# Interface Graphique

# Tkinter Python

Tkinter est un module de base intégré dans Python , normalement vous n'avez rien à faire pour pouvoir l'utiliser. L'un des avantages de Tkinter est sa portabilité sur les OS les plus utilisés par le grand public.

## Installer Tkinter

Tkinter est installé par défaut, si ce n'est pas le cas, lancez la commande suivante:

```
sudo apt-get install python-tk
```

En python 3:

```
sudo apt-get install python3-tk
```

## Python 2, python 3

Les modules ne sont pas les mêmes suivant votre version de python. Si le message suivant apparaît lors de l'exécution de votre script:

```
ImportError: No module named 'Tkinter'
```

C'est que le module appelé n'est pas le bon par rapport à votre version python.

| Python 2 | | Python 3 |
|---|---|---|
| Tkinter | → | tkinter |
| Tix | → | tkinter.tix |
| ttk | → | tkinter.ttk |
| tkMessageBox | → | tkinter.messagebox |
| tkColorChooser | → | tkinter.colorchooser |
| tkFileDialog | → | tkinter.filedialog |
| tkCommonDialog | → | tkinter.commondialog |
| tkSimpleDialog | → | tkinter.simpledialog |



Scanné avec CamScanner

```
tkFont          → tkinter.font
Tkdnd           → tkinter.dnd
ScrolledText    → tkinter.scrolledtext
```

# Hello world

Voici le code de votre premier **hello world**

```
# coding: utf-8
from tkinter import *
fenetre = Tk()
label = Label(fenetre, text="Hello World")
label.pack()
fenetre.mainloop()
```

Une fenêtre comme celle-ci devrait apparaître:



Hello World

## Les widget Tkinter

Pour créer un logiciel graphique vous devez ajouter dans une fenêtre des éléments graphiques que l'on nomme widget. Ce widget peut être tout aussi bien une liste déroulante que du texte.

## Les boutons

Les boutons permettent de proposer une action à l'utilisateur. Dans l'exemple ci-dessous, on lui propose de fermer la fenêtre.

```
# bouton de sortie
bouton=Button(fenetre, text="Fermer", command=fenetre.quit)
bouton.pack()
```



Fermer

# Les labels



Scanné avec CamScanner

Les labels sont des espaces prévus pour écrire du texte. Les labels servent souvent à décrire un widget comme un input

```
# Label

label = Label(fenetre, text="Texte par défaut", bg="yellow")

label.pack()
```




# Entrée / input

```
# entrée

value = StringVar()

value.set("texte par défaut")

entree = Entry(fenetre, textvariable=string, width=30)

entree.pack()
```




# Case à cocher

Les checkbox proposent à l'utilisateur de cocher une option.

```
# checkbutton

bouton = Checkbutton(fenetre, text="Nouveau?")

bouton.pack()
```




## Boutons radio

Les boutons radio sont des cases à cocher qui sont dans un groupe et dans ce groupe seul un élément peut être sélectionné.

```
# radiobutton
```



Scanné avec CamScanner

```
value = StringVar()

bouton1 = Radiobutton(fenetre, text="Oui", variable=value, value=1)

bouton2 = Radiobutton(fenetre, text="Non", variable=value, value=2)

bouton3 = Radiobutton(fenetre, text="Peu être", variable=value, value=3)

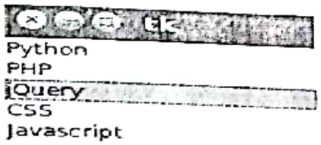
bouton1.pack()

bouton2.pack()

bouton3.pack()
```

## Les listes

Les listes permettent de récupérer une valeur sélectionnée par l'utilisateur.




```
# Liste

liste = Listbox(fenetre)

liste.insert(1, "Python")

liste.insert(2, "PHP")

liste.insert(3, "jQuery")

liste.insert(4, "CSS")

liste.insert(5, "Javascript")

liste.pack()
```

## Canvas

Un canvas (toile. tableau en français) est un espace dans lequel vous pouvez dessiner ou écrire ce que vous voulez:

```
# canvas

canvas = Canvas(fenetre, width=150, height=120, background='yellow')

ligne1 = canvas.create_line(75, 0, 75, 120)
```

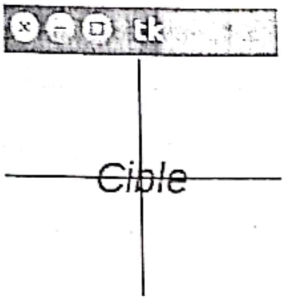


Scanné avec CamScanner

```
ligne2 = canvas.create_line(0, 60, 150, 60)

txt = canvas.create_text(75, 60, text="Cible", font="Arial 15 italic", fill="blue")

canvas.pack()
```




Vous pouvez créer d'autres éléments:

```
create_arc()        :  arc de cercle

create_bitmap()     :  bitmap

create_image()      :  image

create_line()       :  ligne

create_oval()       :  ovale

create_polygon()    :  polygone

create_rectangle()  :  rectangle

create_text()       :  texte

create_window()     :  fenetre
```

Si vous voulez changer les coordonnées d'un élement crée dans le canevas, vous pouvez utiliser la méthode **coords** .

```
canvas.coords(élément, x0, y0, x1, y1)
```

Pour supprimer un élément vous pouvez utiliser la méthode delete

```
canvas.delete(élément)
```

Vous pouvez trouver d'autres méthodes utiles en exécutant l'instruction suivante:

```
print dir(Canvas())
```

## Scale

Le widget scale permet de récupérer une valeur numérique via un scroll



Scanné avec CamScanner

```
value = DoubleVar()
scale = Scale(fenetre, variable=value)
scale.pack()
```



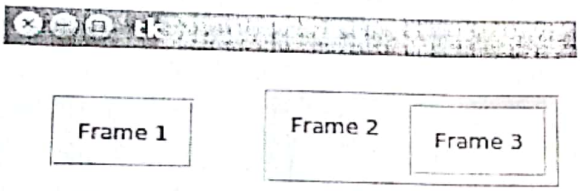

## Frames

Les frames (cadres) sont des conteneurs qui permettent de séparer des éléments.

```
fenetre['bg']='white'
# frame 1
Frame1 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame1.pack(side=LEFT, padx=30, pady=30)
# frame 2
Frame2 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame2.pack(side=LEFT, padx=10, pady=10)
# frame 3 dans frame 2
Frame3 = Frame(Frame2, bg="white", borderwidth=2, relief=GROOVE)
Frame3.pack(side=RIGHT, padx=5, pady=5)
```

```
# Ajout de Labels
Label(Frame1, text="Frame 1").pack(padx=10, pady=10)
Label(Frame2, text="Frame 2").pack(padx=10, pady=10)
Label(Frame3, text="Frame 3",bg="white").pack(padx=10, pady=10)
```

Scanné avec CamScanner




### PanedWindow

Le **panedwindow** est un conteneur qui peut contenir autant de panneaux que nécessaire disposé horizontalement ou verticalement.

```
p = PanedWindow(fenetre, orient=HORIZONTAL)

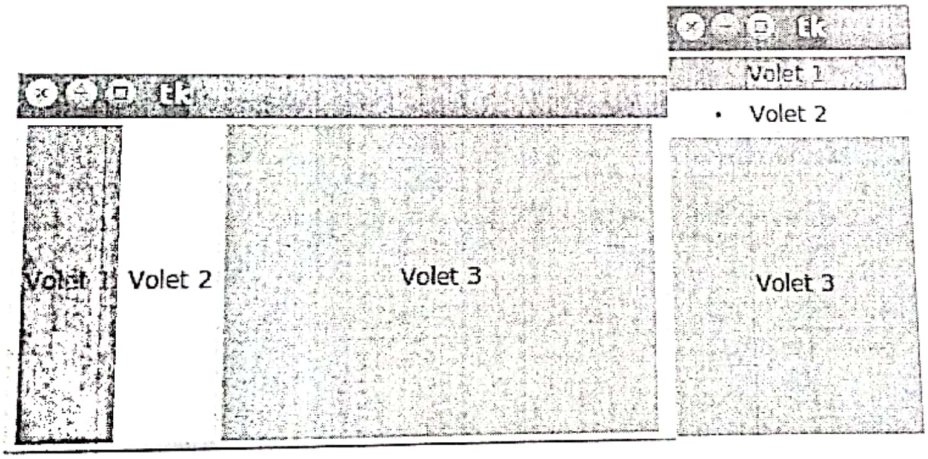
p.pack(side=TOP, expand=Y, fill=BOTH, pady=2, padx=2)

p.add(Label(p, text='Volet 1', background='blue', anchor=CENTER))

p.add(Label(p, text='Volet 2', background='white', anchor=CENTER) )

p.add(Label(p, text='Volet 3', background='red', anchor=CENTER) )

p.pack()
```

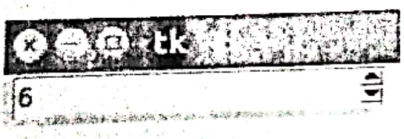



## Spinbox

La **spinbox** propose à l'utilisateur de choisir un nombre

```
s = Spinbox(fenetre, from_=0, to=10)

s.pack()
```






Scanné avec CamScanner

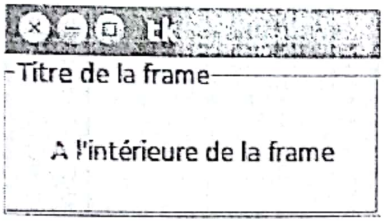
### LabelFrame

Le labelframe est un cadre avec un label.

```
l = LabelFrame(fenetre, text="Titre de la frame", padx=20, pady=20)

l.pack(fill="both", expand="yes")


Label(l, text="A l'intérieure de la frame").pack()
```




# Les alertes

Pour pouvoir utiliser les alertes de votre os, vous pouvez importer le module **tkMessageBox** (Python 2).

```
from tkMessageBox import *
```

Pour python 3:

```
from tkinter.messagebox import *
```

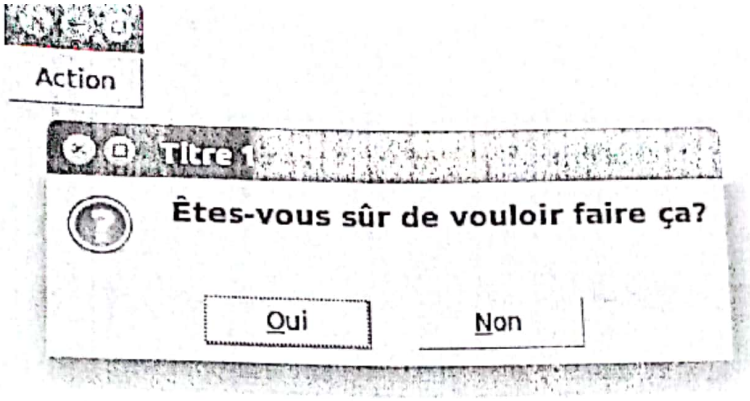
Exemple d'utilisation:

```
def callback():
    if askyesno('Titre 1', 'Êtes-vous sûr de vouloir faire ça?'):
        showwarning('Titre 2', 'Tant pis...')
    else:
        showinfo('Titre 3', 'Vous avez peur!')
        showerror("Titre 4", "Aha")


Button(text='Action', command=callback).pack()
```



Scanné avec CamScanner




Voici les alertes possibles:

```
showinfo()

showwarning()

showerror()

askquestion()

askokcancel()

askyesno()

askretrycancel()
```

# Barre de menu

Il est possible de créer une barre de menu comme-ceci:

```
def alert():
    showinfo("alerte", "Bravo!")

menubar = Menu(fenetre)

menu1 = Menu(menubar, tearoff=0)

menu1.add_command(label="Créer", command=alert)

menu1.add_command(label="Editer", command=alert)

menu1.add_separator()

menu1.add_command(label="Quitter", command=fenetre.quit)

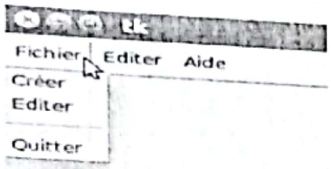
menubar.add_cascade(label="Fichier", menu=menu1)

menu2 = Menu(menubar, tearoff=0)
```



Scanné avec CamScanner

```
menu2.add_command(label="Couper", command=alert)
menu2.add_command(label="Copier", command=alert)
menu2.add_command(label="Coller", command=alert)
menubar.add_cascade(label="Editer", menu=menu2)
menu3 = Menu(menubar, tearoff=0)
menu3.add_command(label="A propos", command=alert)
menubar.add_cascade(label="Aide", menu=menu3)
fenetre.config(menu=menubar)
```




## Connaitre toutes les méthodes / options d'un widget

Pour cela il vous suffit d'exécuter la ligne suivante:

```
print dir(Button())
```

## Les attributs standards

Il est possible de changer la valeur d'attributs présents sur les widgets

## Placer des widgets

Il est possible de placer les widgets à l'aide du paramètre side :

```
side=TOP     :  haut
side=LEFT    :  gauche
side=BOTTOM  :  bas
side=RIGHT   :  droite
```

## Exemple:

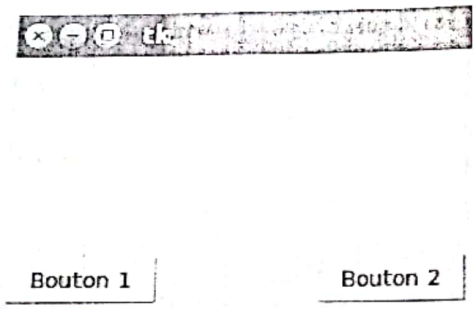
```
Canvas(fenetre, width=250, height=100, bg='ivory').pack(side=TOP, padx=5, pady=5)
```



Scanné avec CamScanner

```
Button(fenetre, text ='Bouton 1').pack(side=LEFT, padx=5, pady=5)

Button(fenetre, text ='Bouton 2').pack(side=RIGHT, padx=5, pady=5)
```

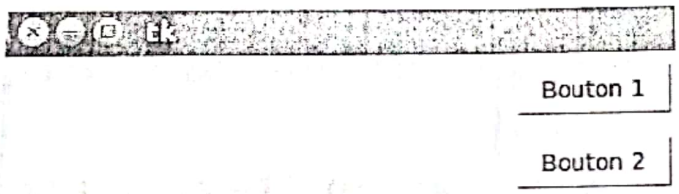



Autre exemple:

```
Canvas(fenetre, width=250, height=50, bg='ivory').pack(side=LEFT, padx=5, pady=5)

Button(fenetre, text ='Bouton 1').pack(side=TOP, padx=5, pady=5)

Button(fenetre, text ='Bouton 2').pack(side=BOTTOM, padx=5, pady=5)
```




### Les unités de dimensions

Si vous indiquez une dimension à travers un integer, l'unité utilisée sera les "pixels". Il est cependant possible de changer l'unité:

i : pouces

m : millimètre

c : centimètre

# Les options de dimensions

**height** : Hauteur du widget.

**width** : Largeur du widget.

**padx, pady** : Espace supplémentaire autour du widget. X pour horizontal et V pour vertical.



Scanné avec CamScanner

borderwidth : Taille de la bordure.

highlightthickness : Largeur du rectangle lorsque le widget a le focus.

selectborderwidth : Largeur de la bordure tridimensionnel autour du widget sélectionné.

wraplength : Nombre de ligne maximum pour les widget en mode "word wrapping".

### Les options de couleurs

Il est possible d'indiquer une valeur de couleur par son nom en anglais: "white", "black", "red", "yellow", etc. ou par son code hexadécimale: #000000, #00FFFF, etc.

background (ou bg) : couleur de fond du widget.

foreground (ou fg) : couleur de premier plan du widget.

activebackground : couleur de fond du widget lorsque celui-ci est actif.

activeForeground : couleur de premier plan du widget lorsque le widget est actif.

disabledForeground : couleur de premier plan du widget lorsque le widget est désactivé.

highlightbackground : Couleur de fond de la région de surbrillance lorsque le widget a le focus.

highlightcolor : couleur de premier plan de la région en surbrillance lorsque le widget a le focus.

selectbackground : Couleur de fond pour les éléments sélectionnés.

selectforeground : couleur de premier plan pour les éléments sélectionnés.

# Le curseur

Vous pouvez changer l'apparance de votre curseur:

```
Button(fenetre, text ="arrow", relief=RAISED, cursor="arrow").pack()
Button(fenetre, text ="circle", relief=RAISED, cursor="circle").pack()
Button(fenetre, text ="clock", relief=RAISED, cursor="clock").pack()
Button(fenetre, text ="cross", relief=RAISED, cursor="cross").pack()
Button(fenetre, text ="dotbox", relief=RAISED, cursor="dotbox").pack()
```



Scanné avec CamScanner

```
Button(fenetre, text ="exchange", relief=RAISED, cursor="exchange").pack()
Button(fenetre, text ="fleur", relief=RAISED, cursor="fleur").pack()
Button(fenetre, text ="heart", relief=RAISED, cursor="heart").pack()
Button(fenetre, text ="man", relief=RAISED, cursor="man").pack()
Button(fenetre, text ="mouse", relief=RAISED, cursor="mouse").pack()
Button(fenetre, text ="pirate", relief=RAISED, cursor="pirate").pack()
Button(fenetre, text ="plus", relief=RAISED, cursor="plus").pack()
Button(fenetre, text ="shuttle", relief=RAISED, cursor="shuttle").pack()
Button(fenetre, text ="sizing", relief=RAISED, cursor="sizing").pack()
Button(fenetre, text ="spider", relief=RAISED, cursor="spider").pack()
Button(fenetre, text ="spraycan", relief=RAISED, cursor="spraycan").pack()
Button(fenetre, text ="star", relief=RAISED, cursor="star").pack()
Button(fenetre, text ="target", relief=RAISED, cursor="target").pack()
Button(fenetre, text ="tcross", relief=RAISED, cursor="tcross").pack()
Button(fenetre, text ="trek", relief=RAISED, cursor="trek").pack()
Button(fenetre, text ="watch", relief=RAISED, cursor="watch").pack()
```

# Le relief

Vous pouvez changer le relief sur vos éléments:

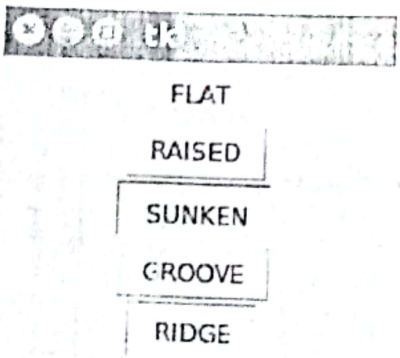
```
FLAT
RAISED
SUNKEN
GROOVE
RIDGE
b1 = Button(fenetre, text ="FLAT", relief=FLAT).pack()
b2 = Button(fenetre, text ="RAISED", relief=RAISED).pack()
b3 = Button(fenetre, text ="SUNKEN", relief=SUNKEN).pack()
```



Scanné avec CamScanner

```
b4 = Button(fenetre, text ="GROOVE", relief=GROOVE).pack()

b5 = Button(fenetre, text ="RIDGE", relief=RIDGE).pack()
```



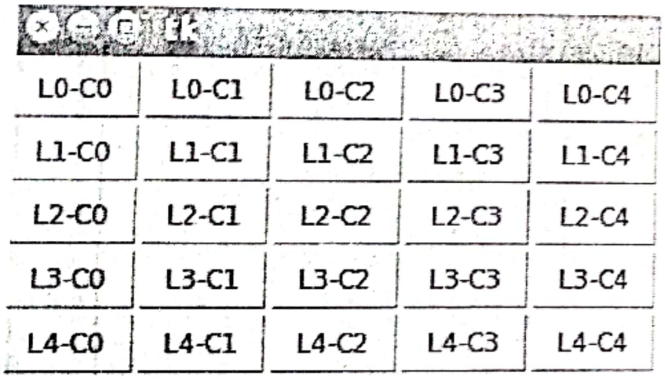

# La grille

Il est possible de placer les éléments en raisonnant en grille:

```
for ligne in range(5):

    for colonne in range(5):

        Button(fenetre, text='L%s-C%s' % (ligne, colonne), borderwidth=1).grid(row
=ligne, column=colonne)
```




```
Button(fenetre, text='L1-C1', borderwidth=1).grid(row=1, column=1)

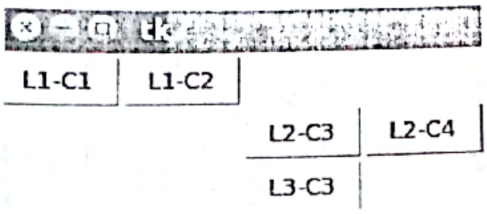
Button(fenetre, text='L1-C2', borderwidth=1).grid(row=1, column=2)

Button(fenetre, text='L2-C3', borderwidth=1).grid(row=2, column=3)

Button(fenetre, text='L2-C4', borderwidth=1).grid(row=2, column=4)

Button(fenetre, text='L3-C3', borderwidth=1).grid(row=3, column=3)
```

Scanné avec CamScanner




# Intégrer une image

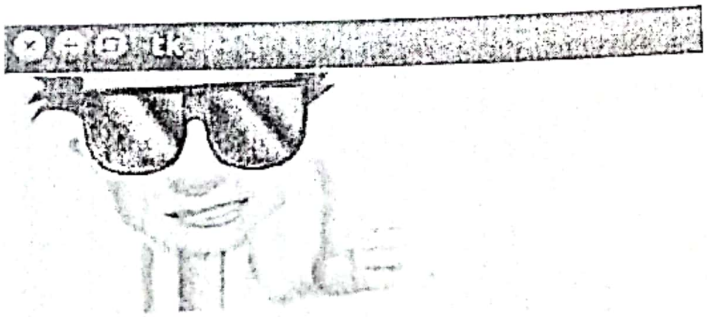
Pour intégrer une image vous pouvez créer un canevas et l'ajouter à l'intérieur comme ceci:

```
photo = PhotoImage(file="ma_photo.png")

canvas = Canvas(fenetre,width=350, height=200)

canvas.create_image(0, 0, anchor=NW, image=photo)

canvas.pack()
```



# Récupérer la valeur d'un input

Pour récupérer la valeur d'un input il vous faudra utiliser la méthode **get()** :

```
def recupere():
    showinfo("Alerte", entree.get())


value = StringVar()

value.set("Valeur")

entree = Entry(fenetre, textvariable=value, width=30)
```



Scanné avec CamScanner

```
entree.pack()

bouton = Button(fenetre, text="Valider", command=recupere)

bouton.pack()
```

### Récupérer une image et l'afficher

Pour cela, vous devez importer le module suivant:

```
from tkinter.filedialog import *

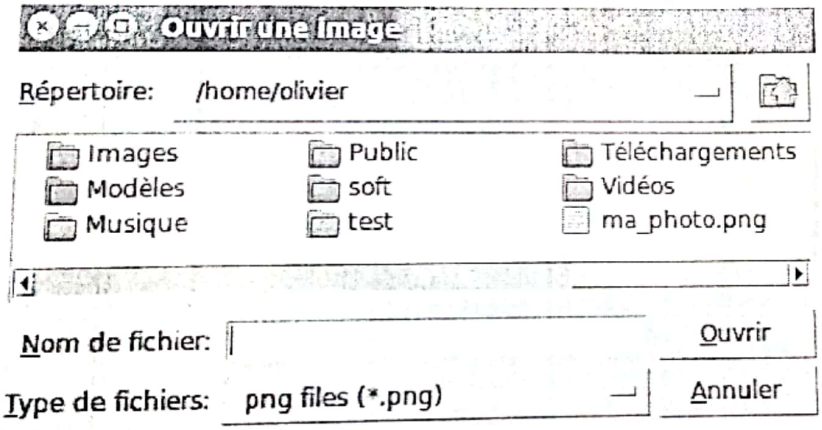
filepath = askopenfilename(title="Ouvrir une image",filetypes=[('png files','.png'
),('all files','.*')])

photo = PhotoImage(file=filepath)

canvas = Canvas(fenetre, width=photo.width(), height=photo.height(), bg="yellow")

canvas.create_image(0, 0, anchor=NW, image=photo)

canvas.pack()
```




La fonction **askopenfilename** retourne le chemin du fichier que vous avez choisi avec le nom de celui-ci.

Exemple: **/home/olivier/ma_photo.png**

## Récupérer un fichier texte et l'afficher

```
filename = askopenfilename(title="Ouvrir votre document",filetypes=[('txt files','
.txt'),('all files','.*')])

fichier = open(filename, "r")

content = fichier.read()
```



Scanné avec CamScanner

```
fichier.close()

Label(fenetre, text=content).pack(padx=10, pady=10)
```

### Les évènements

Vous pouvez récupérer les actions utilisateurs à travers les **events** (évènement en français).

Pour chaque widget, vous pouvez *binder* (lier en français) un évènement, par exemple dire lorsque l'utilisateur appuie sur telle touche, faire cela.

Voici un exemple qui récupère les touches appuyées par l'utilisateur:

```
def clavier(event):

    touche = event.keysym

    print(touche)

canvas = Canvas(fenetre, width=500, height=500)

canvas.focus_set()

canvas.bind("<Key>", clavier)

canvas.pack()
```

On remarque que l' évènement est encadré par des chevrons .

D'autres évènements existent:

| | |
|---|---|
| `<Button-1>` | : Click gauche |
| `<Button-2>` | : Click milieu |
| `<Button-3>` | : Click droit |
| `<Double-Button-1>` | : Double click droit |
| `<Double-Button-2>` | : Double click gauche |
| `<KeyPress>` | : Pression sur une touche |
| `<KeyPress-a>` | : Pression sur la touche A (minuscule) |
| `<KeyPress-A>` | : Pression sur la touche A (majuscule) |
| `<Return>` | : Pression sur la touche entrée |
| `<Escape>` | : Touche Echap |



Scanné avec CamScanner

| | |
|---|---|
| <Up> | : Pression sur la flèche directionnelle haut |
| <Down> | : Pression sur la flèche directionnelle bas |
| <ButtonRelease> | : Lorsque qu'on relache le click |
| <Motion> | : Mouvement de la souris |
| <B1-Motion> | : Mouvement de la souris avec click gauche |
| <Enter> | : Entrée du curseur dans un widget |
| <Leave> | : Sortie du curseur dans un widget |
| <Configure> | : Redimensionnement de la fenêtre |
| <Map> <Unmap> | : Ouverture et iconification de la fenêtre |
| <MouseWheel> | : Utilisation de la roulette |

Pour supprimer la liasion de l'évènement vous pouvez utiliser les méthodes **unbind** ou **unbind_all** .

Voici un exemple où l'on peut bouger un carré avec les touches directionnelles:

```
# fonction appellée lorsque l'utilisateur presse une touche
def clavier(event):
    global coords
    touche = event.keysym
    if touche == "Up":
        coords = (coords[0], coords[1] - 10)
    elif touche == "Down":
        coords = (coords[0], coords[1] + 10)
    elif touche == "Right":
        coords = (coords[0] + 10, coords[1])
    elif touche == "Left":
        coords = (coords[0] -10, coords[1])
    # changement de coordonnées pour le rectangle
```



Scanné avec CamScanner

```
    canvas.coords(rectangle, coords[0], coords[1], coords[0]+25, coords[1]+25)

# création du canvas

canvas = Canvas(fenetre, width=250, height=250, bg="ivory")

# coordonnées initiales

coords = (0, 0)

# création du rectangle

rectangle = canvas.create_rectangle(0,0,25,25,fill='violet")

# ajout du bond sur les touches du clavier

canvas.focus_set()

canvas.bind("<Key>", clavier)

# création du canvas

canvas.pack()
```

Scanné avec CamScanner