NSI : Interfaces graphiques

Interfaces graphiques
Principe

Elles permettent de rendre les programmes plus conviviaux.

Une interface graphique est constituée de composants, appelés **widgets**, qui sont la source d'événements.

Le déroulement d'un programme avec interface graphique est différent des programmes vus jusqu'à présent.
On a :
♦ des **widgets** : boutons, zones de saisie, labels, zones de dessin, boutons radio, listes déroulantes …
♦ des **événements** : clic sur bouton, appui sur une touche, saisie d'un texte …

Les librairies

Il existe plusieurs librairires (modules) en Python pour réaliser des interfaces graphiques :

- tkinter
- wxPython
- pygame
- PyQT, PyGTX …

Nous étudierons tkinter

La fenêtre racine

Voici le squelette d'un programme utilisant une interface graphique : il est indispensable de le respecter !

```
from tkinter import*

fen=Tk()

#Ajout des widgets

fen.mainloop()
```

Importation de la bibliothèque tkinter

Création de la fenêtre principale. On affecte l'instance Tk() dans la variable fen

Affichage de la fenêtre principale. A écrire à la fin du programme

Quelques attributs de la fenêtre principale que l'on peut modifier :

- Son nom : pour la renommer on écrit fen.title("…..")
- Sa taille : pour la redimensionner (en pixel) on écrit fen.geometry("largeur x longueur")

Principe de création des widgets

Lorsque l'on crée une interface graphique avec tkinter, on va créer puis placer des widgets dans une fenêtre graphique.
Cela se fait donc en deux étapes :

- Création du widget (Label, Entry, Button, …)
- Placement du widget avec la méthode appropriée (.pack(), .grid(), .place())

Les labels : la classe Label

Ils sont utilisés pour afficher des textes et des images.
Voici quelques-uns des attributs de la classe Label

text	Texte du Label
bitmap	Image bmp du label
image	Image gif du label
bg	Couleur de fond
fg	Couleur du texte
bd	Largeur de bordure
width	Largeur en pixel
height	Hauteur en pixel
relief	À choisir entre RAISED, FLAT, SUNKEN, GROOVE, RIDGE,SOLID
font	Police de caractère (Arial, Helvetica, Verdana, Times New Roman …), taille de police, style (normal, bold, italic, bold italic …) Exemple : font=("Arial",20,"italic")

Une première méthode de placement : la méthode .pack()

La méthode .pack() permet d'empiler les objets les uns en dessous des autres.

Quelques options existent :

side	à choisir entre RIGHT, LEFT, TOP (par défaut) ou BOTTOM
expand	à choisir entre True et False (par défaut). Si cette option est vraie, l'objet occupe tout l'espace.
fill	à choisir entre X et Y : occupe toute la largeur ou toute la hauteur

Un premier exemple : fichier ex_Label.py

```
from tkinter import*

fen=Tk()

obj1=Label(fen,text="Zone de texte",bg="blue",fg="yellow",bd=3)
obj1.pack()

obj2=Label(fen,bitmap="question",bg="red",fg="grey",bd=11,relief=GROOVE)
obj2.pack()

fen.mainloop()
```

On précise la fenêtre dans laquelle on place le Label. Ce n'est pas indispensable s'il n'y a qu'une fenêtre mais ça le sera si on travaille avec plusieurs sous fenêtres.

Cette image est prédéfinie. Les bitmap disponibles sont "error", "gray75", "gray50", "gray25", "gray12", "hourglass", "info", "questhead", "question", "warning"

On remarque bien qu'à chaque fois qu'on crée un Label, on le place.
Modifie les attributs des Labels pour voir ce qui se passe.
Modifie les options de placements (par exemple obj1.pack(side=LEFT) puis ob1.pack(fill=X)

Les boutons : la classe Button

Ils permettent de lier le clic sur le bouton à une action.
Voici quelques attributs de la classe Button :

bg, fg, bd, text	Voir classe Label
image	Image sur le bouton (attention l'objet doit être dans la classe PhotoImage (voir ci-dessous)
state	NORMAL (par défaut le bouton est actif), DISABLED (pour rendre le bouton inactif)
command	Indispensable pour associer la procédure qui va définir l'action liée au clic sur le bouton

Pour importer une image :

Image=PhotoImage(file="nom de l'image avec son extension")
On peut importer des gif, des pgm ou des ppm.
Pour importer d'autres types d'images il faut utiliser la bibliothèque PIL

Exemple d'utilisation : fichier ex_Button.py

```
from tkinter import*

#Fonction
def plus():
    global nb
    nb=nb+1
    label.config(text=nb)

#Programme principal
fen=Tk()

nb=0

label=Label(fen,text=nb)
label.pack()

bouton=Button(fen,text="Incrémenter",command=plus)
bouton.pack()

fen.mainloop()
```

On définit la procédure appelée par le clic sur le bouton. Cette procédure ne comporte pas de paramètre.

Cette instruction permet que la valeur de nb soit la même que celle du programme principal. Ainsi si elle est incrémentée dans la fonction, elle l'est aussi dans le programme principal

Même si la variable nb est global, le texte du Label ne se met pas à jour automatiquement. On utilise la méthode config qui permet de modifier les attributs du Label, ici le texte.

On appelle la procédure (sans paramètres et sans parenthèses)

Une méthode particulière pour éviter d'avoir recours à la méthode .config

En tkinter on a un type de variables spécifique (les variables de contrôle) de type IntVar(), StringVar() ou DoubleVar(). Pour récupérer leur valeur on utilise la méthode .get() et pour modifier leur valeur on utilise la méthode .set(). Ceci évite l'utilisation de variables globales. On peut alors modifier le programme précédent de la manière suivante :

```
from tkinter import*

#Fonction
def plus():
    nb.set(nb.get()+1)
#Programme principal
fen=Tk()
nb=IntVar()
nb.set(0)
label=Label(fen,textvariable=nb)
label.pack()
bouton=Button(fen,text="Incrémenter",command=plus)
bouton.pack()
fen.mainloop()
```

On récupère la valeur de nb, on lui ajoute 1 et on modifie la valeur de nb.

On déclare la variable entière nb et on l'initialise à 0

On utilise textvariable pour signifier que le texte contient une variable

Les zones de saisie : la classe Entry

Elles sont utilisées pour la saisie de texte ou de valeurs au clavier par l'utilisateur.
Quelques attributs de la classe Entry :

bg, fg, bd	Voir classe Label
width	Largeur en caractères de la zone de saisie

Quelques méthodes liées à la classe Entry :

.get()	Permet d'obtenir le contenu de la zone de saisie. **Renvoie une chaîne de caractères**
.insert(pos,ch)	Insère à la position pos, la chaîne de caractère ch
.delete(pos1,pos2)	Supprime le texte entre pos1 et pos2

Exemple d'utilisation : fichier ex_Entry.py

```
from tkinter import*

#Fonction
def affichage():
    ch=zone.get()
    label.config(text=ch)
    zone.delete(0,len(ch))

#Programme principal
fen=Tk()

label=Label(fen,text="Entrer un texte")
label.pack()

zone=Entry(fen,bg="white",fg="blue",width=55)
zone.pack()

bouton=Button(fen,text="Valider",command=affichage)
bouton.pack()

fen.mainloop()
```

On récupère le contenu de la zone de saisie

On modifie le texte du label avec le méthode .config()

On efface toute la zone de saisie. ch contient la chaîne de caractères récupérée dans zone donc len(ch) contient bien le nombre de caractères présents dans cette zone.

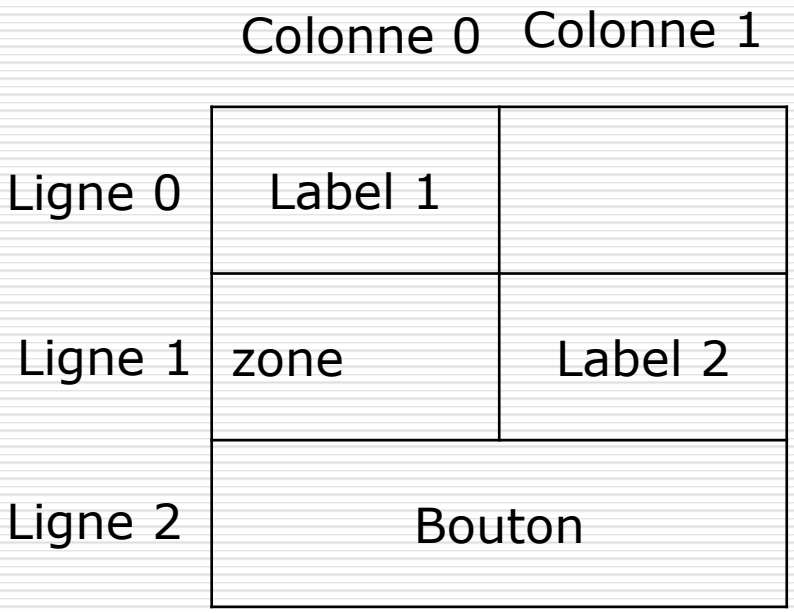
La méthode de placement .grid()

Cette méthode organise les objets dans une grille dont chaque cellule peut recevoir un objet. Il faut indiquer la colonne et la ligne.
Attention la taille des colonnes et des lignes se définit automatiquement par rapport aux objets qui y sont placés.
Quelques options :

column (indispensable)	Colonne dans laquelle sera placé l'objet (numérotée à partir de 0)
row (indispensable)	Ligne à laquelle sera place l'objet (numérotée à partir de 0)
rowspan, columnspan	Nombre de lignes, de colonnes que doit occuper l'objet
sticky	Par défaut l'objet est centré dans la cellule mais il est possible de l'aligner sur un ou plusieurs bords : "N", "S", "W", "E", "N+E"
padx, pady	Espacement autour du widget

Exemple d'utilisation : fichier ex_grid.py

```
from tkinter import*

#Programme principal
fen=Tk()

label1=Label(text="Label 1",width=20)
label1.grid(column=0,row=0)

zone=Entry(width=5)
zone.grid(column=0,row=1,sticky="W")

label2=Label(text="Label 2",bd=2,relief=SOLID)
label2.grid(column=1,row=1)

bouton=Button(text="Bouton")
bouton.grid(column=0,row=2,columnspan=2,pady=8)

fen.mainloop()
```



La méthode de placement .place()

Elle permet de placer les objets à une position définie par ses coordonnées. Le point de coordonnées (0,0) est en haut à gauche de la fenêtre et les coordonnées sont calculées positivement.

On écrit : widget.place(x=…,y=…)

Je déconseille cette méthode qui est particulièrement difficile à gérer.

REMARQUE :

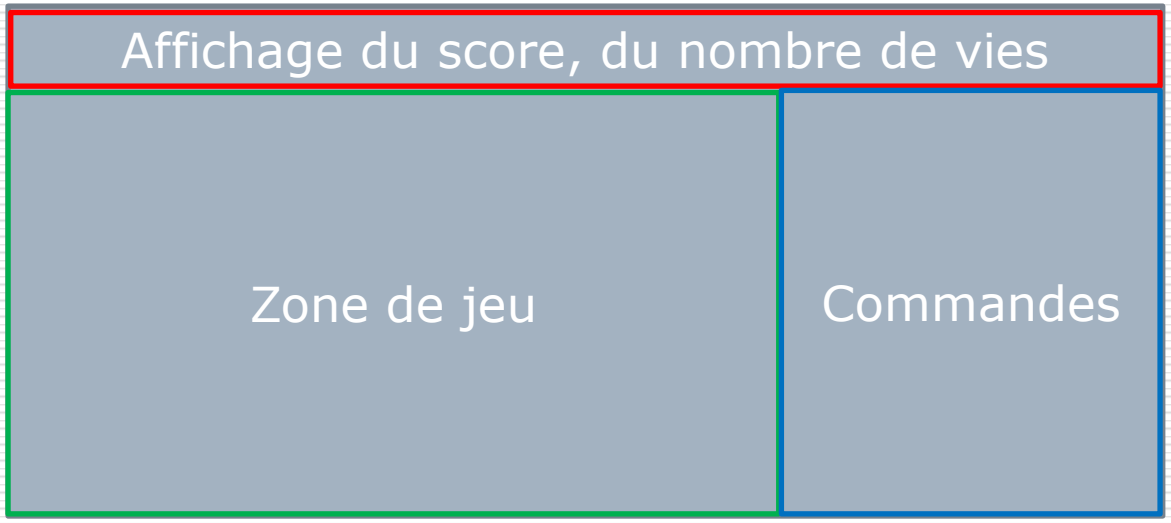
Dans une même fenêtre ou sous-fenêtre on n'utilise qu'une méthode de placement.

Quelques méthodes supplémentaires

widget.destroy	Pour détruire un widget ou la fenêtre
widget.pack_forget() widget.grid_forget() widget.place_forget()	Pour ne plus afficher un widget. Cette méthode (qui dépend de la manière dont on a placé le widget) permet de ne pas le détruire. Il est alors éventuellement possible de le réafficher plus tard.
widget.config (…)	Pour modifier l'attribut d'un widget

Les sous-fenêtres : la classe Frame

On peut regrouper les objets dans des sous-fenêtres avec des organisations indépendantes et les imbriquer selon une arborescence.
A l'intérieur d'une même sous-fenêtre il faut utiliser une seule méthode de placement.
Mais on peut utiliser une autre méthode pour placer les fenêtres les unes par rapport aux autres.
Image une fenêtre de jeu avec plusieurs sous fenêtres



```
from tkinter import*

#Fonction
def jouer():
    zoneJeu.config(text="Jouons")

#Programme principal
fen=Tk()

stat=Frame(fen,relief=SOLID,bd=2,bg="red")
labelVie=Label(stat,text="Nombre de vies")
labelVie.grid(row=0,column=0)
labelScore=Label(stat,text="Score")
labelScore.grid(row=0,column=1)
stat.grid(row=0,column=0,columnspan=2)

zoneJeu=Label(fen,text="Zone de jeu")
zoneJeu.grid(row=1,column=0)

commandes=Frame(fen,relief=SOLID,bd=1)
b1=Button(commandes,text="Jouer",command=jouer)
b1.pack()
b2=Button(commandes,text="Quitter",command=fen.destroy)
b2.pack()
commandes.grid(row=1,column=1)

fen.mainloop()
```

Exemple d'utilisation : fichier ex_Frame.py

Les zones de dessin : la classe Canvas

Le widget Canvas est une zone de dessin rectangulaire.
L'angle en haut à gauche est l'origine des coordonnées qui sont ensuite calculées positivement.
Voici quelques attributs :

height	Hauteur du Canvas
width	Largeur du Canvas
bg	Couleur de fond
bd	Taille de la bordure
relief	Style de bordure (voir attributs de la classe Label)

Quelques méthodes de la classe Canvas

.create_line(xd,yd,xa,ya)	Pour créer un segment : on donne en paramètres les coordonnées du point de départ et du point d'arrivée
.create_oval(xd,yd,xa,ya)	Pour créer un cercle ou une ellipse : on donne en paramètres les coordonnées des sommets opposés du rectangle qui contient l'ellipse.
.create_rectangle(xd,yd,xa,ya)	Pour créer un rectangle. On donne en paramètres les coordonnées des sommets opposés
.create_texte(x,y,text="…")	(x,y) sont les coordonnées du centre du texte.
.create_image(x,y,image="nom du fichier")	(x,y) sont les coordonnées du centre de l'image.

Quelques attributs et méthodes

fill	Couleur de remplissage de l'objet
width	Épaisseur du trait de contour
outline	Couleur de la ligne de contour

On peut modifier un attribut d’un objet dans un canevas de la manière suivante :
canvas.itemconfigure(objet,attribut=…)

Un exemple de canvas : fichier ex_Canvas

```
from tkinter import*

fen=Tk()

zone_dessin=Canvas(fen,width=200,height=200)
zone_dessin.pack()

zone_dessin.create_line(0,0,200,200,fill="violet",width=2)
zone_dessin.create_line(0,200,200,0,fill="violet",width=2)

rec=zone_dessin.create_rectangle(50,50,150,150,fill="red")

cercle=zone_dessin.create_oval(75,75,125,125,fill="yellow",outline="green",width=3)

texte=zone_dessin.create_text(100,100,text="Vive la\n  NSI",fill="blue")

fen.mainloop()
```

Un deuxième exemple avec une image : ficher ex_Canvas2.py

```
from tkinter import *

fen=Tk()

photo=PhotoImage(file="andromede2.gif")

fen.geometry("800x600")
fen.title("800x600")

fond=Canvas(fen, bg='blue',width=600,height=300)
fond.pack()

for i in range(1,6):
        fond.create_line(i*100,0,i*100,300)
for i in range(1,3):
        fond.create_line(0,i*100,600,i*100)

img=fond.create_image(400,100,image=photo)

fen.mainloop()
```