Introduction à tkinter

Le module tkinter de python permet de faire des interfaces graphiques (la traduction anglaise est GUI pour *Graphical User Interface*), donc de rendre les programmes plus conviviaux. Une interface graphique est constituée de composants (widgets en anglais) qui peuvent chacun être la source d'événements.
Les widgets sont essentiellement des : zones de texte, zones de saisie, boutons, zones de dessin...
D'autres librairies souvent plus connues (comme GTK) permettent de créer des interfaces graphiques, mais la librairie tkinter a l'avantage d'être intégrée par défaut dans python et d'être assez simple à utiliser.
L'adresse de sa documentation sur Internet est http://docs.python.org/3/library/tk.html
L'adresse suivante donne des renseignements précis, en particulier sur les widgets et leurs méthodes :
http://effbot.org/tkinterbook/

I. Squelette d'un code avec tkinter

Pour utiliser tkinter, il faut déjà l'importer en plaçant en tête du fichier python :

```
1 from tkinter import *
```

Par suite, le squelette du programme qui construit et lance une interface graphique est le suivant :

```
1 fenetre=Tk()
2
3 # définition des différents widgets et des actions associées.
4
5 fenetre.mainloop()
```

Commentaires sur certaines lignes :
1 : crée la fenêtre principale appelée ici fenetre.
5 : lance le processus de collecte des événements et des actions correspondantes.

On peut configurer un certain nombre de choses, comme le titre de la fenêtre et sa couleur de fond :

```
1 fenetre.title("Fenêtre exemple")
2
3 fenetre.configure(bg='white')
```

II. Les widgets

Les widgets sont les éléments que l'on place dans une fenêtre. Il y en a de plusieurs sortes.
Chaque widget a un constructeur qui « construit » une instance du widget et est associé à diverses méthodes.
Lors de l'appel au constructeur, on indique de préférence la fenêtre à laquelle il est rattaché, puis on peut préciser certaines propriétés comme son contenu, les couleurs utilisées ou certaines longueurs.
On peut modifier ou ajouter plus tard des propriétés en utilisant la méthode **config()**.
Pour insérer un widget dans une fenêtre, on applique la méthode **pack()**.
Nous donnerons un exemple pour les labels, mais les suivants se manipulent de même manière. Un exemple plus complet sera donné ensuite.

1. Les labels

Un label sert à afficher du texte ou une image. Son constructeur est Label(...). Exemple :

```
objet=Label(fenetre,text="Texte à écrire")
objet.pack()
```

Les propriétés qui peuvent être initialisées à l'appel du constructeur, en plus de text, sont :
bg (couleur de fond), fb (couleur du texte), bd (bordure), width (largeur), height (hauteur)...

2. Les zones de saisie

Comme son nom l'indique, une zone de saisie permet à l'utilisateur de saisir du texte.
Son constructeur est Entry(...).
Les propriétés qui peuvent être initialisées à l'appel du constructeur sont : bg, fb, text, bd, width...
Ce constructeur possède les méthodes suivantes :

- **insert(pos,ch)** : insère à la position pos* le texte de la chaîne ch ;
- **delete(pos1,pos2)** : supprime le texte situé entre pos1 et pos2 ;
- **get()** : obtient le contenu de la zone de saisie, sous forme de chaîne de caractères.

* La position **END** permet d'obtenir celle du dernier caractère, **0** étant celle du premier caractère.

3. Les boutons

Son constructeur est Button(...).
Les propriétés pouvant être initialisées à l'appel du constructeur sont : bg, fb, text, bd, width...
Il y a de plus la propriété **command** qui permet de relier le bouton avec une action.
Ce constructeur possède la méthode **invoke()** qui permet d'appeler la commande associée au bouton.

4. Un exemple

Le code suivant est disponible dans un fichier déposé sur le réseau.

```
1  from tkinter import *
2
3  fenetre=Tk()
4
5  def surface(r):
6      return 3.1415*r**2
7
8  def calcul():
9    r=float(entry_rayon.get())
10   s=surface(r)
11   label_surface.config(text="Surface du disque de rayon "+str(r)+" : %.2f" % s)
12   fenetre.pack_propagate(1)
13
14 fenetre.title("Surface")
15 fenetre.configure(bg='beige',width=260,height=80)
16 fenetre.pack_propagate(0)
17 fenetre.resizable(width=FALSE , height=FALSE)
18
19 label_Rayon=Label(fenetre ,text="Rayon :",bg='beige')
20 label_Rayon.pack()
21 entry_rayon=Entry(fenetre ,width=4)
22 entry_rayon.pack()
23 bouton_calcul=Button(fenetre ,text="Calculer",bg='beige',command=calcul)
24 bouton_calcul.pack()
25 label_surface=Label(fenetre ,text="",bg='beige')
26 label_surface.pack()
27
28 fenetre.mainloop()
```

Commentaires sur certaines lignes :
16 : empêche le redimensionnement automatique selon les widgets.
17 : empêche le redimensionnement de la fenêtre par l'utilisateur.
12 : autorise (à nouveau) le redimensionnement automatique selon les widgets.

5. Exercice

En utilisant les fonctions définies lors des séances précédentes (disponibles dans un fichier sur le réseau) et en partant de l'exemple donné ci-dessus, faire une interface graphique permettant de convertir des entiers en binaire et réciproquement.

III. Placement des widgets dans l'interface

En fait, la méthode pack() permet plus de liberté sur le placement des widgets que la partie précédente ne le laisse croire. De plus, il ne s'agit pas de la seule méthode pour placer les widgets dans la fenêtre. Nous donnons ci-dessous les trois méthodes de placement.
Il est préférable d'utiliser la même méthode pour l'ensemble des objets d'une même fenêtre. Pour permettre plus de libertés, on peut créer des sous-fenêtres, qui sont des widgets appelés par le constructeur **Frame()**.

1. pack()

Comme on a pu l'observer avec l'exemple de la section précédente, cette méthode empile les objets les uns à la suite des autres, et par défaut les uns en dessous des autres.
On peut utiliser les options suivantes :

- **side**, à choisir entre RIGHT, LEFT, TOP (par défaut) ou BOTTOM ;
- **expand**, égal à True ou False (par défaut). Si cette option est à True, le widget occupera tout l'espace disponible.

```
1 lp=Label(text="du texte")
2 lp.pack(side=LEFT) # ajouter à droite (par le côté gauche)
```

2. grid()

Cette méthode organise les objets dans une grille. On indique alors la ligne et la colonne dans laquelle doit se situer le widget.
Quelques options :

- **column**, indique la colonne ;
- **row**, indique la ligne ;
- **rowspan** et **columnspan**, indiquent les nombres de lignes et de colonnes occupées par l'objet ;
- **sticky**, si précisé, permet d'aligner l'objet sur un ou plusieurs bords. Par défaut l'objet est centré dans la cellule. Les valeurs sont "N", "S", "W", "E", "N+S" et "E+W".

```
1 l=Label(text="ligne 0 colonne 0 ")
2 l.grid(column=0,row=0)
3 s=Entry(width=5)
4 s.grid(column=0,row=1,sticky="E")
5 l2=Label(text="ligne 1 colonne 1 ")
6 l2.grid(column=1,row=1)
7 b=Button(text="bouton 1 ")
8 b.grid(column=0,row=2,columnspan=2)
```

Remarque : Le numéro de ligne ou de colonne commence à 0.

3. place()

Cette méthode place les éléments à une position définie par des coordonnées x et y en pixels.

```
1 L=Label(text="du texte")
2 L.place(x=10,y=50)
```

4. Enlever un widget

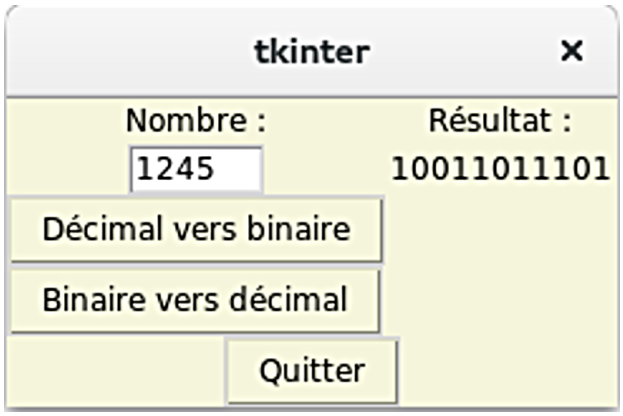
Tout objet placé avec l'une des trois méthodes peut être effacé de la fenêtre qui le contient à l'aide des méthodes respectives :
pack_forget(), **grid_forget()** et **place_forget()**.

```
1 L.place_forget()
```

Cela n'empêche pas de les faire réapparaître avec la méthode de placement, car le widget n'est pas supprimé.

5. Exercice

Modifier l'interface graphique du programme précédent pour qu'elle ait l'apparence ci-dessous :



Pour le bouton « Quitter », la commande associée sera **fenetre.destroy**, la méthode prédéfinie **destroy** permettant de fermer la fenêtre.

IV. Gestion des événements

Tout widget peut être la cible d'événements (déplacement de souris, clic sur un bouton, appui d'une touche du clavier). Il est possible de lier ces événements à des fonctions. On utilise pour cela la méthode **bind()**. Son utilisation est la suivante :
obj.bind(evt,fct)

- **obj** : nom de l'objet (widget) qui intercepte l'événement **evt**, dont les valeurs sont données plus bas ;
- **fct** : fonction appelée lorsque l'événement survient.

Les événements donnés à **evt** sont :

- "<**Key**>" : n'importe quelle touche du clavier ;
- "<**a**>", ..., "<**Return**>" : le code d'une touche particulière ;
- "<**Button-i**>" : Le bouton 'i' de la souris (i étant 1,2 ou 3) pressé ;
- "<**ButtonRelease-i**>" : le bouton 'i' de la souris relâché ;
- "<**Double-Button-i**>" : le bouton 'i' de la souris double-cliqué ;
- "<**Motion**>" : la souris de déplace ;
- "<**Enter**>" ou "<**Leave**>" : la souris entre ou sort de la zone (du widget) ;

La fonction appelée prend nécessairement un paramètre, qui est de type **Event**. Celui-ci permet d'obtenir des informations précises sur l'événement. Voici les attributs de ce paramètre :

- **char** : code usuel de la touche pressée (sauf ctrl, alt...) ;
- **keysym** : code système de la touche pressée (y compris ctrl, alt...) ;
- **widget** : nom de l'objet qui a reçut l'événement ;
- **x**, **y** : coordonnées du pointeur de la souris, l'origine étant le coin en haut à gauche de l'objet ;
- **x_root**, **y_root** : coordonnées du pointeur, à partir du coin supérieur gauche de l'écran ;

L'exemple suivant est très pratique pour obtenir les codes des touches :

```
1  from tkinter import *
2
3  def affiche_touche_pressee(evt):
4    print("----- touche  pressee -----")
5    print("evt.keysym  =  ",  evt.keysym)
6    print("evt.char  =  ",evt.char)
7    print("evt.num  =  ",evt.num)
8    print("evt.x,evt.y  =  ",evt.x, ",",evt.y)
9    print("evt.x_root,evt.y_root  =  ",evt.x_root,",",evt.y_root)
10   print("evt.widget  =  ",evt.widget)
11
12 root = Tk()
13 b = Button(text="appuyer sur une touche")
14 b.pack()
15
16 b.bind("<Key>",affiche_touche_pressee)
17 b.bind("<Button>",affiche_touche_pressee)
18 b.bind("<Motion>",affiche_touche_pressee)
19 b.focus_set()
20
21 root.mainloop()
```

Commentaires sur certaines lignes :
19 : Indique que le bouton doit recevoir le focus, autrement dit le rend actif, prêt à recevoir les événements.

Remarque Si la fonction à appliquer n'a pas besoin d'utiliser l'argument evt, et si l'on a déjà au préalable défini la fonction (par exemple s'il s'agit de la même qu'une commande exécutée par un bouton), on peut utiliser la syntaxe suivante :

```
1  widget.bind("<Key>",lambda evt:command())
```

En particulier, pour permettre la fermeture du programme en appuyant sur la touche 'Echap' :

```
1  widget.bind("<Escape>",lambda evt:fenetre.destroy())
```

1. Exercice

Améliorer encore le programme de conversion en permettant d'appuyer sur des touches pour obtenir une conversion.
C'est dans la zone de saisie que l'événement devra être détecté, les touches choisies pour lancer les conversions devant être muettes, c'est à dire ne pas correspondre à un caractère texte, comme « Entrée », « shift »...

Pour aller plus loin

On pourra améliorer encore le programme, par exemple et dans l'ordre :

- Tester la validité des entrées avant d'appliquer les fonctions de conversion, afin d'éviter les erreurs ;
- Ne mettre qu'un bouton indiquant la conversion choisie qui, à chaque clic, permet de changer le type de conversion, le texte du bouton étant mis à jour ;
- Modifier le programme pour permettre des conversions entre toutes les bases de 2 à 10 voire 2 à 16.