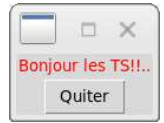


Référence

1 Présentation de tkinter de python

Le domaine des interfaces graphiques (ou GUI : *Graphical User Interface*) est complexe. Ces interfaces graphiques vont exploiter des bibliothèques de fonctions graphiques de base que l'on considère comme les composants de cette bibliothèque.

Tous ces composants sont généralement présents comme des *classes d'objets*, dont il vous faut étudier les *attributs* et les *méthodes* pour les utiliser. Avec Python, une bibliothèque graphique très utilisée est la bibliothèque *tkinter*.



1.1 Première présentation

Tkinter est un module Python spécifiques aux interfaces graphiques (GUI).

Les composants graphiques (ou widgets) correspondent à des classes d'objets dont il faudra étudier les attributs et les méthodes.

```
from tkinter import *                                #Importer toutes les classes du module
fenetre=Tk()                                          #Utilise la classe Tk() pour créer une fenêtre
text1=Label(fenetre,text='Bonjour les TS!!..',fg='red') #text1 est esclave de fenetre
text1.bouton.grid(row=1,column=1)                   #La méthode grid place le widget sur une grille.
bouton=Button(fenetre,text=('Quitter'),command=fenetre.destroy)#bouton est esclave de fenetre.
bouton.bouton.grid(row=2,column=1)
fenetre.mainloop()                                  #Démarrage du récepteur d'événements
```

J'utilise pour le placement des widgets pour les exemples simples j'utilise La méthode pack() mais pour des projets plus complets je vous encourage à utiliser la méthode grid()

2 Les widgets

2.1 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

```
from tkinter import *
fenetre=Tk()
bouton=Button(fenetre, text="Fermer", command=fenetre.destroy)
bouton.grid(row=1,column=1)
fenetre.mainloop()
```



On peut aussi placer le bouton directement

Bouton

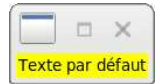
```
from tkinter import *
fenetre=Tk()
Button(fenetre, text="Fermer", command=fenetre.destroy).grid(row=1,column=1)
fenetre.mainloop()
```

2.2 Les labels

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

Label

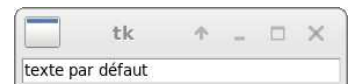
```
from tkinter import *
fenetre=Tk()
label = Label(fenetre, text="Texte par défaut", bg="yellow")
label.grid(row=1,column=1)
fenetre.mainloop()
```



2.3 Entrée

Entrée

```
from tkinter import *
fenetre=Tk()
value = StringVar()
value.set("texte par défaut")
entree = Entry(fenetre, textvariable=value, width=30)
entree.grid(row=1,column=1)
fenetre.mainloop()
```

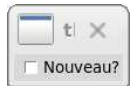


2.4 Case à cocher

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

Case à cocher

```
from tkinter import *
fenetre=Tk()
bouton = Checkbutton(fenetre, text="Nouveau?")
bouton.grid(row=1,column=1)
fenetre.mainloop()
```

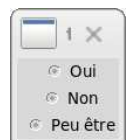


2.5 Boutons radio

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

Boutons radio

```
from tkinter import *
fenetre=Tk()
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.grid(row=1,column=1)
bouton2.grid(row=2,column=1)
bouton3.grid(row=3,column=1)
fenetre.mainloop()
```

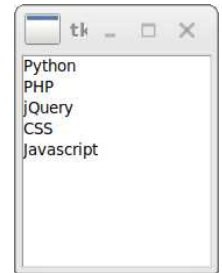


2.6 Les listes

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

Listes

```
from tkinter import *
fenetre=Tk()
liste = Listbox(fenetre)
liste.insert(1, "Python")
liste.insert(2, "PHP")
liste.insert(3, "jQuery")
liste.insert(4, "CSS")
liste.insert(5, "Javascript")
liste.grid(row=1,column=1)
fenetre.mainloop()
```

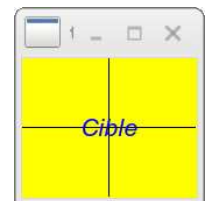


2.7 Canvas

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

Canvas

```
from tkinter import *
fenetre=Tk()
canvas = Canvas(fenetre, width=150, height=120, background='yellow')
ligne1 = canvas.create_line(75, 0, 75, 120)
ligne2 = canvas.create_line(0, 60, 150, 60)
txt = canvas.create_text(75, 60, text="Cible", font="Arial 16 italic", fill="blue")
canvas.grid(row=1,column=1)
fenetre.mainloop()
```



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 élément 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()))` ou visitez la page suivante [infohost](#)

2.8 Scale

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

Scale

```
from tkinter import *
fenetre=Tk()
value = DoubleVar()
value.set(50)
scale = Scale(fenetre, variable=value, from_=0 , to=100 )
scale.grid(row=1,column=1)
fenetre.mainloop()
```

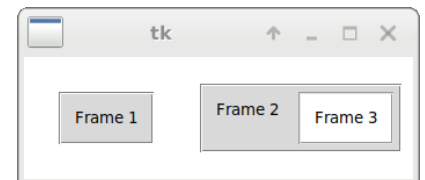


2.9 Frames

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

Frames

```
from tkinter import *
fenetre=Tk()
fenetre['bg']='white'
# frame 1
Frame1 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame1.grid(row=1,column=1,sticky="w", padx=30, pady=30)
# frame 2
Frame2 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame2.grid(row=1,column=2,sticky="w", padx=10, pady=10)
# frame 3 dans frame 2
Frame3 = Frame(Frame2, bg="white", borderwidth=2, relief=GROOVE)
Frame3.grid(row=1,column=3,sticky="e", padx=5, pady=5)
# Ajout de labels
Label(Frame1, text="Frame 1").grid(row=1,column=1,padx=10, pady=10)
Label(Frame2, text="Frame 2").grid(row=1,column=1,padx=10, pady=10)
Label(Frame3, text="Frame 3",bg="white").grid(row=1,column=1,padx=10, pady=10)
fenetre.mainloop()
```



2.10 PanedWindow

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

PanedWindow

```
from tkinter import *
fenetre=Tk()
pw = PanedWindow(fenetre, orient=HORIZONTAL)
pw.pack(side=TOP, expand=Y, fill=BOTH, pady=2, padx=2)
pw.add(Label(pw, text='Volet 1', background='blue', anchor=CENTER))
pw.add(Label(pw, text='Volet 2', background='white', anchor=CENTER) )
pw.add(Label(pw, text='Volet 3', background='red', anchor=CENTER) )
pw.grid(row=1,column=1)
fenetre.mainloop()
```

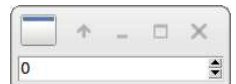


2.11 Spinbox

La spinbox propose à l'utilisateur de choisir un nombre

Spinbox

```
from tkinter import *
fenetre=Tk()
spin = Spinbox(fenetre, from_=0, to=10)
spin.grid(row=1,column=1)
fenetre.mainloop()
```

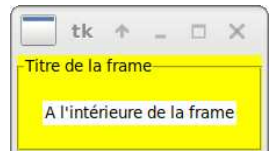


2.12 LabelFrame

Le labelframe est un cadre avec un label.

LabelFrame

```
from tkinter import *
fenetre=Tk()
labelf = LabelFrame(fenetre, text="Titre de la frame", padx=20, pady=20, bg="yellow")
labelf.grid(row=1,column=1)
label2=Label(labelf, text="A l'intérieure de la frame", bg="white")
label2.grid(row=1,column=1)
fenetre.mainloop()
```



2.13 Les alertes

Alerte

```
from tkinter import *
from tkinter.messagebox import *
fenetre=Tk()
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',width=20, command=callback).grid(row=1,column=1)
fenetre.mainloop()
```



Voici les alertes possibles :

- showinfo()
- showwarning()
- showerror()
- askquestion()
- askokcancel()
- askyesno()
- askretrycancel()

2.14 Barre de menu

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

Barre de menu

```
from tkinter import *
from tkinter.messagebox import *
fenetre=Tk()
def alert(msg):
    showinfo("alerte", msg)

menubar = Menu(fenetre)
menu1 = Menu(menubar, tearoff=0)
menu1.add_command(label="Créer", command=lambda : alert('Créer'))
menu1.add_command(label="Editer", command=lambda : alert('Éditer'))
menu1.add_separator()
menu1.add_command(label="Quitter", command=fenetre.destroy)
menubar.add_cascade(label="Fichier", menu=menu1)
menu2 = Menu(menubar, tearoff=0)
menu2.add_command(label="Couper", command=lambda : alert("Couper"))
menu2.add_command(label="Copier", command=lambda : alert("Copier"))
menu2.add_command(label="Coller", command=lambda : alert("Coller"))
menubar.add_cascade(label="Editer", menu=menu2)
menu3 = Menu(menubar, tearoff=0)
menu3.add_command(label="À propos", command=lambda : alert("À propos"))
menubar.add_cascade(label="Aide", menu=menu3)
fenetre.config(menu=menubar)
fenetre.mainloop()
```



2.15 Connaitre toutes les méthodes et options d'un widget

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

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

2.16 Les attributs standards de la méthode grid

Dans les exemple ci-dessus j'ai utiliser la méthode **grid** pour placer les widgets (il en existe d'autre comme pack).

Cette méthode enregistre le widget dans le gestionnaire de grille. Si vous ne faites pas ça, votre widget existera sans pour autant apparaître à l'écran. Voici les options possibles :

Paramètres :

- **row** : Le numéro de ligne où vous souhaitez placer votre widget en partant de zéro. Sa valeur par défaut est le numéro de de la première ligne inoccupée.
- **column** : Le numéro de la colonne où vous souhaitez placer votre widget en partant de zéro. Sa valeur par défaut est 0.
- **rowspan** : Normalement un widget occupe seulement une cellule. Cependant, vous pouvez regrouper plusieurs cellules d'une colonne en indiquant via **rowspan** le nombre de cellules à fusionner. Cette option peut-être utilisée en combinaison avec **columnspan** afin de préciser un bloc de cellules.

Par exemple, `w.grid(row=3, column=2, rowspan=4, columnspan=5)` aura pour effet de placer w dans une zone obtenue en fusionnant 20 cellules, avec les numéros de lignes 3 - 6 et les numéros de colonnes 2 - 7.

- **columnspan** : Normalement un widget occupe seulement une cellule. Cependant, vous pouvez regrouper plusieurs cellules d'une ligne en indiquant via **columnspan** le nombre de cellules à regrouper.

Par exemple, `w.grid(row=0, column=2, columnspan=3)` aura pour effet de placer le widget w dans une cellule qui s'étale sur les colonnes 2, 3 et 4 de la ligne 0.

- **ipadx** : marge interne horizontale (en x). Cette dimension est ajoutée à l'intérieur du widget sur ses côtés gauche et droit.
- **ipady** : marge interne verticale (en y). Cette dimension est ajoutée à l'intérieur du widget sur ses côtés haut et bas.
- **padx** : Marge externe horizontale.
- **pady** : Marge externe verticale.
- **sticky** : Cette option détermine la façon de distribuer l'espace inoccupé par un widget à l'intérieur d'une cellule. Il y a quatre orientations :

e pour *est* (droite), w pour *west* (gauche), n pour *north* (haut) et s pour *south* (bas).

Si vous ne donnez aucune valeur à l'attribut **sticky**, le comportement par défaut est de centrer le widget dans sa cellule.

- Vous pouvez positionner le widget dans un des coins de la cellule en indiquant `sticky='ne'` (nord-est : en haut à droite), `sticky='se'` (en bas à droite), `sticky='sw'` (en bas à gauche), ou `sticky='nw'` (en haut à gauche).
- Vous pouvez centrer le widget contre l'un des bords de la cellule en utilisant `sticky='n'` (centré en haut), `sticky='e'` (centré à droite), `sticky='s'` (centré en bas), ou `sticky='w'` (centré à gauche).
- Utilisez `sticky='ns'` pour l'étirer verticalement tout en le laissant centré horizontalement.
- Utilisez `sticky='ew'` pour l'étirer horizontalement tout en le laissant centré verticalement.
- Utilisez `sticky='nesw'` pour l'étirer dans les deux directions afin de remplir la cellule.
- Les autres combinaisons fonctionneront aussi. Par exemple, `sticky='nsw'` l'étirera verticalement en le plaçant contre le bord gauche de la cellule.

2.17 Les options de dimensions

- **height** : Hauteur du widget.
- **width** : Largeur du widget.
- **padx, pady** : Espace supplémentaire autour du widget. X pour horizontal et Y pour vertical.
- **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".

2.18 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 hexadécimal : #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.

2.19 Visualiser une image

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

PhotoImage

```
from tkinter import *
fenetre=Tk()
photo = PhotoImage(file="dessin.png")
canvas = Canvas(fenetre,width=400, height=400)
canvas.create_image(0, 0, anchor=NW, image=photo)
canvas.grid(row=1,column=1)
fenetre.mainloop()
```

2.20 askopenfilename

Le widget *askopenfilename* permet de sélectionner un fichier. Il faut au préalable importer le module *tkinter.filedialog* : Voici deux exemples.

a) Sélectionner une image et l'afficher

Ouvrir et visualiser une image *.png

```
from tkinter import *
from tkinter.filedialog import *
fenetre=Tk()
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.grid(row=1,column=1)
fenetre.mainloop()
```

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

b) Sélectionner un fichier .txt et l'afficher

Ouvrir et visualiser un fichier .txt

```
from tkinter import *
from tkinter.filedialog import *
fenetre=Tk()
```



```
filename = askopenfilename(title="Ouvrir votre document",filetypes=[('txt files','*.txt'),('allfiles','*.*')])
fichier = open(filename, "r")
content = fichier.read()
fichier.close()
Label(fenetre, text=content).grid(row=1,column=1,padx=10, pady=10)
fenetre.mainloop()
```

3 Des exemples importants

3.1 Récupérer la valeur d'un widget Entry et modifier un label

- Pour récupérer la valeur d'un input il vous faudra utiliser la méthode *get()*.
- Pour changer le texte d'un label on utilisera *label['text']*

```
from tkinter import *
from tkinter.messagebox import *
def changelabel():
    label['text']=str(entree.get())

fenetre=Tk()
value = StringVar()
value.set("Valeur")
entree = Entry(fenetre, textvariable=value, width=30)
entree.grid(row=1,column=1,columnspan=2)
bouton = Button(fenetre, text="Valider", command=changelabel)
bouton.grid(row=2,column=1,sticky='w')
label = Label(fenetre, text="Un texte")
label.grid(row=2,column=2,sticky='w')
fenetre.mainloop()
```

3.2 Paramètres dans une fonction pour un bouton

Quand on clique sur un bouton cela déclenche une fonction. Cependant si on veut utiliser un paramètre pour cette fonction, il est nécessaire d'utiliser une fonction *lambda*.

Voici un exemple pour calculer le carré d'un nombre.

```
from tkinter import *
def changelabel(x):
    label['text']=str(x**2)

fenetre=Tk()
value = StringVar()
value.set("2")
entree = Entry(fenetre, textvariable=value, width=30)
entree.grid(row=1,column=1,columnspan=2)
bouton = Button(fenetre, text="Carré", command=lambda : changelabel(int(entree.get())))
bouton.grid(row=2,column=1,sticky='w')
label = Label(fenetre, text="Un texte")
label.grid(row=2,column=2,sticky='w')
```

```
fenetre.mainloop()
```

4 Les évènements

Tous les programmes d'ordinateur comportent grosso-modo trois phases principales :

- Une phase d'initialisation, laquelle contient les instructions qui préparent le travail à effectuer (appel des modules externes nécessaires, ouverture de fichiers ...).
- Une phase centrale où l'on trouve la véritable fonctionnalité du programme.
- Une phase de clôture qui sert à clôturer « proprement » les opérations (c'est-à-dire fermer les fichiers restés ouverts etc...)

Dans le cas d'un programme qui utilise une interface graphique, le programme est ***piloté par des événements*** (déplacement de la souris, appui sur une touche du clavier etc...).

Après sa phase d'initialisation, un programme de ce type se met en quelque sorte « en attente » d'évènements.

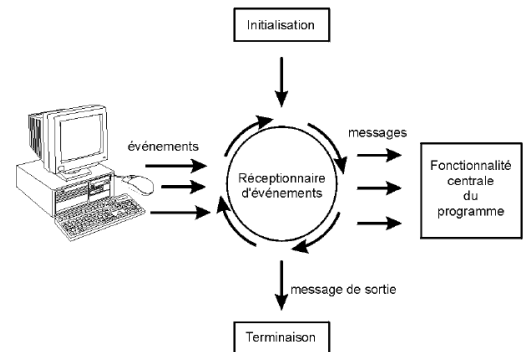
Le réceptonnaire d'évènements scrute sans cesse tous les périphériques (clavier, souris, modem, etc.) et réagit immédiatement si un événement est détecté.

Il est important de bien comprendre ici que pendant tout ce temps, le réceptonnaire continue à « tourner » et à guetter l'apparition d'autres événements éventuels.

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

Pour chaque widget, vous pouvez ***binder*** (lier en français) un événement, 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.grid(row=1,column=1)
```

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
- <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 liaison 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 appelée 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
    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.grid(row=1,column=1)
```

5 TP

Créer une interface pour calculer l'indice de la masse corporelle.

$IMC = \frac{\text{masse}}{\text{taille}^2}$ ou la masse est en *kg* et la taille en *m*

Les valeurs 18 et 25 constituent des repères communément admis pour un IMC normal et présentant donc un rapport de risque jugé acceptable.

Quand vous avez fini faire afficher en dessous un label selon l'imc obtenu.

IMC (<i>kg · m⁻²</i>)	Interprétation
moins de 16,5	dénutrition ou anorexie
16,5 à 18,5	maigreur
18,5 à 25	poids idéal
25 à 30	surpoids
30 à 35	obésité modérée
35 à 40	obésité sévère
plus de 40	obésité morbide ou massive

Calculez votre IMC

Masse en kg

Taille en cm

75

170

Cliquez!

25.95