Interface Homme Machine
Python-Tkinter
Travaux Pratiques N° 6

Objetcif :

- Programmer l'application pour répondre aux actions de l'utilisateur. Pour ce faire, on définira par des exemples :
  - ✓ Evénment (event)
  - ✓ Gestionnaire d'événemnts (event handler)
  - ✓ Liaison événement<->Gestionnaire d'événements
  - ✓ Niveaux de liaison:
    - Niveau widget
    - Niveau classe de widgets
    - Niveau application

Gestion des Événements

Pour que votre application réponde aux actions de l'utilisateur, vous devez définir des gestionnaires des événements causés par celui-ci. Un utilisateur peut causer un événement en cliquant sur un bouton de la souris, en la déplaçant, ou en appuyant sur une touche du clavier, etc. l'application réagit généralement en exécutant une fonction que nous appelons gestionnaire de l'évènement.

Les widgets ont normalement un grand nombre de comportements prédéfinis. Par exemple, un bouton réagira à un clic souris en appelant la fonction associée à son option **command**. tkinter fourni tous les moyens pour ajouter, changer ou supprimer de tels comportements.

Définitions des termes utilisés:

- **Evénement** (*event*) : est la survenue d'une action (clavier, souris) dont votre application a besoin d'être informée.
- **Gestionnaire d'événement** (*event handler*) : est une fonction de votre application qui est exécuté lorsqu'un événement se produit sur un widget.
- Liaison **widget-événement-gestionnaire d'événement** : consiste à spécifier la fonction à exécuter lorsque l'événement se produit sur le widget.

Niveaux de liaisons

Vous pouvez lier un gestionnaire à un événement à l'un de ces trois niveaux:

1. **Liaison au niveau d'un widget** : Il est possible de lier un gestionnaire d'événement à un événement sur un widget particulier en utilisant sa méthode bind().

   La syntaxe est :

bind(*sequenceEvenement=None, gestionnaireEvenement=None, add=None*)
Cette méthode est utilisée pour attacher un gestionnaire d'événement (une fonction) à la survenue d'un événement, précisé par ***sequenceEvenement***, sur le widget appelant (sur lequel cette méthode a été appliquée). L'argument ***sequenceEvenement*** sert à décrire le type d'événement (action de l'utilisateur) auquel il faut réagir en exécutant ***gestionnaireEvanement***, c'est à dire en appelant cette fonction lorsque survient l'événement surveillé sur le widget. Si une liaison avait déjà été définie sur ce widget, l'ancien gestionnaire d'événement est remplacé par le nouveau sauf si vous utilisez add='+'; dans ce cas les gestionnaires définis précédemment sont préservés.

2. **Liaison au niveau d'une classe** : Vous pouvez lier un événement à tous les widgets d'une classe donnée en utilisant la méthode bind_class().
   La syntaxe est :

bind_class(*type, sequenceEvenement=None, GestionnaireEvenement=None, add=None*)
Similaire à la méthode bind(), mais s'applique à tous les widgets du type indiqué par l'argument ***type*** (par exemple 'Button'). Pout l'appeler, on peut utiliser un widget quelconque: w.bind_class('Button', '<Button-1>, afficher) w est un widget qui n'est pas forcément instance de la classe Button

3. **Liaison au niveau de l'application** : Vous pouvez définir une liaison d'événement de telle sorte que le gestionnaire d'événement soit appelé indépendamment du widget qui a le focus ou qui se trouve sous la souris. Pour ce faire appeler la méthode bind_all() sur n'importe quel widget

La syntaxe est:

Bind_all(*sequenceEvenement=None, gestionnaireEvenement=None, add=None*)
Similaire à la méthode bind(), mais s'applique à tous les widgets de l'application.

Séquence d'événements

Tkinter dispose d'une méthode générale et puissante pour vous permettre d'indiquer précisément à quels événements vos gestionnaires sont liés.

En général, une séquence d'événements est une chaîne de caractères qui contient un ou plusieurs **motifs d'événements**. Chaque motif d'événements décrit quelque chose qui peut survenir pendant l'exécution de votre application. S'il y a plus d'un motif dans une séquence d'événements, le gestionnaire associé sera appelé seulement si tous les motifs de la séquence se produisent en effet.

La forme générale d'un motif d'événement est la suivante:

<[modificateur-]...type[-detail]>	
• Le motif est enfermé entre des chevrons <…>.	
• type	Le type de l'événement décrit le genre général de celui-ci, comme un appui sur une touche, KeyPress, ou un clic souris, Button. Voir Types d'événements ci-dessous.
• [modificateur-]...	Vous pouvez indiquer un ou plusieurs modificateurs avant son type pour décrire une combinaison comme un appui sur la touche *Maj* ou *Control* pendant qu'une autre touche ou qu'un bouton de la souris est enfoncé. Voir *Modificateurs d'événement* ci-dessous
• [-detail]>	Vous pouvez ajouter d'autres détails après le type pour décrire la touche ou le bouton précis qui vous intéresse. Pour les boutons de la souris, 1 indique normalement le bouton de gauche, 2 celui du milieu et 3 celui de droite. Pour les touches du clavier, il s'agit soit d'un caractère (pour un caractère unique comme pour la touche *A* ou *) ou le nom d'une touche comme Escape, Down, Up, BackSpace, Delete..etc. Voir Noms des touches ci-dessous.

Quelques Exemples de motifs d'événements

• <MouseWheel>	L'utilisateur a tourné la molette de la souris
• <Button-3>	L'utilisateur a cliqué sur le bouton 3, c'est-à-dire le bouton droit de la souris.
• <Alt-Button-1>	L'utilisateur a cliqué sur le bouton gauche de la souris tout en maintenant la touche Alt appuyée
• <KeyPress-a>	L'utilisateur a appuyé sur la touche "a".

• <Control-Shift-KeyPress-H>	L'utilisateur a appuyé sur la touche H tout en maintenant les deux touches Control et Shift appuyées
• <Double-Button-1>	L'utilisateur a double cliqué sur le bouton gauche de la souris

Types d'événements

Les types d'événements les plus utilisés sont fournis dans le tableau ci-dessous :

Type	Nom	Description
36	Activate	Un widget est passé de l'état inactif à l'état actif. Se rapporte au changement de l'option **state** des widgets comme un bouton qui est inactif (grisé) et devient actif.
4	Button	L'utilisateur a appuyé sur l'un des boutons de la souris. La partie *détail* précise le bouton: 1 : Bouton Gauche 2 : Bouton du milieu 3 : Bouton droit Pour la roulette, il faut utiliser le type ButtonWheel
5	ButtonRelease	L'utilisateur relâche un bouton de la souris. C'est probablement un meilleur choix dans la plupart des cas d'utiliser ce type d'événement plutôt que Button parce que si l'utilsateur appuie accidentellement sur le bouton, il peut bouger la souris en-dehors du widget pour éviter de lancer l'action.
22	Configure	L'utilisateur a modifié la taille d'un widget, par exemple en déplaçant un coin ou un côté de la fenêtre.
37	Deactivate	Un widget est passé de l'état actif à l'état inactif. Se rapporte au changement de l'option **state** des widgets.
17	Destroy	Un widget a été détruit.
7	Enter	L'utilisateur a bougé la souris qui est entrée dans la partie visible d'un widget. (Ne pas confondre avec la touche Entrée, qui est un événement de type KeyPress pour une touche dont le nom est 'Return').
12	Expose	Cet événement se produit à chaque fois qu'au moins une partie de votre application ou d'un widget devient visible après avoir été recouverte par une autre fenêtre.
9	FocusIn	Un widget obtient le focus. Cela peut se produire soit en réponse à une action de l'utilisateur (comme en utilisant la touche *Tab* pour déplacer le focus entre les widgets) ou de manière programmée (par exemple lorsque votre programme appelle la méthode focus_set() sur un widget).
10	FocusOut	Le focus a été perdu par un widget. Comme avec FocusIn, l'utilisateur peut produire un tel événement ou il peut être produit de manière programmée.

Type	Nom	Description
2	KeyPress	L'utilisateur a appuyé sur une touche du clavier. La partie *détail* précise optionnellement une touche en particulier. Ce mot clé peut être abrégé par Key.
3	KeyRelease	L'utilisateur a relâché une touche du clavier.
8	Leave	L'utilisateur a déplacé le pointeur de la souris en dehors d'un widget.
19	Map	Un widget a été «mappé» (associé), c'est à dire, a été rendu visible dans l'application. Cela arrive, par exemple, lorsque vous appelez la méthode **grid()** d'un widget.
6	Motion	L'utilisateur a déplacé la souris à l'intérieur d'un widget.
38	MouseWheel	L'utilisateur a tourné la molette de la souris, vers le haut ou vers le bas.
18	Unmap	Un widget a perdu l'association (le «mappage») et n'est plus visible. Cela arrive, par exemple, lorsque vous appelez la méthode grid_remove() d'un widget.
15	Visibility	Se produit lorsqu'au moins une partie de la fenêtre d'application est devenue visible à l'écran.

Modificateurs d'événement

Les modificateurs que vous pouvez utiliser dans une séquence d'événements sont une combinaison d'un ou plusieurs noms dont les principaux figurent dans le tableau ci-dessous :

Nom	Description
Alt	Vrai si l'utilisateur est en train de maintenir enfoncée la touche *Alt*.
Any	Ce modificateur généralise un type d'événement. Par exemple, le motif d'événement '<Any-KeyPress>'correspond à l'appui sur une touche arbitraire.
Control	Vrai si l'utilisateur est en train de maintenir enfoncée la touche *Ctrl*.
Double	Indique qu'un événement s'est produit 2 fois dans un cours laps de temps. Par exemple, <Double-Button-1> indique un double clic sur le bouton gauche (normalement) de la souris.
Lock	Vrai si l'utilisateur a verrouiller le mode Majuscule.
Shift	Vrai si l'utilisateur est en train de maintenir enfoncée la touche *Maj*.
Triple	Comme Double, mais pour l'apparition du même événement 3 fois dans un cours laps de temps.

Noms des touches

La partie *detail* d'un motif pour un événement KeyPress ou KeyRelease précise la touche que vous souhaitez surveiller. (Voir le modificateur Any ci-dessus si vous souhaitez surveiller toutes les touches).

Le tableau ci-dessous montre plusieurs façons de nommer les touches.

Nom de la touche	Signification
keysym	Correspond au «symbole de touche», qui est une chaîne de caractères pour la touche. Cela correspond à l'attribut keysym des objets Event.
keycode	Correpond au «code de touche». C'est un *identifiant* de touche (chaque touche possède un unique keycode) qui permet de savoir quelle touche a été enfoncée. Notez cependant qu'il ne permet pas de savoir si une touche modificatrice (*Maj*, *Ctrl* et *VerrMaj*) a été ou est enfoncée; ainsi, par exemple, *a* et *A* ont le même code de touche.
keysym_num	Correspond à un code numérique équivalent au symbole de la touche. Il a la particularité d'être différent selon qu'une touche modificatrice a été ou est enfoncée. Par exemple, le chiffre 2 du clavier numérique (dont le symbole de touche est KP_2) et la flèche «sud» du clavier numérique (de symbole KP_Down) ont le même code de touche (88), mais leurs codes numériques keysym_num sont différents (65433 et 65458, respectivement).

Il y a beaucoup de noms de touches pour couvrir de nombreux ensembles de caractères internationaux. Ce tableau montre uniquement l'ensemble «Latin-1» pour un clavier type.

keysym	*keycode*	*keysym_num*	Touche
Alt_L	*64*	*65513*	La touche *Alt* située à gauche.
BackSpace	*22*	*65288*	La touche *Retour Arrière*
Cancel	*110*	*65387*	???
Caps_Lock	*66*	*65509*	*Verr Maj*
Control_L	*37*	*65507*	La touche *Ctrl* de gauche
Control_R	*105*	*65508*	La touche *Ctrl* de droite
Delete	*119*	*65535*	*Suppr*
Down	*116*	*65364*	↓
End	*115*	*65367*	*Fin*
Escape	*9*	*65307*	*Echap*
Execute	*111*	*65378*	
F1	*67*	*65470*	La touche fonction *F1*
F2	*68*	*65471*	La touche fonction *F2*
Fi	*66+i*	*65469+i*	La touche fonction *Fi*
F12	*96*	*65481*	La touche fonction *F12*
Home	*110*	*65360*	*Début*
Insert	*118*	*65379*	*Inser*
Left	*113*	*65361*	←
Linefeed	*54*	*106*	Linefeed (control-J)
KP_0	*90*	*65456*	0 sur le clavier numérique
KP_1	*87*	*65457*	1 sur le clavier numérique
KP_2	*88*	*65458*	2 sur le clavier numérique
KP_3	*89*	*65459*	3 sur le clavier numérique

keysym	*keycode*	*keysym_num*	Touche
KP_4	*83*	*65460*	4 sur le clavier numérique
KP_5	*84*	*65461*	5 sur le clavier numérique
KP_6	*85*	*65462*	6 sur le clavier numérique
KP_7	*79*	*65463*	7 sur le clavier numérique
KP_8	*80*	*65464*	8 sur le clavier numérique
KP_9	*81*	*65465*	9 sur le clavier numérique
KP_Add	*86*	*65451*	+ sur le clavier numérique
KP_Begin	*84*	*65437*	La touche centrale (même que 5) sur le clavier numérique
KP_Decimal	*91*	*65454*	Symbole de la ponctuation décimale (,) sur le clavier numérique
KP_Delete	*91*	*65439*	*Suppr* sur le clavier numérique
KP_Divide	*106*	*65455*	/ sur le clavier numérique
KP_Down	*88*	*65433*	↓ sur le clavier numérique
KP_End	*87*	*65436*	*Fin* sur le clavier numérique
KP_Enter	*104*	*65421*	*Entrée* sur le clavier numérique
KP_Home	*79*	*65429*	*Début* sur le clavier numérique
KP_Insert	*90*	*65438*	*Insert* sur le clavier numérique
KP_Left	*83*	*65430*	← sur le clavier numérique
KP_Multiply	*63*	*65450*	× sur le clavier numérique
KP_Next	*89*	*65435*	*PageDown* sur le clavier numérique
KP_Prior	*81*	*65434*	*PageUp* sur le clavier numérique
KP_Right	*85*	*65432*	→ sur le clavier numérique
KP_Subtract	*82*	*65453*	- sur le clavier numérique
KP_Up	*80*	*65431*	↑ sur le clavier numérique
Next	*117*	*65366*	*PageDown*
Num_Lock	*77*	*65407*	*Verr Num*
Pause	*127*	*65299*	*Pause*
Print	*111*	*65377*	*ImprÉcran*
Prior	*112*	*65365*	*PageUp*
Return	*36*	*65293*	La touche *Entrée* (control-M). Le nom Enter se réfère à un événement associé à la souris et non au clavier ; voir Types d'événements
Right	*114*	*65363*	→
Scroll_Lock	*78*	*65300*	Verrouillage Défilement (*ScrollLock*)
Shift_L	*50*	*65505*	La touche *Maj* de gauche
Shift_R	*62*	*65506*	La touche *Maj* de droite
space	*65*	*32*	La barre espace
Tab	*23*	*65289*	La touche de Tabulation, *Tab*
Up	*111*	*65362*	↑

Gestionnaire d'événement : La classe Event

Dans les sections précédentes nous avons appris comment spécifier ou décrire l'évenement que nous voulons gérer, et comment le lier à un widget de l'interface graphique de l'application. Dans ces sections, nous allons apprendre comment écrire la fonction qui va gérer cet événement autrement dit qui va être appelée lorsque l'événement se produira sur le widget

La fonction gestionnaire d'événements recevra un objet de type Event qui sert à décrire les circonstances de l'événement. Le gestionnaire peut être une fonction ou une méthode. Voici la forme de la déclaration d'une fonction :

```
Def nomGestionnaire(event):
```

Les attributs de l'objet de type Event passé au gestionnaire, par l'intermédiaire de son paramètre event sont décrits ci-dessous. Certains attributs possèdent toujours une valeur, mais d'autres n'en possède une que pour certains types d'événements.

Attribut	Signification
char	Si l'événement est produit par l'appui ou le relâchement d'une touche qui produit un caractère ASCII régulier, cet attribut est le caractère sous la forme d'une chaîne. (Pour des touches spéciales comme *Suppr*, voir l'attribut keysym ci-dessous)
delta	Pour un événement du type MouseWheel, cet attribut contient un entier dont le signe est positif pour un déplacement vers le haut, négatif pour un déplacement vers le bas.
height	Si l'événement est du type Configure, cet attribut porte la nouvelle hauteur du widget en pixels.
keycode	Pour un événement de type KeyPress ou KeyRelease, cet attribut contient le code de touche. Cependant, cet entier n'identifie pas quel caractère de la touche a été produit, ainsi «x» ou «X» ne se différencient pas par leur code de touche. Pour des valeurs possibles de cet attribut, voir *Noms des touches*.
keysym	Pour un événement de type KeyPress ou KeyRelease impliquant une touche spéciale, cet attribut porte le nom de touche, par exemple 'Prior' pour la touche *PageUp*. Voir *Noms des touches* pour une liste complète des nom de touches.
keysym_num	Pour un événement de type KeyPress ou KeyRelease, cet attribut est une version numérique de l'attribut keysym. Pour une touche régulière qui produit un seul caractère, cet attribut prend pour valeur le code ASCII du caractère. Pour des touches spéciales, référez-vous à Noms des touches.
num	Si l'événement est associé à un bouton de la souris, cet attribut porte la valeur entière qui indique le numéro du bouton (1, 2 ou 3). Pour le support de la molette sous linux, lier les événements Button-4et Button-5; lorsque la molette de la souris tourne vers l'avant, cet attribut prend la valeur 4, il prend la valeur 5 dans l'autre sens.
serial	Un entier qui est incrémenté à chaque fois que le serveur répond à une requête du client. Vous pouvez utiliser cet attribut pour découvrir la

	séquence temporelle des événements: ceux qui ont eu lieu plus tôt ont une valeur plus petite.
state	Un entier qui décrit l'état de toutes les touches modificatrice. Reportez-vous à la table des masques des modificateurs pour l'interprétation de cette valeur.
time	Cet attribut porte un entier qui n'a pas de signification dans l'absolu, mais qui est incrémenté chaque milliseconde. Cela permet à votre application de déterminer, par exemple, le temps écoulé entre deux clic souris.
type	Un code numérique qui décrit le type de l'événement. Pour l'interprétation de ce code, reportez-vous à *Types d'événements*.
widget	Porte toujours la référence du widget qui a causé l'événement. Par exemple, si l'événement était un clic souris sur un canevas, cet attribut serait ce canevas.
width	Si l'événement était du type Configure, cet attribut est la nouvelle largeur du widget en pixels.
x	L'abscisse de la souris en pixels au moment de l'événement. Elle est relative au coin supérieur gauche du widget sur lequel se trouve la souris.
y	Similaire à x mais dans la direction verticale.
x_root	L'abscisse de la souris au moment où survient l'événement, relativement au coin supérieur gauche de l'écran.
y_root	Similaire à x_root mais dans la direction verticale.

Utilisez ces masques pour tester les bits de la valeur de l'attribut state pour savoir quel(s) touche(s) modificatrice(s) et/ou bouton(s) ont été utilisé(s) pendant l'événement.

Masque	Modificateur
0x0001	*Maj.*
0x0002	*Verr Maj.*
0x0004	*Control.*
0x0008	Touche *Alt* de gauche.
0x0010	*Verr Num.*
0x0080	Touche *Alt* de droite.
0x0100	Bouton 1 de la souris.
0x0200	Bouton 2 de la souris.
0x0400	Bouton 3 de la souris.

Événements virtuels

Vous pouvez créer vos propres genres d'événements appelés «événements virtuels». A l'aide de la méthode event_add dont la syntaxe est :

event_add(*"<<nomEvenement>>", *sequences*)
Cette méthode crée un événement virtuel dont le nom est la chaîne de caractères donnée comme premier argument *placée entre << et >>*. Chaque argument supplémentaire décrit une «séquence», c'est à dire, la description d'un événement physique (appui sur une

touche, mouvement de la souris ...). Lorsque cet événement se produit, le nouvel événement virtuel est déclenché.

Vous pouvez leur donner le nom que vous souhaitez du moment qu'il est entouré par des doubles paires de chevrons `<<…>>`.

Par exemple, supposez que vous vouliez créer un nouvel événement appelé `'<<panic>>'`, qui est déclenché par le bouton 3 de la souris ou la touche *Pause*. Pour créer cet événement, appeler cette méthode sur un widget `w` arbitraire:

```
w.event_add('<<panic>>', '<Button-3>', '<KeyPress-Pause>')
```

Vous pouvez alors utiliser `'<<panic>>'` dans n'importe quelle séquence d'événements. Par exemple:

```
w.bind('<<panic>>', g)
```

L'appui sur le bouton 3 de la souris ou sur la touche *Pause* dans le widget `w` déclenchera le gestionnaire `g`.

Pour supprimer l'évènement utiliser la méthode **event_delete()** dont la syntaxe est:

event_delete(*"<<nomEvenement>>", *sequences*)
Supprime le ou les événements physiques associés à l'événement virtuel dont le nom est précisé en premier argument. Si tous les événements physiques sont supprimés de l'événement virtuel, cet événement virtuel ne sera plus déclenché.

La méthode **event_info** permet de récupérer les événements physiques associés à un événement virtuel.

Sa syntaxe est:

event_info(*virtual=None*)
Si vous appelez cette méthode sans argument, vous obtenez la liste de tous les événements virtuels qui sont actuellement définis. Pour récupérer les événements physiques associés à un événement virtuel, précisez son nom et vous obtiendrez la liste de tous les événements physiques associés ou `None` s'il n'y en a pas.

Programmes de mise en œuvre des différents concepts exposés dans les sections précedentes.

Programme mettant en œuvre la méthode bind et les différents formats de son paramètre sequenceEvenement.

bind(*sequenceEvenement=None, gestionnaireEvenement=None, add=None*)

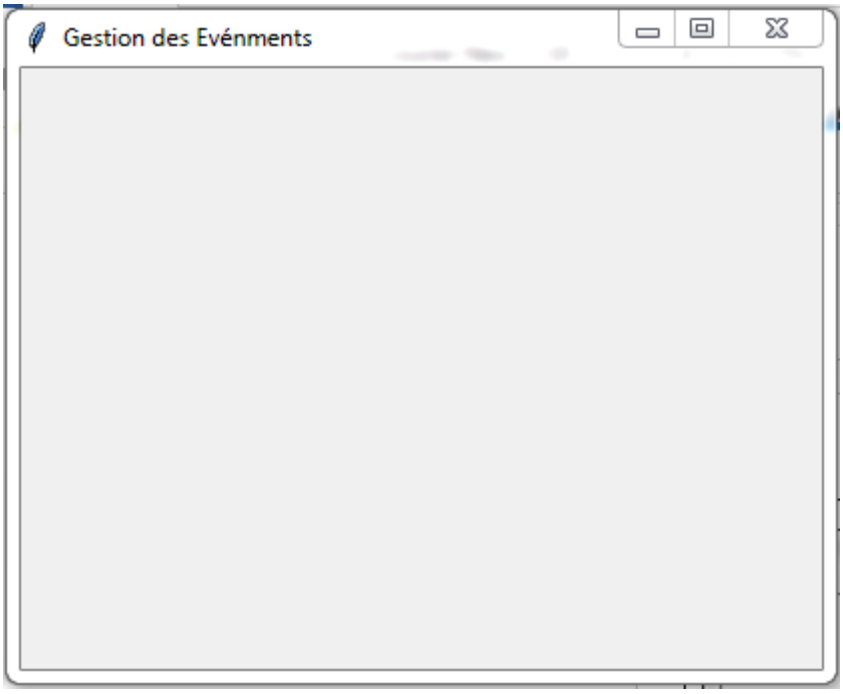
sequenceEvenement:

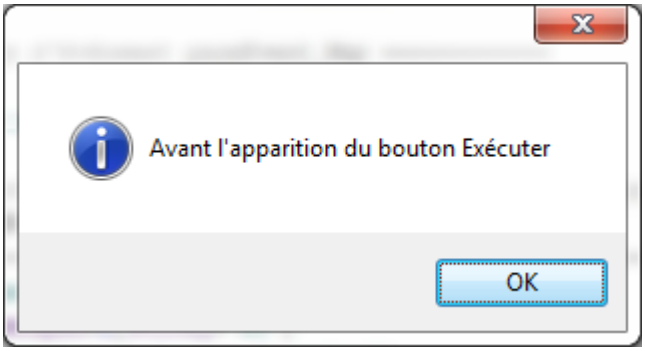
<[modificateur-]...type[-detail]>

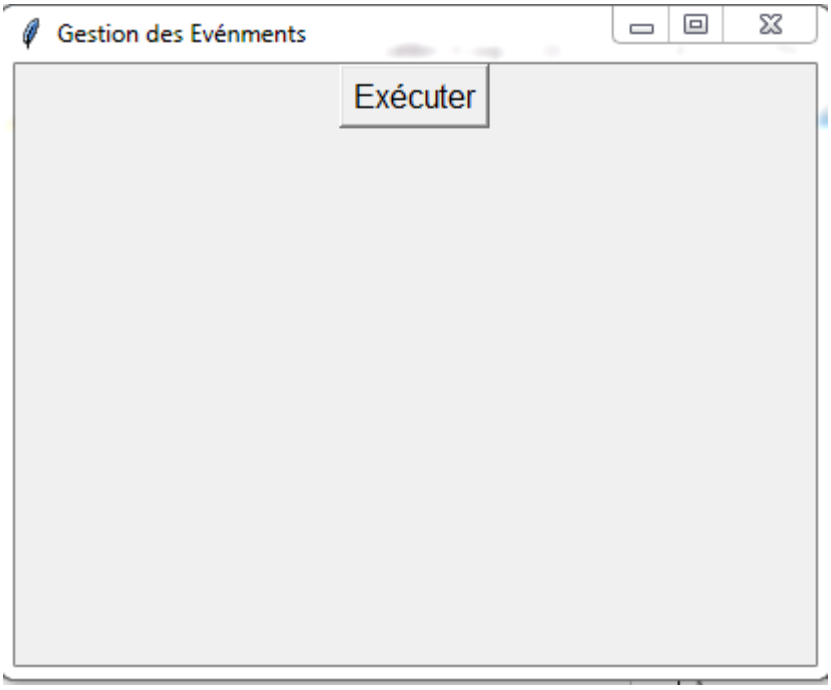
```
from tkinter import *
from tkinter import messagebox
from tkinter import scrolledtext
from functools import partial

root = Tk()
root.geometry("400x300+50+50")
root.title("Gestion des Evénments")
frame = Frame(root)
frame.pack()

btnevent = Button(frame, font=(16), text="Exécuter")
```

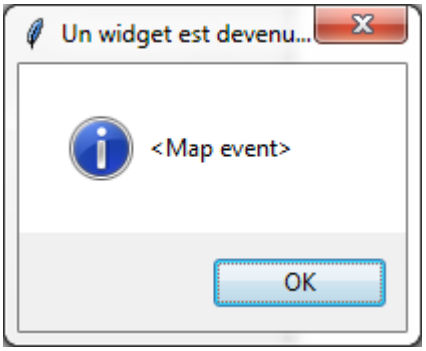
```
#************ Type d'événement = Map (apparition d'un widget)***************
```

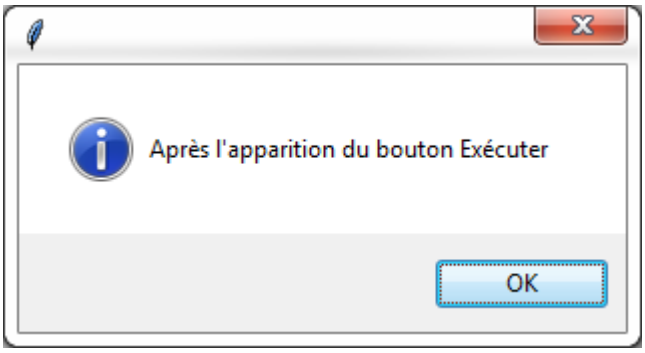
```
#=========Définition un gestionnaire d'événemet gereEvent_Map ============
def gereEvent_Map(event, widget):
    messagebox.showinfo("Un widget est devenu visible", widget)

#=Liaison du gestionnaire gereEvent_Map à l'évenemnt <Map>  pour le bouton btnevent ==
btnevent.bind('<Map>', partial(gereEvent_Map, btnevent))
#=============================================================================
messagebox.showinfo("","Avant l'apparition du bouton Exécuter")
btnevent.grid(row=0, column=0, columnspan=2,sticky="ew")
messagebox.showinfo("","Après l'apparition du bouton Exécuter")
#=============================================================================
```

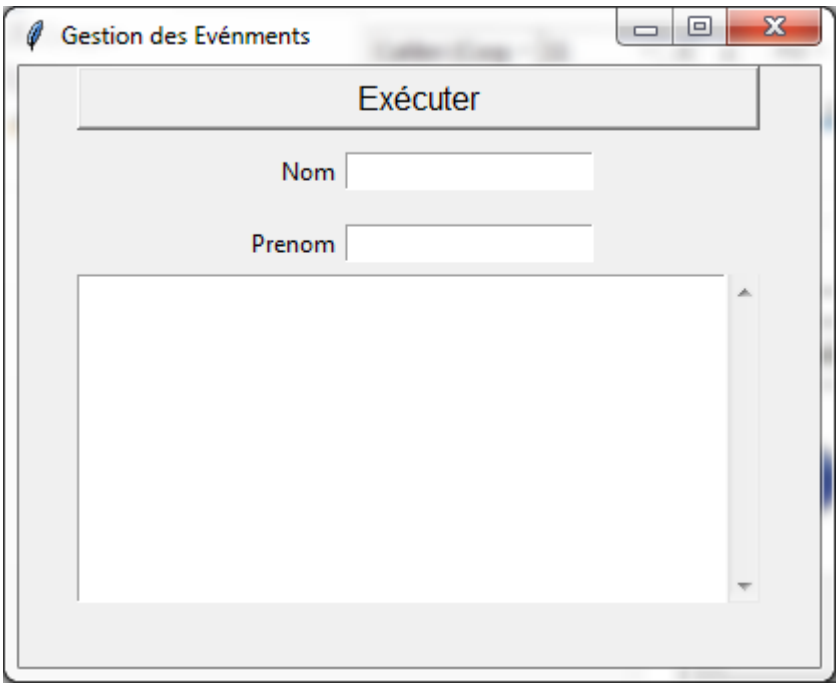

Gestion des Evénments


Avant l'apparition du bouton Exécuter
OK


Gestion des Evénments
Exécuter








```
#***Type Evenement = Button (Click sur un bouton de la souris)**********************
```

```
#-------------------------------------------------------------------------------
#Type d'événement = Button, detail=1 ,   pas de modificateur
# Simple click sur le bouton Gauche de la souris
#-------------------------------------------------------------------------------
```

```
def gereEventMouse_1_Gauche(event):
    messagebox.showinfo("clic sur un bouton", "vous avez cliqué sur le bouton gauche")

#Transformer la ligne suivante en commentaire pour exécuter <Double-Button-1>
```

```
btnevent.bind('<Button-1>', gereEventMouse_1_Gauche)
#=============================================================================================

#---------------------------------------------------------------------------------------------
#Type d'événement = Button, detail=2 ,   pas de modificateur
# Simple click sur le bouton Milieu de la souris
#---------------------------------------------------------------------------------------------
def gereEventMouse_2_Milieu(event):
    messagebox.showinfo("clic sur un bouton", "vous avez cliqué sur le bouton Milieu")

btnevent.bind('<Button-2>', gereEventMouse_2_Milieu)
#=============================================================================================

#---------------------------------------------------------------------------------------------
#Type d'événement = Button, detail=3 ,   pas de modificateur
# Simple click sur le bouton Droit de la souris
#---------------------------------------------------------------------------------------------
def gereEventMouse_3_Droite(event):
    messagebox.showinfo("clic sur un bouton", "vous avez cliqué sur le bouton Droite")

btnevent.bind('<Button-3>', gereEventMouse_3_Droite)
#=============================================================================================

nomVar = StringVar(frame)
lbNom = Label(frame, text = "Nom")
entNom = Entry(frame, textvariable=nomVar, name="nom")
lbNom.grid(row=1, column=0, sticky="e", padx=2, pady=10)
entNom.grid(row=1, column=1, sticky="w", pady=10)

prenomVar = StringVar(frame)
lbPrenom = Label(frame, text = "Prenom")
entPrenom = Entry(frame, name="prenom")
lbPrenom.grid(row=2, column=0, sticky="e", padx=2, pady=5)
entPrenom.grid(row=2, column=1, sticky="w", pady=5)

#***************Type Evenement = FocusIn**************************

def gereEvent_FocusIn(event):
    w = event.widget
    print("FocusIn :" +  w._name + " a obtenu le focus")

entNom.bind('<FocusIn>', gereEvent_FocusIn)

entPrenom.bind('<FocusIn>', gereEvent_FocusIn)
#=============================================================================================

#***************Type Evenement = FocusOut**************************
def gereEvent_FocusOut(event):
    w = event.widget
    print("FocusOut : " +  w._name + " a perdu le focus")

entNom.bind('<FocusOut>', gereEvent_FocusOut)

entPrenom.bind('<FocusOut>', gereEvent_FocusOut)
#=============================================================================================

#***************Type Evenement = Escape (Appui sur la touche Echap*****************
```

```
def quitApp(e):
    ret = messagebox.askquestion("Question", "Etes-vous sur de vouloir quitter?",
default=messagebox.NO)
    if (ret == "yes"):
        quit()
    else:
        return

#==Liaison du gestionnaire quitApp à l'événement <Escape> sur la fenêtre root
root.bind("<Escape>", quitApp)
#=============================================================================
```

```
textarea = scrolledtext.ScrolledText(frame,width=40,height=10)
textarea.grid(row=3, column=0, columnspan=2)
```

```
#******Type Evenement = MouseWheel(Tourner la molette de la souris)****************

def gereEventMouseWheel(event):
    messagebox.showinfo("", "vous avez tourné la molette de la souris")

textarea.bind('<MouseWheel>', gereEventMouseWheel)
#=============================================================================
```

```
#**********Type Evenement : KeyPress (Appui sur une touche quelconque)***********

def gereEvent_KeyPress(event):
    kp = event.char
    messagebox.showinfo("Appui sur une touche", kp)

entPrenom.bind('<KeyPress>', gereEvent_KeyPress)
#=============================================================================
```

```
#-Type Evenemnt : KeyPress detail = Return (Appui sur la touche Entrée du clavier----

def gereEvent_KeyPressReturn(event):
    messagebox.showinfo("Evenement sur textarea", "la touche Entrée a été appuyée,
passgae à la ligne suivante ")

textarea.bind("<KeyPress-Return>", gereEvent_KeyPressReturn)
#=============================================================================
```

```
#------Type Evenemnt : KeyPress detail = h (Appui sur la touche h minuscule)--------

def gereEvent_KeyPress_h(event):
    messagebox.showinfo("Evenement sur textarea", "****la touche h a été appuyée***** ")

textarea.bind("<KeyPress-h>", gereEvent_KeyPress_h)
#=============================================================================
```

```
#Type Evenemnt:KeyPress, detail=flèche vers le bas(Appui sur la touche flèche vers le
bas)

def gereEvent_KeyPress_Down(event):
    messagebox.showinfo("Evenement sur textarea", "**la touche Down a été appuyée** ")

textarea.bind("<KeyPress-Down>", gereEvent_KeyPress_Down)
#=============================================================================
```

```
#--Type Evenemnt : KeyPress detail = flêche vers le haut (Appui sur la touche flêche
vers le haut du clavier principal)----

def gereEvent_KeyPress_Up(event):
```

```
    messagebox.showinfo("Evenement sur textarea", "**la touche Up a été appuyée** ")

textarea.bind("<KeyPress-Up>", gereEvent_KeyPress_Up)
#=============================================================================================

#*****************************************************
# Type Evenement = Button, detail = 1 avec Modificateur
#*****************************************************

#----------------------
#Modificateur = Double
#----------------------
def gereEvent_DoubleClick_Mouse_1_Gauche(event):
    messagebox.showinfo("clic sur un bouton", "vous avez cliqué sur la touche ALT et sur
le bouton gauche")

btnevent.bind('<Double-Button-1>', gereEvent_DoubleClick_Mouse_1_Gauche)
#-------------------------------------------------------------------------------------------

--------------------
#Modificateur = Alt
#--------------------
def gereEvent_Alt_Mouse_1_Gauche(event):
    messagebox.showinfo("clic sur un bouton", "vous avez double cliqué  sur le bouton
gauche tout en appuyant sur la touche Alt")

btnevent.bind('<Alt-Button-1>', gereEvent_Alt_Mouse_1_Gauche)
#-------------------------------------------------------------------------------------------

-----------------------------
#Modificateur = Control+Shift
#-----------------------------
def gereEvent_Control_Shift_Mouse_1_Gauche(event):
    messagebox.showinfo("clic sur un bouton", "vous avez cliqué  sur le bouton gauche
tout en appuyant sur les touches Control + Shift")

btnevent.bind('<Control-Shift-Button-1>', gereEvent_Control_Shift_Mouse_1_Gauche)
#-------------------------------------------------------------------------------------------


root.mainloop()
```

Exemple de programme qui crée un événement personnalisé

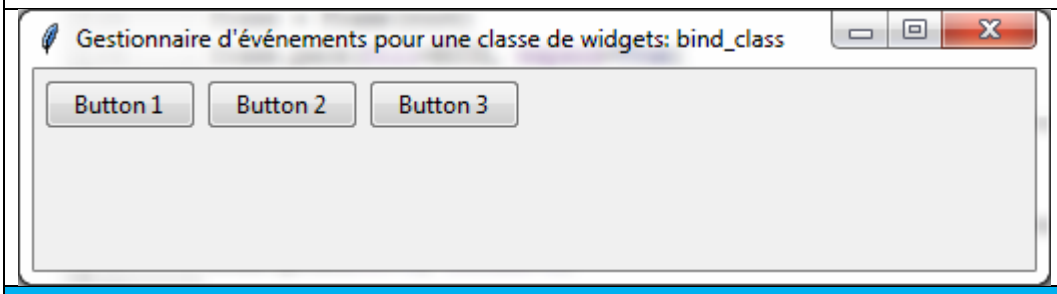
```
from tkinter import Tk, BOTH, StringVar
from tkinter.ttk import Frame, Button, Label
from random import randint

root = Tk()
root.geometry("500x100+300+300")
root.title("Evénement personnalisé")

frame = Frame(root)
frame.pack(fill=BOTH, expand=True)

def onClick():
    frame.event_generate('<<Random>>')
```

```
def updateLabel(e):
    global lvar
    lvar.set(randint(0,100))


#Création d'un nouveau Evenement <<Random>>
frame.bind("<<Random>>", updateLabel)

btn = Button(frame, text="Random", command=onClick)
btn.grid(row=0, column=0, padx=10, pady=10)

lvar = StringVar()
lbl = Label(frame, textvariable=lvar)
lbl.grid(row=0, column=1, padx=50)

root.mainloop()
```

Liaison d'un gestionnaire d'événement à une classe de widget : méthode bind_class

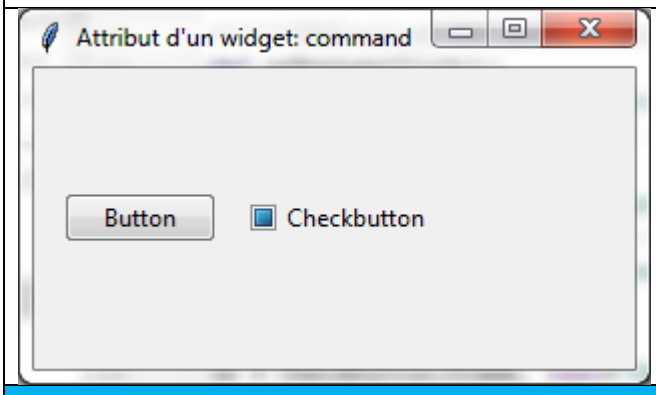
```
from tkinter import Tk, BOTH
from tkinter.ttk import Frame, Button

#==Gestionnaire de l'événement click sur le bouton gauche de la souris
#====pour la classe des boutons=====
def onButtonsClick(e):
    print(e.widget)


def onButton1Click():
    print("Gestionnaire particulier de : ")


def onButton2Click():
    print("Gestionnaire particulier de : ")


def onButton3Click():
    print("Gestionnaire particulier de : ")

root = Tk()
root.geometry("300x200+300+300")
root.title("Gestionnaire d'événements pour une classe de widgets: bind_class")

frame = Frame(root)
frame.pack(fill=BOTH, expand=True)

btn1 = Button(frame, text="Button 1", name="btn1", command=onButton1Click)
btn1.grid(row=0, column=0, pady=5, padx=5)

btn2 = Button(frame, text="Button 2", name="btn2", command=onButton2Click)
btn2.grid(row=0, column=1)

btn3 = Button(frame, text="Button 3", name="btn3", command=onButton3Click)
btn3.grid(row=0, column=2, padx=5)

btn1.bind_class("TButton", "<Button-1>", onButtonsClick, add="+")
```

```
root.mainloop()
```



Utilisation de l'attribut "command" d'un widget pour lier une fonction à l'événement click

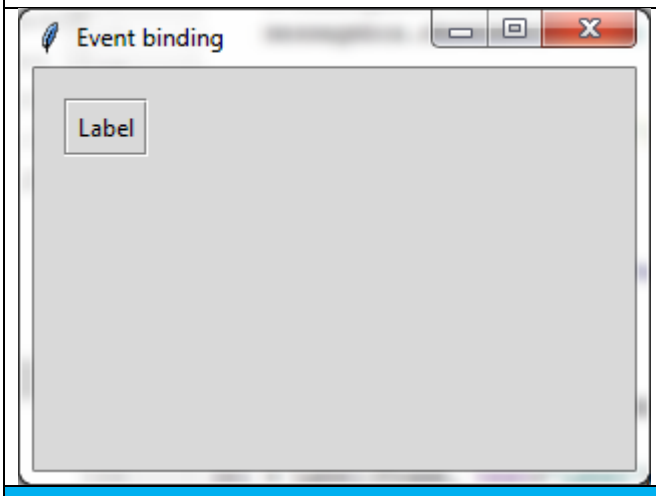
```
from tkinter import Tk, BOTH, LEFT
from tkinter.ttk import Frame, Button, Checkbutton
from tkinter import messagebox


root = Tk()
root.geometry("300x150+300+300")
root.title("Attribut d'un widget: command")
frame = Frame(root)
frame.pack(fill=BOTH, expand=True)

#*****Définition des gestionnaires d'événements*****
def onButton1Click():
    messagebox.showinfo("Réponse à un événement", "Button cliqué")
#-------------------------------------------------
def onButton2Click():
    messagebox.showinfo("Réponse à un événement", "Checkbutton cliqué")
#-------------------------------------------------
btn = Button(frame, text="Button", command=onButton1Click)
btn.pack(side=LEFT, padx=15)

cb = Checkbutton(frame, text="Checkbutton", command=onButton2Click)
cb.pack(side=LEFT)

root.mainloop()
```



Gestionnaire d'événement recevant un argument en plus de event

```
from tkinter import Tk, BOTH, N, W, GROOVE
from tkinter.ttk import Frame, Label, Style
from tkinter import messagebox
```

```
def onLabelEnter(event, s):
    s.configure('TLabel', foreground='red')
    messagebox.showinfo("Entrée de la Souris dans le label")


def onLabelLeave(event, s):
    s.configure('TLabel', foreground='black')
    messagebox.showinfo("Sortie de la Souris du label")

root = Tk()
root.geometry("300x200+300+300")
root.title("Event binding")

frame = Frame(root)
frame.pack(fill=BOTH, expand=True)

s = Style()
s.theme_use("alt")
s.configure('TLabel', relief=GROOVE, padding=5, bd=2)

lbl = Label(frame, text="Label")
lbl.pack(anchor=N+W, padx=15, pady=15)

lbl.bind("<Enter>", lambda event : onLabelEnter(event, s))
lbl.bind("<Leave>", lambda event : onLabelLeave(event, s))

root.mainloop()
```



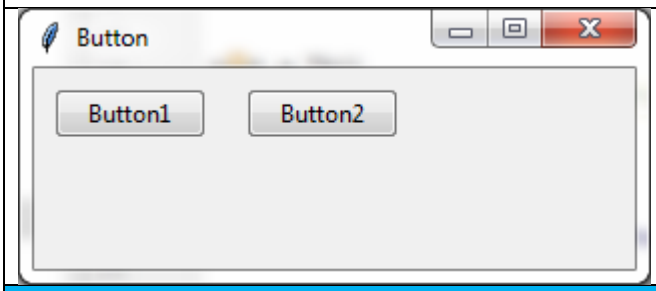
Liaison et déliaison d'un gestionnaire d'événement: Méthodes bind et unbind

```
from tkinter import Tk, BOTH, BooleanVar
from tkinter.ttk import Frame, Button, Checkbutton
from tkinter import messagebox

def onBind(widget):
    if (var.get() == True):
        widget.bind("<Button-1>", onClick)
    else:
        widget.unbind("<Button-1>")

def onClick(event):
    messagebox.showinfo("Liaison et déliaison d'un gestionnaire d'événement", "Bouton
cliqué, Exécution de onClick")

root = Tk()
root.geometry("300x200+300+300")
root.title("Liaison et Déliaison d'un Gestionnaire d'Evenement : Méthodes bind et
unbind")

frame = Frame(root)
frame.pack(fill=BOTH, expand=True)

btn = Button(frame, text="Click")
btn.grid(row=0, column=0, padx=10, pady=10)

var = BooleanVar()
cb = Checkbutton(frame, text="Bind event", variable=var, command=lambda :
onBind(btn))
cb.grid(row=0, column=1)

root.mainloop()
```

Liaison de plusieurs gestionnaires à un même évènement sur un widget

```
from tkinter import Tk, BOTH
from tkinter.ttk import Frame, Button
from tkinter import messagebox

#===Définition de la première fonction gestionnaire d'événement
def onClick1(event):
    messagebox.showinfo("Appel de plusieurs gestionnaires","Appel de la méthode :
onClick1()")

#===Définition de la deuxième fonction gestionnaire d'événement
def onClick2(event):
    messagebox.showinfo("Appel de plusieurs gestionnaires","Appel de la méthodes :
onClick2()")


root = Tk()
root.geometry("300x100+300+300")
root.title("Button")

frame = Frame(root)
```

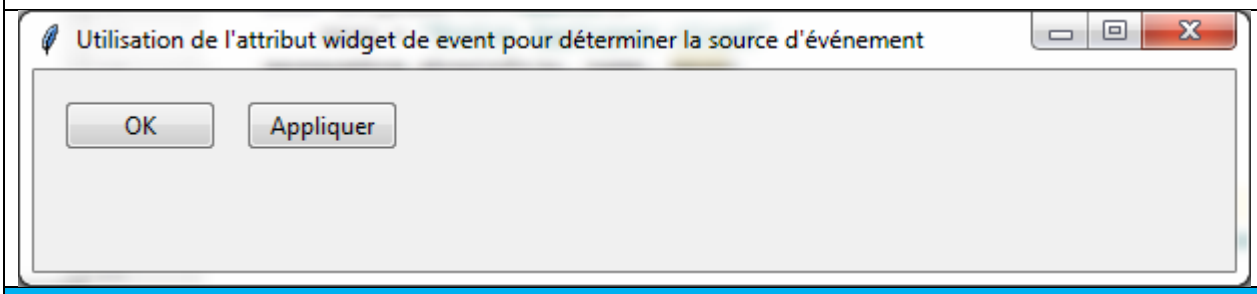
```
frame.pack(fill=BOTH, expand=True)

#====Définition d'un premier bouton'
btn1 = Button(frame, text="Button1")
btn1.grid(row=0, column=0, padx=10, pady=10)

#Liaison du 1er gestionnaire d'événement onClick1 à l'événement click
## sur le bouton gauche de la souris pour btn1
btn1.bind("<Button-1>", onClick1)

#Liaison et Ajout du 2éme gestionnaire d'événement onClick2 à l'événement click sur
le bouton gauche de la souris
btn1.bind("<Button-1>", onClick2, add="+")
#=============================================================================

btn2 = Button(frame, text="Button2")
btn2.grid(row=0, column=1, padx=10, pady=10)

#Liaison du 1er gestionnaire d'événement onClick1 à l'événement click
## sur le bouton gauche de la souris pour btn2
btn2.bind("<Button-1>", onClick1)

#Remplacement du 1er gestionnaire d'événement onClick1 par le gestionnaire onClick2
btn2.bind("<Button-1>", onClick2, add="")

root.mainloop()
```



Utilisation de l'attribut widget de l'objet event pour déterminer la source de l'événement

```
from tkinter import Tk, BOTH, StringVar, W, E
from tkinter.ttk import Frame, Button
from tkinter import messagebox


def onClick(event):
    global svar
    w = event.widget
    if (w._name == "okbtn"):
        mess = "bouton OK cliqué"
    elif (w._name == "appbtn"):
        mess = "Bouton Appliquer cliqué"
    messagebox.showinfo(w._name, mess)


root = Tk()
root.geometry("600x100+300+300")
```

```
root.title("Utilisation de l'attribut widget de event pour déterminer la source
d'événement ")

frame = Frame(root)
frame.pack(fill=BOTH, expand=True)


frame.grid_columnconfigure(2, weight=1)
frame.grid_rowconfigure(2, weight=1)

okBtn = Button(frame, text="OK", name="okbtn")
okBtn.grid(row=0, column=0, padx=15, pady=15)
okBtn.bind("<Button-1>", onClick)

applyBtn = Button(frame, text="Appliquer", name="appbtn")
applyBtn.bind("<Button-1>", onClick)
applyBtn.grid(row=0, column=1)

root.mainloop()
```



Exemple de programme d'animation

```
from PIL import Image, ImageTk
from tkinter import Tk, BOTH, CENTER
from tkinter.ttk import Frame, Label

DELAY = 850

def doCycle():
    global i
    i += 1

    if (i >= 9):
        return

    img = ImageTk.PhotoImage(Image.open(img_names[i]))
    label.configure(image=img)
    label.image = img

    frame.after(DELAY, doCycle)


root = Tk()
root.geometry("300x200+300+300")
root.title("Animation")
frame = Frame(root)
frame.pack(fill=BOTH, expand=True)

i = 0
img_names = ("one.png", "two.png", "three.png",
```

```
            "four.png", "five.png", "six.png", "seven.png",
            "eight.png", "nine.png")

img = Image.open(img_names[i])
num = ImageTk.PhotoImage(img)
label = Label(frame, image=num)
label.pack(pady=30)

# reference must be stored
label.image = num
frame.after(DELAY, doCycle)

root.mainloop()
```