Python – Réaliser une GUI avec le module Tkinter

La bibliothèque *Tkinter* doit être appelée pour la création d'une GUI (Graphical User Interface).

Dans la suite de la fiche, nous supposons que la ligne suivante a été insérée au début du script :

```
import tkinter as tk
```

On suppose dans l'ensemble de la présente fiche que la fenêtre `fen` *est créée.*

Fonctions principales de *Tkinter*

(Consulter le site https://docs.python.org/fr/3/library/tkinter.html pour la notice complète)

	Méthodes	Actions réalisées
Fenêtre	`fen = tk.Tk()`	Créer une fenêtre Tkinter
	`fen.title(titre_fenetre)`	Ajouter le titre `titre_fenetre` à la fenêtre
	`fen.geometry("900x800")` `fen.resizable(width=True,height=False)`	Dimension de la fenêtre *Largeur X Hauteur* (le coin supérieur gauche est l'origine du repère) Permet de rendre possible de redimensionnement de la fenêtre

Variables	`val = tk.DoubleVar`
	`val = tk.IntVar`
	`val = tk.StringVar`
		Création d'une variable de contrôle :
		✓ pour un flottant
		✓ pour un entier
		✓ pour une chaîne de caractères

Widgets	`texte = tk.Label(fen, text=texte_aff)`	Création d'un widget zone de texte sur la fenêtre `fen` contenant le texte `texte_aff`
	`bouton = tk.Button(fen, text=texte_bout, command=fonction)`	Création d'un widget bouton sur la fenêtre `fen`, ayant comme texte `texte_bout` et exécutant `fonction` lors d'un clic dessus.
	`var_recup = StringVar()` `saisie = tk.Entry(fen, textvariable = var_recup)`	Création d'une saisie utilisateur sur la fenêtre `fen` avec récupération d'une chaine de caractères dans une variable de contrôle `var_recup`
	`cadre = tk.Frame(fen, bg='#CCCCCC', relief=tk.GROOVE, borderwidth=2)`	Création d'un cadre Frame sur la fenêtre fen ayant une couleur de fond, un relief de tour et une taille de bordure
	`tk.Radiobutton(cadre, variable=var_select, text=etiq, value=val)`	Création d'un bouton radio dans le cadre `cadre` avec le texte `etiq` et la valeur `val`
	`can = tk.Canevas(fen, width=largeur, height=hauteur)`	Création d'un canevas pour créer une zone graphique (voir page 2 pour tracer d'une ligne, un ovale (et cercle), un rectangle, ajouter d'une image)

	Utiliser une des trois méthodes au choix (pas de mélange)
	`objet.pack(side=tk.TOP)`
	`objet.grid(row=0,column=1)`
	`objet.place(x=a, y=b, height=..., width=...)`
		Positionnement d'un objet sur la fenêtre :
		✓ Pack : possibilité de préciser l'alignement avec `side` (TOP, BOTTOM, LEFT, RIGHT) ou d'utiliser toute la place `expand=tk.YES`
		✓ Grid : possibilité de fusionner des lignes `rowspan` ou des colonnes `columnspan`
		✓ Place : positionne l'objet à la coordonnée (a,b)

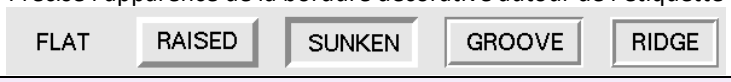
Évènements	`fen.bind("<Event>", fonction)`	Attente de l'action sur la fenêtre (clavier/souris) correspondant à l'évènement `Event` et exécution de `fonction` si l'évènement est détecté.
	`fen.mainloop()`	Lancer la boucle d'attente des évènements (obligatoire)
	`objet.after(temps, fonction)`	Relance l'action de fonction `widget` objet toutes les `temps` millisecondes

Enjoliver vos widgets

✓ Paramètres standards des widgets

Les tailles, les couleurs et les polices de caractères (fontes) ont des définitions communes à tous les widgets.

- Vous pouvez préciser les options lors de l'appel du constructeur du widget en utilisant des mots clés comme `text='A afficher'` ou `height=14`.
- Après avoir créé un widget, on peut :
  - modifier chacune de ses options en utilisant sa méthode `config()` ou la syntaxe `w['option'] = valeur`
  - récupérer la valeur courante de n'importe laquelle de ses options en utilisant sa méthode `cget()` ou l'écriture `w['option']`.

Paramètres	Action	Remarque
`width`	Largeur	Supposée être en pixels sauf si une chaîne de caractères qui contient un nombre suivi de : ✓ c : Centimètres ✓ m : Millimètres ✓ i : Pousses (Inches) ✓ p : Points d'impression
`height`	Hauteur	
`font=(tuple)`	Régler la police	Tuple dont le premier élément est la famille de la fonte, suivi par une taille (en point si positif, en pixel si négatif), optionnellement suivi par une chaîne contenant un ou plusieurs modificateurs de style (voir encadré)
`background` ou `bg`	Couleur de fond	En système hexadécimal #RRVVBB ou 'white', 'black', 'red', 'green', 'blue', 'cyan', 'yellow', and 'magenta' seront toujours disponibles
`foreground` ou `fg`	Couleur de l'étiquette	
`padx`	Marge horizontale	Espace horizontal supplémentaire à insérer à gauche et à droite
`pady`	Marge verticales	Espace vertical supplémentaire à insérer à gauche et à droite
`relief`	Donner du relief	Précise l'apparence de la bordure décorative autour de l'étiquette 

Exemple : `font=('Times', 14, 'bold'),fg='red', bg='#AAAAAA'`
→ le texte sera en Times gras de taille 14, écrit en rouge sur fond gris

Chaîne	Modificateur de style
`bold`	**Gras**
`italic`	*Italique*
`underline`	Souligné
`overstrike`	~~Barré~~

✓ Types d'objets graphiques dans un canevas

- `can.create_line(x1, y1, x2, y2)` : pour tracer une ligne entre le point (x1,y1) et (x2,y2)
- `can.create_rectangle(x1, y1, x2, y2)` : pour tracer un rectangle entre le coin (x1,y1) et (x2,y2)
- `can.create_oval(x1, y1, x2, y2)` : pour tracer une ellipse ou cercle dans le rectangle (x1,y1) et (x2,y2)
- `can.create_polygon(...)` : pour tracer un polygone. Points mis à la suite sous forme de tuples (x,y)
- `can.create_text(x, y, text=chaîne)` : pour afficher un texte
- `can.create_image(width,height,image=img)` : pour insérer une image

Ces objets possèdent des **attributs** communs : `width` = épaisseur du contour, `fill` = couleur de remplissage, `outline` = couleur de contour, `activefill` = couleur de remplissage au survol de la souris, `state` = NORMAL, DISABLED, HIDDEN (un objet peut être caché, désactivé).

Attention : le repère d'un canevas a pour origine le point haut gauche et l'axe des ordonnées est donc orienté vers le bas.

Gérer les évènements

La méthode `bind` permet de tester si un évènement se produit :

- **Une action sur la souris** : <Button-1> clic sur le bouton gauche, <DoubleButton-1> double clic sur le bouton gauche, <ButtonRelease-3> relâchement du bouton droit, <Motion> mouvement de la souris à l'intérieur d'un widget
  On peut alors récupérer par exemple la position de la souris avec event.x et event.y dans la fonction callback.
- **Une action sur le clavier** : <KeyPress-g> appui sur la touche g, <KeyRelease-C> relâchement de la touche C, <KeyPress-Escape>, <KeyPress-Return> appui sur la touche Entrée, <KeyPress-KP_Enter> appui sur la touche Entrée du clavier numérique.
- **Une action associée à un widget** <Enter>. La souris passe au-dessus du widget, …

✓ Déplacer, modifier, supprimer des widgets

Utilisables avec les objets graphiques (non exhaustif)

- `obj.coords(nom_objet, x, y)` : déplacement absolu (nouvelles coordonnées)
- `obj.move(nom_objet, dx, dy)` : déplacement relatif de `dx, dy` à partir de la position initiale
- `obj.delete(nom_objet)` : suppression d'un ou plusieurs objets
- `obj.itemconfig(nom_objet, attribut=valeur)` : modification d'une ou plusieurs propriétés
- `obj.findclosest(x, y)` : recherche l'objet le plus proche au voisinage de la position `(x, y)` et retourne un numéro qui correspond à un objet qu'on peut alors supprimer, modifier…
- La méthode `coords` permet également de déterminer les coordonnées d'un objet donné : `obj.coords(nom_objet)` renvoie les coordonnées de `nom_objet`.