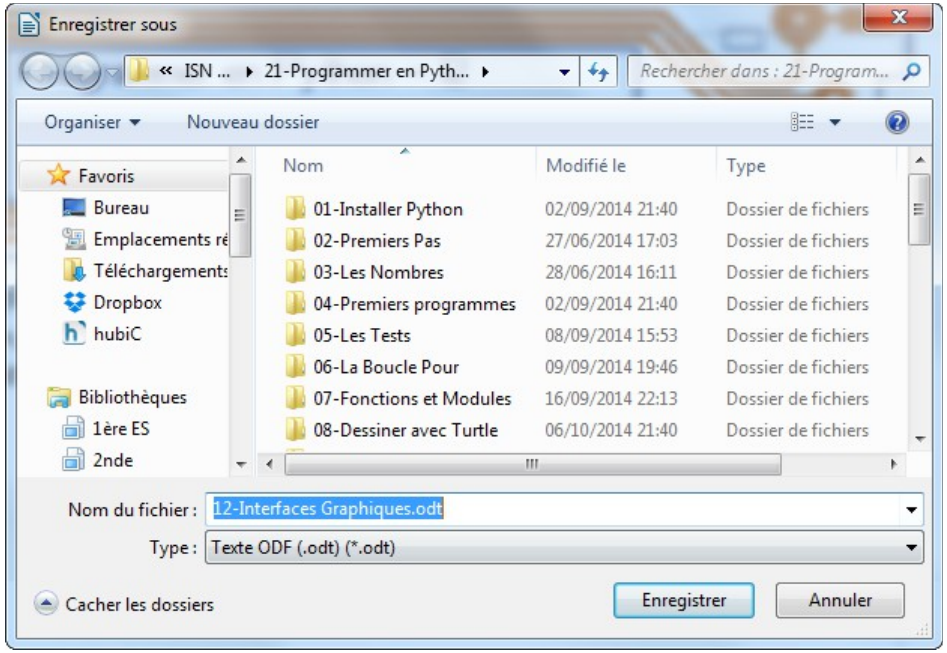
Programmer en Python – 12

Interfaces Graphiques

On n'imagine plus au XXI[ème] siècle un programme sans interface graphique. Pour faire « simple », créer une interface graphique, c'est créer une fenêtre graphique et placer à l'intérieur des éléments sur lesquels l'utilisateur pour agir.

La fenêtre suivante s'ouvre pour enregistrer un fichier dans LibreOffice :



On peut constater que cette fenêtre est décomposée en plusieurs parties :

- Des boutons sur lesquels cliquer.
- Des zones de "dessins" qui permettent ici de parcourir l'arborescence.
- Des zones d'affichage de texte.
- Des zones qui permettent d'entrer un texte.
- Des zones avec une liste déroulante.
- etc...

Le module tkinter

tkinter est un module de « gadgets graphiques » (*widgets*) intégré par défaut dans Python. On utilisera cette bibliothèque graphique en raison de sa simplicité. Il existe aussi d'autres bibliothèques de ce type (**wxPython**, **pyQT**, **pygame**...) mais elle seront à découvrir de manière autonome.

Un manuel de référence sur **tkinter** (*168 pages en anglais !*) est disponible en cliquant sur ce lien.

Le principe général est :

- de créer une fenêtre graphique (*cf. ci-contre*)
- puis de placer et organiser des éléments (*boutons, textes, zones de dessin, etc...*) à l'intérieur de cette fenêtre.

Exemple n°12A : *Création de la fenêtre principale.*

Le programme ci-dessous est disponible dans le dossier **[ISN]**, à l'intérieur du répertoire **[12-Interfaces Graphiques]** sous le nom **Exemple12A.py**.

```
1 ##-----Importation des Modules-----##
2 from tkinter import *
3
4 ##-----Création de la fenêtre-----##
5 fen = Tk()                          ## Stockée dans la variable "fen"
6
7
8 ##-----Programme principal-----##
9 fen.mainloop()                      ## Boucle d'attente des événements
```

Ce script affiche une fenêtre « vide », de dimensions modifiables à loisir. Les trois boutons en haut à droite de la fenêtre sont actifs.

La fenêtre stockée dans la variable **fen** est un **objet** informatique. Pour modifier (*manipuler*) cet **objet**, il faut lui appliquer une **méthode** selon la syntaxe :

objet.methode(*parametres eventuels*).

a) Quelle méthode est-il indispensable d'appliquer à une fenêtre ?

.....

b) Ligne **6**, appliquer la méthode **.title()** à la fenêtre pour lui donner un titre sous forme de chaîne de caractères.

c) Appliquer la méthode **.geometry('500x300+50+10')** à la fenêtre. Comment cette méthode agit-elle sur la fenêtre ?

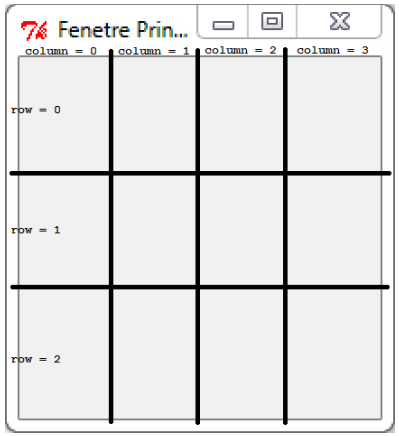
Organiser la fenêtre

Une fenêtre est tout simplement un rectangle affiché à l'écran. A l'intérieur de cette fenêtre, on place d'autres objets graphiques (***widgets***) après les avoir affectés à des variables. **Il existe de très nombreux objets graphiques**, on se limitera à :

Widget	Description
`Label()`	Zone d'affichage de texte.
`Entry()`	« Formulaire » qui permet de récupérer du texte entré par l'utilisateur
`Button()`	Un bouton sur lequel cliquer.
`Canvas()`	Canevas très « souple » qui correspond à une zone de dessin ou d'animation. Ce sera le widget le plus utilisé.

Une fois les widgets créés, il faut **anticiper leur positionnement dans la fenêtre** grâce à la méthode `.grid()` appliquée au widget :

```
nom_widget.grid(row=…, column=…)
```

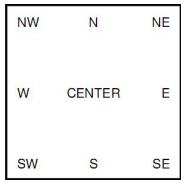


La fenêtre est alors « partagée » virtuellement (*comme la feuille de calcul d'un tableur*) en lignes (`row = a`) et colonnes (`column = b`) numérotées à partir de `0`.

On obtient ainsi des « cellules » dans lesquelles placer les différents widgets.

La méthode `.grid()` accepte des paramètres optionnels qui permettent...

- ...de positionner le widget dans la cellule si celle-ci est trop grande pour lui avec l'attribut :
  `sticky = position`



- ...de fusionner plusieurs cellules à partir de la cellule dans laquelle le widget est positionné.
  L'attribut `rowspan = n` fusionne `n` cellules verticalement
  L'attribut `columnspan = n` fusionne `n` cellules horizontalement

La disposition des différents widgets dans une fenêtre doit être décidée AVANT de programmer ces widgets. **Faire des croquis de l'apparence désirée à l'aide d'un papier et d'un crayon est indispensable.**

Bouton

L'instruction :

```
nom_bouton = Button(nom_parent, text='…', command=fonction)
```

affecte un **widget** `Button()` à la variable intitulée `nom_bouton`.
Ce bouton se place dans le widget `nom_parent` (*généralement une fenêtre*).
Le texte … sera affiché sur le bouton et le clic lancera la fonction `fonction`.

Exemple n°12B : *Le bouton « Quitter ».*

Reprendre le fichier de l'`Exemple12A.py`.

a) Sauter plusieurs lignes à partir de la ligne `8` puis insérer les suivantes :

```
9  ##-----Création des boutons-----##
10 bouton_quitter = Button(fen, text='Quitter', command=fen.quit)
```

b) Lancer le programme. Le bouton s'affiche-t-il ? Pourquoi ?
Est-ce toujours le cas en lui appliquant la méthode `.grid()` ?

c) Une fois le bouton apparu, cliquer dessus. Que se passe-t-il ?
Il faut associer une fonction à la commande de fermeture de la fenêtre.
Pour cela, en dernière ligne du programme, saisir l'instruction :

```
15 fen.destroy()                ## Fermeture "correcte" de la fenêtre
```

Zone de texte

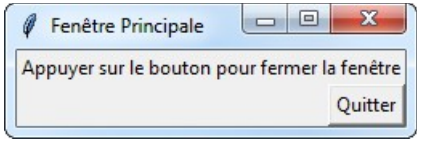
L'instruction :

```
nom_zone = Label(nom_parent, text='BlaBla')
```

affecte à la variable intitulée `nom_zone` le **widget** `Label()`.
La zone de texte se place dans le widget `nom_parent` (*une fenêtre*).

Exemple n°12C : *Création de la fenêtre principale.*

Améliorer l'`Exemple12B.py` afin d'obtenir une fenêtre ayant l'apparence ci-contre.



- La zone de texte sera affectée à la variable `texte_aide`.
- Attention au placement de la zone de texte et du bouton.
  Ne pas oublier les numéros de ligne et de colonne ni l'option `sticky`.

Formulaire

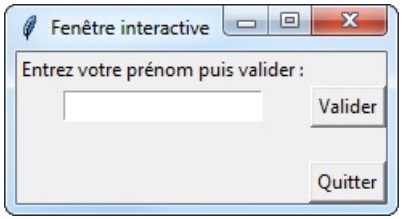
L'instruction :

```
nom_form = Entry(nom_parent, textvariable=StringVar())
```

affecte à la variable intitulée **nom_form** le **widget** **Entry()**.
Le formulaire se place dans le widget **parent** (*généralement une fenêtre*).
On peut récupérer le texte entré par l'utilisateur avec la méthode **.get()**.

Exemple n°12D : *Fenêtre « interactive ».*

Ouvrir le fichier **Exemple12D.py** et vérifier que l'on obtient bien la fenêtre ci-contre.



a) A quoi sert la méthode **.configure()** utilisée dans la fonction **prenom()** ?

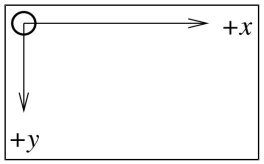
.....
.....
.....

b) Améliorer la disposition des widgets dans la fenêtre pour aboutir à une présentation originale en utilisant des options de la méthode **.grid()** :

- **rowspan** et **columnspan** ;
- **padx** et **pady** : écart (*en pixels*) entre le widget et le bord de sa « cellule » (*gauche et droite puis haut et bas*).

Canevas

Les canevas **Canvas()** sont les **widgets** les plus « souples » du module **tkinter**.
Ce sont des rectangles orientés par un repère d'origine le coin supérieur gauche et dans lesquels on peut :



- Afficher des images et du texte ;
- Tracer des dessins (*pixelisés et en 2D*) ;
- Animer des figures.

L'instruction :

```
nom_canevas = Canvas(nom_parent, width=a, height=b)
```

affecte à la variable intitulée **nom_canevas** le **widget** **Canvas()**.
Il aura pour **dimensions intérieures** **a** pixels en largeur et **b** pixels en hauteur.

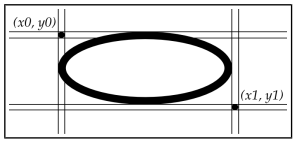
Tracer dans un canevas

Voici quelques méthodes à appliquer dans un canevas intitulé **dessin**.

- **Tracer un segment** entre les points $A(x_0 ; y_0)$ et $B(x_1 ; y_1)$.

```
dessin.create_line(x0, y0, x1, y1, [option = …])
```

- **Tracer un ovale** entre les points de coordonnées $(x_0 ; y_0)$ et $(x_1 ; y_1)$.




```
dessin.create_oval(x0, y0, x1, y1, [option = …])
```

La colonne de pixels d'abscisses x_1 n'est pas atteinte.
La ligne de pixels d'ordonnées y_1 n'est pas atteinte.

- **Tracer un rectangle** de côtés parallèles aux axes et ayant pour sommets opposés les points de coordonnées $(x_0 ; y_0)$ et $(x_1 ; y_1)$.

```
dessin.create_rectangle(x0, y0, x1, y1, [option = …])
```

Comme pour les ovales, la colonne de pixels d'abscisses x_1 n'est pas atteinte.
la ligne de pixels d'ordonnées y_1 n'est pas atteinte.

- **Incorporer une image** chargée grâce à un lien relatif (*cf. HTML*).
Le coin supérieur gauche aura pour coordonnées $(x_0 ; y_0)$ et $(x_1 ; y_1)$.

```
dessin.create_image(x0, y0, image = PhotoImage(file = 'ma_photo.jpg'))
```

- **Supprimer une figure** affichée dans le canevas.
Pour cela, la figure doit avoir été affectée à une variable.

```
ligne = dessin.create_line(x0, y0, x1, y1)
dessin.delete(ligne)
```

On **supprime toutes les figures** grâce à l'instruction : **dessin.delete(ALL)**.

Méthodes universelles

- Récupérer sous forme d'entier la hauteur actuelle (*px*) d'un widget :

```
h = nom_widget.winfo_reqheight()
```

- Récupérer sous forme d'entier la largeur actuelle (*px*) d'un widget :

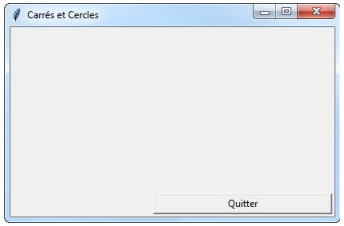
```
l = nom_widget.winfo_reqwidth()
```

- Supprimer un widget (*et tous ses widgets enfants*).

```
nom_widget.destroy()
```

S'approprier les canevas

Exemple n°12E : *Carrés et cercles.*



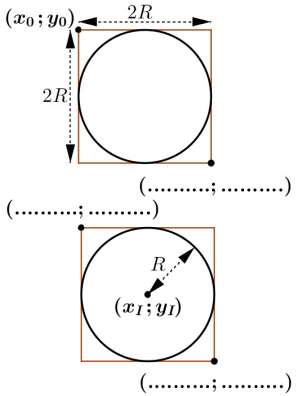
La fenêtre ouverte par le fichier **`Exemple12E.py`** contient un canevas et un bouton [*Quitter*].

a) Le canevas ne se distingue pas du fond de la fenêtre. Changer sa couleur de fond en utilisant l'option **`bg='white'`**.

b) En utilisant **`.create_rectangle()`**, encadrer le canevas par un rectangle de mêmes dimensions que le canevas. Les coordonnées (**`x0`** ; **`y0`**) seront (**`0`** ; **`0`**). Que constate-t-on ? Modifier les coordonnées (**`x0`** ; **`y0`**) et (**`x1`** ; **`y1`**) de sorte que le rectangle entoure parfaitement le canevas.

c) Tracer un carré dans le canevas : ses bords sont noirs et son fond est blanc.
Le paramètre **`width=n`** permet de tracer des bords de n pixels.
Le paramètre **`outline='#rrggbb'`** ou **`outline='nom_de_couleur'`** permet d'obtenir des bords de la couleur désirée.
`fill='#rrggbb'` ou **`fill='nom_de_couleur'`** colorie le fond.
Utiliser ces options pour personnaliser le carré.

d) **`.create_oval()`** permet de tracer un cercle en l'inséran t dans un carré de côté $2R$.
Compléter les pointillés. En déduire l'instruction permettant de tracer le cercle ci-contre :

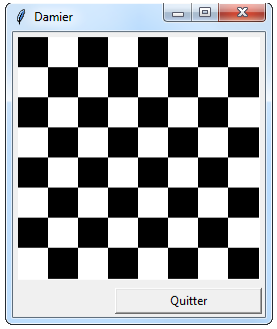


.....

e) Un cercle peut aussi être défini par son centre I (**`xI`** ; **`yI`**) et son rayon **`R`**. Compléter la 2nde figure. En déduire l'instruction permettant de tracer le cercle ci-contre :

.....

Exemple n°12F : *Une liste de carrés.*



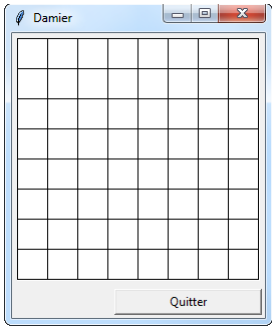
Le quadrillage ci-contre rentre parfaitement dans le canevas. Pour cela, on a défini comme variables la longueur d'un côté d'une case et le nombre de cases par ligne et par colonne.

a) Le programme du fichier **`Exemple12F.py`** est incomplet. Compléter les paramètres **`width`** et **`height`** du canevas **`dessin`** à la ligne **`22`**.

Les carrés constituant les cases sont stockés dans une liste de listes. Cette liste est obtenue en utilisant deux boucles **`for`** imbriquées :

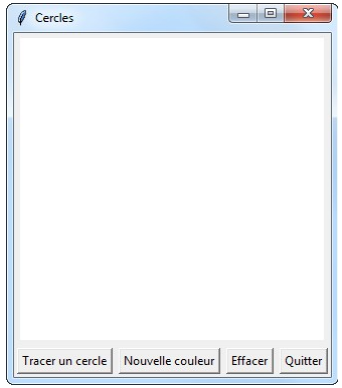
- la 1ère boucle construit une liste de transition **`transit`** qui contient toutes les cases d'une ligne ;
- la 2nde boucle ajoute chaque nouvelle ligne à la liste générale **`cases`**.

b) Ligne **`30`**, compléter les coordonnées permettant de tracer chaque carré en fonction de **`ligne`**, de **`colonne`** et de **`c`**. *Le décalage **`+2`** provient de la constatation effectuée à l'**Exemple n°12E**.*



c) La question précédente permet d'obtenir le quadrillage ci-contre. En rajoutant une instruction conditionnelle et en utilisant les paramètres **`fill`** et **`outline`**, afficher le damier désiré.

Exemple n°12G : *Ce sont les boutons qui commandent.*



La fenêtre ouverte par le fichier **`Exemple12G.py`** contient un canevas et quatre boutons :

- le bouton [*Tracer un cercle*] doit afficher, dans le canevas, un cercle dont les coordonnées du centre et le rayon sont aléatoires (*module **`random`***) ;
- le bouton [*Changer de couleur*] doit modifier aléatoirement la couleur des tracés suivants ;
- le bouton [*Effacer*] doit effacer tous les cercles déjà tracés.

a) Le bouton [*Changer de couleur*] appelle la fonction **`change_couleur()`**.
Cette fonction modifie la valeur de la variable globale **`couleur`**. Les différentes couleurs possibles sont définies dans une liste :
`['purple', 'cyan', 'green', 'red', 'blue', 'orange', 'black']`

b) Le bouton [*Tracer un cercle*] appelle la fonction **`tracer()`**.
Cette fonction dessine en **`couleur`** un cercle d'épaisseur **`2`** pixels.
Son centre est le point de coordonnées (**`x`** ; **`y`**) où **`x`** et **`y`** sont deux entiers aléatoires compris entre **`50`** et **`250`**. Son rayon est un entier aléatoire **`r`** compris entre **`10`** et **`40`**.

c) Le bouton [*Effacer*] appelle la fonction **`effacer()`**.
Cette fonction efface toutes les figures déjà tracées dans le canevas et ré-initialise **`couleur`** à **`'blue'`**.