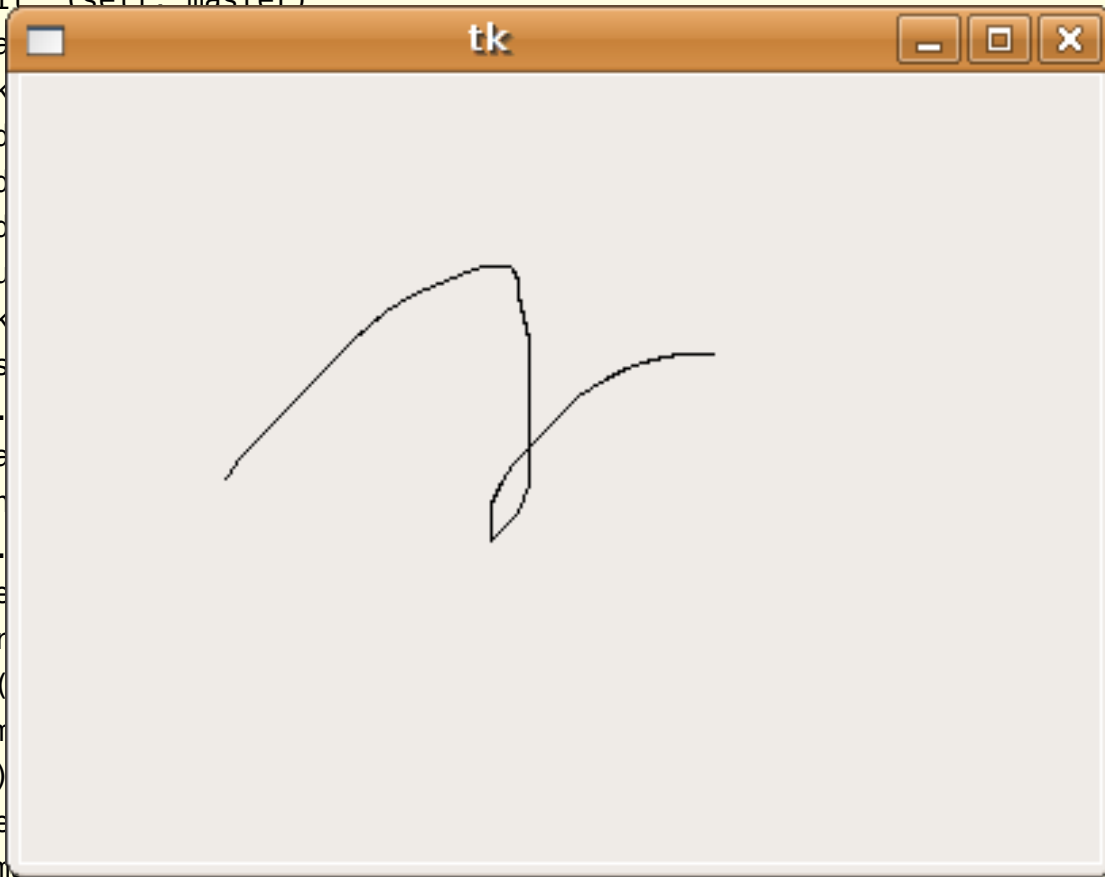
```
if __name__ == '__m
```

```
    master=Tk()
```

```
    master.geometry("640x480")
```

```
    app = exemplo_canvas2(master)
```

```
    mainloop()
```



# Exemplo

```
class exemplo_canvas2(Frame):
```

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

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

```
        self.c = Ca
```

```
        self.c.pack
```

```
        self.c.bind
```

```
        self.c.bind
```

```
        self.c.bind
```

```
        self.b = Bu
```

```
        self.b.pack
```

```
    def novalinha(s
```

```
        x,y = self.
```

```
        self.c.crea
```

```
    def estendelinh
```

```
        x,y = self.
```

```
        coords = se
```

```
        self.c.coor
```

```
    def fechalinha(
```

```
        self.c.item
```

```
    def limpa(self)
```

```
        self.c.dele
```

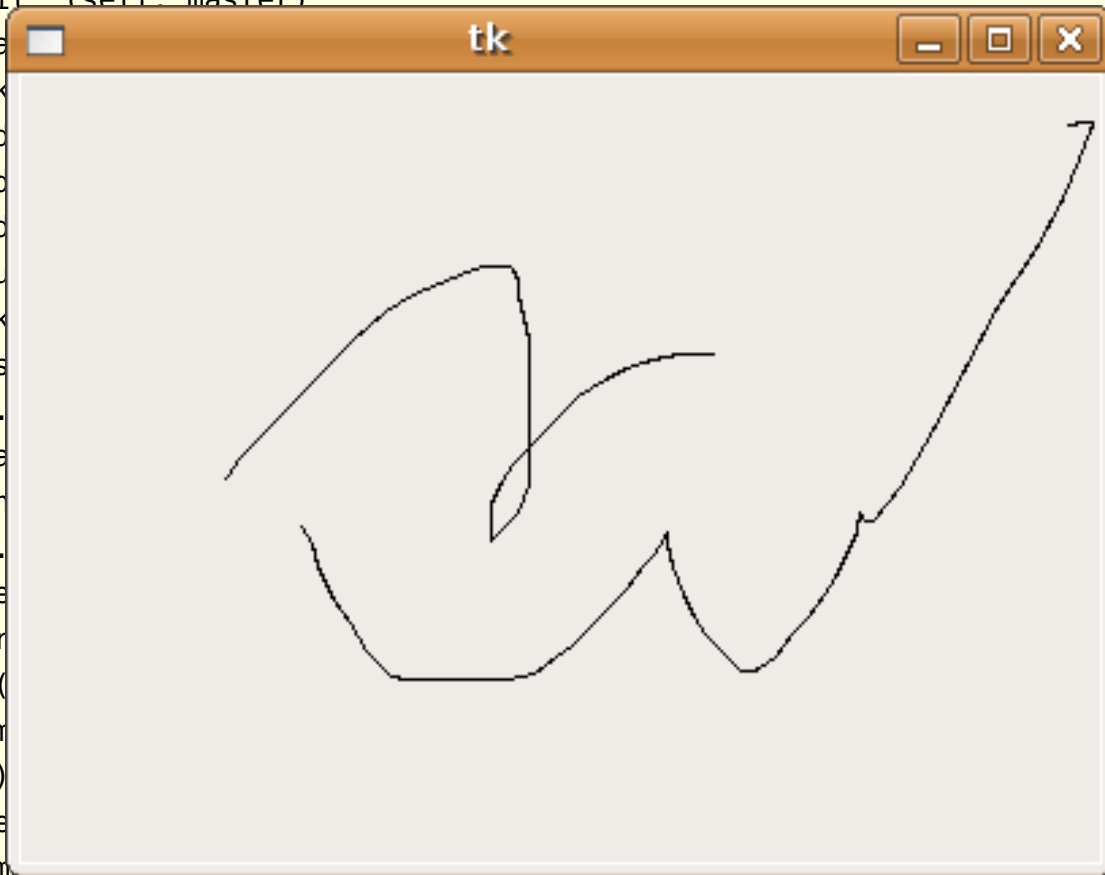
```
if __name__ == '__m
```

```
    master=Tk()
```

```
    master.geometry("640x480")
```

```
    app = exemplo_canvas2(master)
```

```
    mainloop()
```



# Exemplo

```
class exemplo_canvas2(Frame):
    def __init__(self, master=None):
        ...
        self.c.bind("<Button-3>", self.selecionalinha)
        self.c.bind("<B3-Motion>", self.movelinha)
        self.c.bind("<ButtonRelease-3>", self.deselecionalinha)
        self.x0, self.y0 = self.c.canvasx(0), self.c.canvasy(0)
        ...
    def selecionalinha(self, e):
        self.x0, self.y0 = self.c.canvasx(e.x), self.c.canvasy(e.y)
        self.c.itemconfig(CURRENT, tags="sel")
    def movelinha (self, e):
        x1, y1 = self.c.canvasx(e.x), self.c.canvasy(e.y)
        self.c.move("sel", x1-self.x0, y1-self.y0)
        self.x0, self.y0 = x1, y1
    def deselecionalinha(self, e):
        self.c.itemconfig("sel", tags=())
if __name__ == '__main__':
    master=Tk()
    master.geometry("640x480")
    app = exemplo_canvas2(master)
    mainloop()
```

# Exemplo

```
class exemplo_canvas2(Frame):  
    def __init__(self, master=None):
```

```
        ...  
        self.c.  
        self.c.  
        self.c.  
        self.x0
```

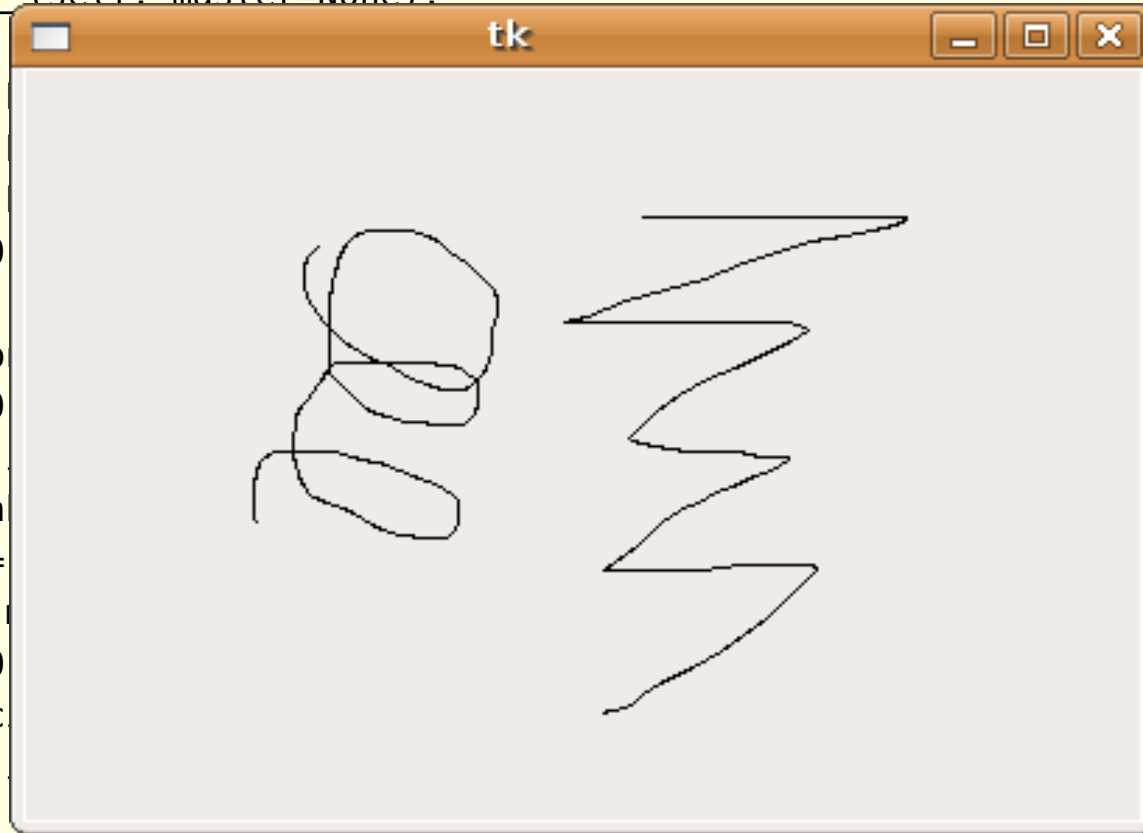
```
        ...
```

```
    def selecio  
        self.x0  
        self.c.
```

```
    def movelin  
        x1,y1 =  
        self.c.  
        self.x0
```

```
    def deselec  
        self.c.
```

```
if __name__ ==  
    master=Tk()  
    master.geometry("640x480")  
    app = exemplo_canvas2(master)  
    mainloop()
```





# Exemplo

```
class exemplo_canvas2(Frame):  
    def __init__(self, master=None):
```

```
        ...  
        self.c.  
        self.c.  
        self.c.  
        self.x0
```

```
        ...
```

```
    def selecio  
        self.x0  
        self.c.
```

```
    def movelin  
        x1,y1 =  
        self.c.  
        self.x0
```

```
    def deselec  
        self.c.
```

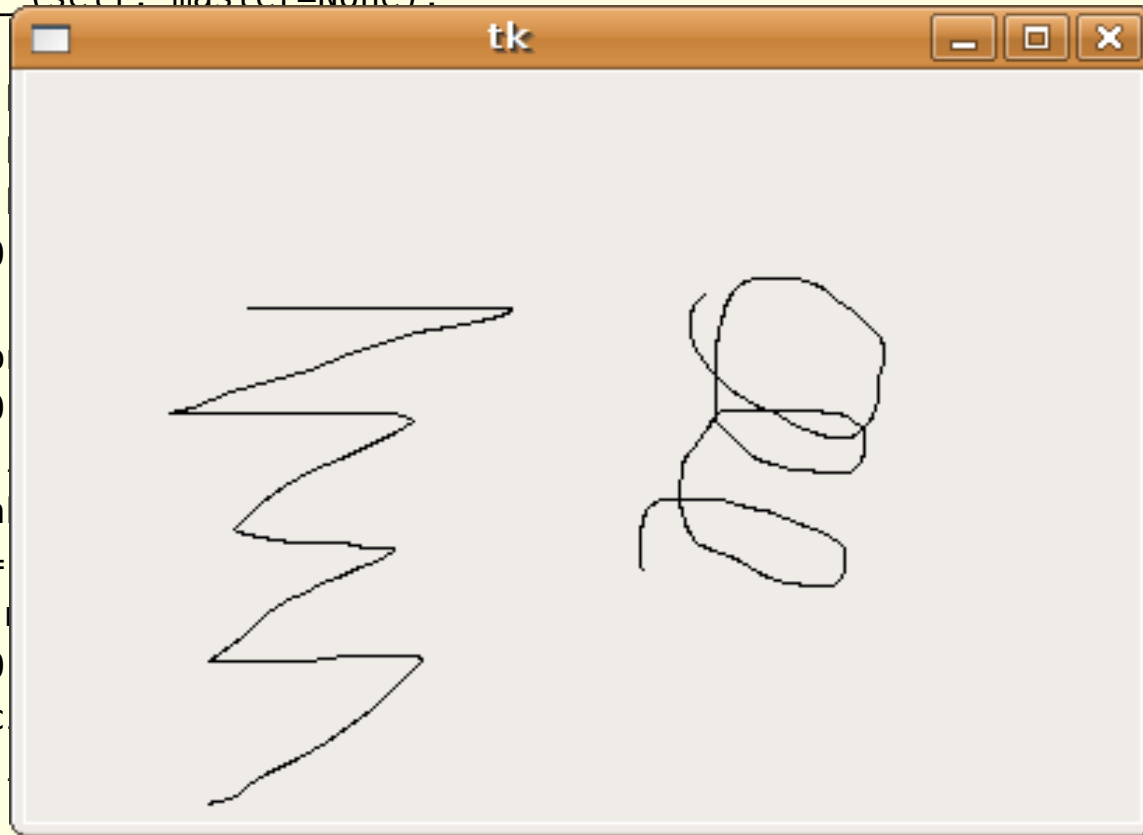
```
if __name__ ==
```

```
    master=Tk()
```

```
    master.geometry("640x480")
```

```
    app = exemplo_canvas2(master)
```

```
    mainloop()
```



# 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

```
class exemplo_scrollcanvas(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.c = Canvas(self.master, width=1000, height=1000)
        self.c.configure(scrollregion=(0,0,1000,1000))
        self.c.pack(expand=True, fill='both')
        self.sby = Scrollbar(self.c, command=self.c.yview, orient=VERTICAL)
        self.sbx = Scrollbar(self.c, command=self.c.xview, orient=HORIZONTAL)
        self.sby.pack(side=RIGHT, fill='y')
        self.sbx.pack(side=BOTTOM, fill='x')
        self.c.configure(yscrollcommand=self.sby.set)
        self.c.configure(xscrollcommand=self.sbx.set)
        img = Image.open("canvas2.jpg")
        self.img = ImageTk.PhotoImage(img)
        self.c.create_image(0,0,image=self.img,anchor=NW)

if __name__ == '__main__':
    master=Tk()
    master.geometry("640x480")
    app = exemplo_scrollcanvas(master)
    mainloop()
```

# Exemplo

```
class exemplo_scrollcanvas:
    def __init__(self, master):
        Frame.__init__(self, master)
        self.c = Canvas(self, width=400, height=400)
        self.c.config(bg='yellow')
        self.c.pack()
        self.sby = Scrollbar(self, command=self.c.yview)
        self.sbx = Scrollbar(self, command=self.c.xview)
        self.sby.pack(side='right', fill='y')
        self.sbx.pack(side='left', fill='x')
        self.c.config(scrollregion=(0,0,400,400))
        img = Image.open('gabi.jpg')
        self.img = img
        self.c.create_image(200, 200, image=self.img)

if __name__ == '__main__':
    master = Tk()
    master.geometry('640x480')
    app = exemplo_scrollcanvas(master)
    mainloop()
```



\_)  
(AL)