Programmation avec Python
Interfaces graphiques (GUI)

Nom :

Prénom :

Cette partie est un résumé du chapitre 8 du livre « Apprendre à programmer avec Python » de *Gérard Swinnen*. Pour plus de détails et d'explications, vous pouvez vous y référer.

Nous utiliserons la bibliothèque graphique **Tkinter**, incluse dans Python.

I-Premiers pas

Création d'une fenêtre avec deux widgets (contraction de window gadget) : un bout de texte (**label**) et un bouton (**button**).

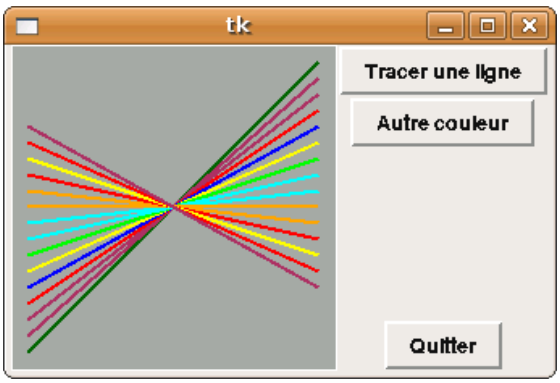
En mode interactif, taper les lignes suivantes :

```
from Tkinter import*
fen1=Tk()
tex1=Label(fen1, text='Bonjour tout le monde !', fg='red')
tex1.pack()
bou1=Button(fen1, text='Quitter', command=fen1.destroy)
bou1.pack()
fen1.mainloop()
```

fen1=**Tk()** crée la fenêtre qui s'appellera fen1.
tex1=**Label**(fen1, **text**='Bonjour tout le monde !', **fg**='red') : le premier argument est le nom de la fenêtre dans lequel sera le bouton (fen1 est le widget maître de l'objet tex1, ou l'objet tex1 est un widget esclave de l'objet fen1). Le deuxième est facile à comprendre et le troisième est la couleur d'avant plan (foreground, en anglais).
tex1.**pack()** : nous activons ici la méthode pack() à l'objet tex1. Cette méthode agit sur la disposition géométrique, la fenêtre maître est réduite automatiquement pour qu'elle soit juste assez grande pour contenir les widgets esclaves.
fen1.**mainloop()** : c'est cette ligne qui provoque le démarrage du réceptionnaire d'événements associé à la fenêtre. Cette instruction est nécessaire pour que l'application soit « à l'affût » des clics de souris, des pressions exercées sur les touches du clavier, etc. C'est donc cette instruction qui la met en marche.

II-Tracé de lignes dans un canevas

Taper le programme suivant, l'enregistrer et l'exécuter.




```
#-*- coding:Utf-8 -*-
# Petit exercice utilisant la bibliothèque graphique Tkinter

from Tkinter import*
from random import randrange

# --- définition des fonctions gestionnaires d'événements : ---
def drawline():
    "Tracé d'une ligne dans le canevas can1"
    global x1, y1, x2, y2, coul
    can1.create_line(x1,y1,x2,y2,width=2,fill=coul)
    # modification des coordonnées pour la ligne suivante :
    y2, y1 = y2+10, y1-10
```

```
def changecolor():
    "Changement aléatoire de la couleur du tracé"
    global coul
    pal=['purple','cyan','maroon','green','red','blue','orange','yellow']
    c = randrange(8)          # => génère un nombre aléatoire de 0 à 7
    coul = pal[c]


#------ Programme principal ------

# Les variables suivantes seront utilisées de manière globale :
x1, y1, x2, y2 = 10, 190, 190, 10       # coordonnées de la ligne
coul = 'dark green'                     # couleur de la ligne

# Création du widget principal ("maître") :
fen1 = Tk()

# Création des widgets "esclaves" :
can1 = Canvas(fen1,bg='dark grey',height=200,width=200)
can1.pack(side=LEFT)
bou1 = Button(fen1,text='Quitter',command=fen1.quit)
bou1.pack(side=BOTTOM)
bou2 = Button(fen1,text='Tracer une ligne',command=drawline)
bou2.pack()
bou3 = Button(fen1,text='Autre couleur',command=changecolor)
bou3.pack()

fen1.mainloop()              # démarrage du réceptionnaire d'événement

fen1.destroy()               # destruction (fermeture) de la fenêtre
```

Un canevas dans Tkinter est une surface rectangulaire délimitée, dans laquelle on peut installer ensuite divers dessins et images à l'aide de méthodes spécifiques (canevas s'écrit canvas en anglais !).

La fonctionnalité de ce programme est assurée essentiellement par les deux fonctions *drawline( )* et *changecolor( )*.

Dans la fonction *changecolor( )*, une couleur est choisie au hasard dans une liste, à l'aide de la fonction *randrange( )* importée du module *random*. Appelée avec un argument N, cette fonction renvoie un nombre entier, tiré au hasard entre zéro et N-1.

La commande liée au bouton « Quitter » appelle la méthode *quit( )* de la fenêtre fen1.
Cette méthode sert à fermer le réceptionnaire d'événements (*mainloop*) associé à cette fenêtre. Lorsque cette méthode est activée, l'exécution du programme se poursuit avec les instructions qui suivent l'appel de *mainloop*. Dans l'exemple, cela consiste donc à effacer (*destroy*) la fenêtre.

Exercices : Vous allez avoir à modifier le programme précédent. Après chaque modification, enregistrez le avec un nom différent pour ne pas l'écraser (*save as*).

II-1. Modifier le programme pour ne plus avoir que des lignes de couleur cyan, maroon et green.

II-2. Modifier le programme pour que toutes les lignes tracées soient horizontales et parallèles.

II-3. Agrandir le canevas de manière à lui donner une largeur de 500 unités et une hauteur de 650. Modifier également la taille des lignes, afin que leurs extrémités se confondent avec les bords du canevas.

II-4. Ajouter une fonction drawline2( ) qui tracera deux ligne rouges en croix au centre du canevas : l'une horizontale et l'autre verticale. Ajouter également un bouton portant l'indication « viseur ». Un clic sur ce bouton devra provoquer l'affichage de la croix.

II-5. Reprendre le programme initial. Remplacer la méthode create_line par la méthode create_rectangle. Que se passe-t'il ? Qu'indique les coordonnées fournies en paramètres ?

.....

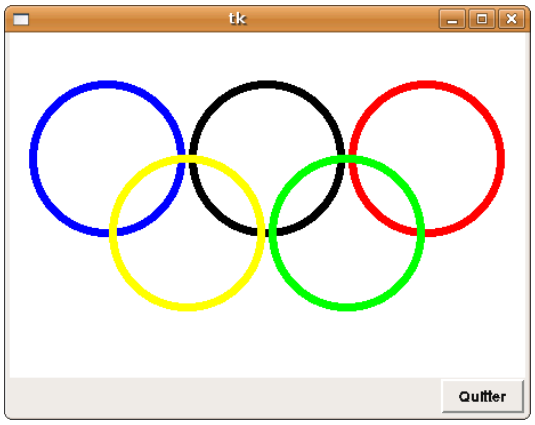
.....

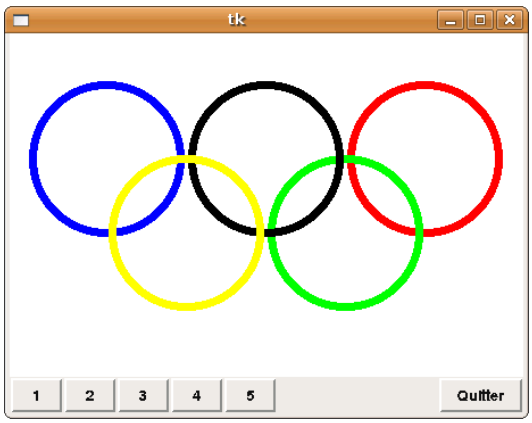
Recommencer en remplaçant cette fois par create_oval. Que se passe-t'il et qu'indique les coordonnées fournies en paramètres ?

.....

.....

II-6. Créer un programme qui dessinera les cinq anneaux olympiques dans un rectangle de fond blanc (white).Utiliser l'argument *outline* à la place de *fill* pour la couleur des anneaux. Un bouton « quitter » doit permettre de fermer la fenêtre.





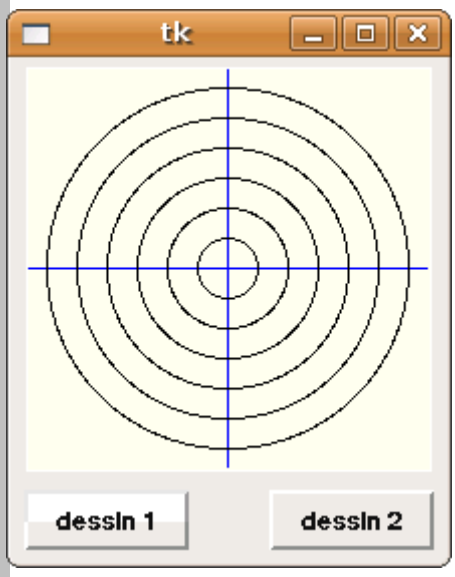
II-7. Modifier le programme précédent en y ajoutant cinq boutons. Chacun de ces boutons provoquera le tracé d'un anneau.

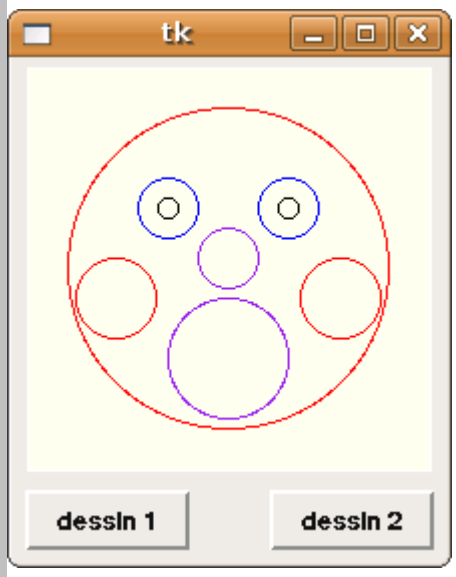
III-Deux dessins alternés

Taper le programme suivant, l'enregistrer et l'exécuter.

```
#-*- coding:Utf-8 -*-

from Tkinter import*

def cercle(x, y, r, coul='black'):
    "tracé d'un cercle de centre (x,y) et de rayon r"
    can.create_oval(x-r, y-r, x+r, y+r, outline=coul)

def figure_1():
    "dessiner une cible"
    # Effacer d'abord tout dessin préexistant :
    can.delete(ALL)
    # Tracer les deux lignes (vert. et horiz.) :
```

```
    can.create_line(100, 0, 100, 200, fill='blue')
    can.create_line(0, 100, 200, 100, fill='blue')
    # Tracer plusieurs cercles concentriques :
    rayon = 15
    while rayon < 100:
        cercle(100, 100, rayon)
        rayon += 15

def figure_2():
    "dessiner un visage simplifié"
    # Effacer d'abord tout dessin préexistant :
    can.delete(ALL)
    # Les caractéristiques de chaque cercle sont
    # placées dans une liste de liste :
    cc=[[100, 100, 80, 'red'],          # visage
        [70, 70, 15, 'blue'],           # yeux
        [130, 70, 15, 'blue'],
        [70, 70, 5, 'black'],
        [130, 70, 5, 'black'],
        [44, 115, 20, 'red'],           # joues
        [156, 115, 20, 'red'],
        [100, 95, 15, 'purple'],        # nez
        [100, 145, 30, 'purple']]       # bouche
    # on trace tous les cercles à l'aide d'une boucle :
    i=0
    while i < len(cc):          # parcourt de la liste
        el = cc[i]              # chaque élément est lui-même une liste
        cercle(el[0], el[1], el[2], el[3])
        i += 1

##### Programme principal #####

fen = Tk()
can = Canvas(fen, width=200, height=200, bg='ivory')
can.pack(side=TOP, padx=5, pady=5)
b1 = Button(fen, text='dessin 1', command=figure_1)
b1.pack(side=LEFT, padx=3, pady=3)
b2 = Button(fen, text='dessin 2', command=figure_2)
b2.pack(side=RIGHT, padx=3, pady=3)
fen.mainloop()
```





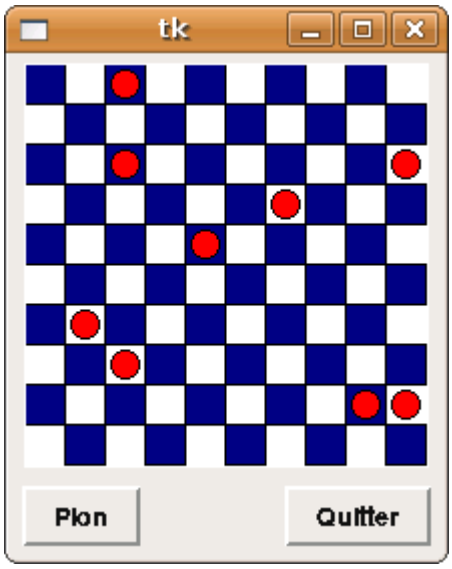
L'option *side* peut accepter les valeurs TOP, BOTTOM, LEFT ou RIGHT, pour « pousser » le widget du côté correspondant de la fenêtre.

Les options *padx* et *pady* permettent de réserver un petit espace autour du widget. Cet espace est exprimé en nombre de pixels : *padx* réserve un espace à gauche et à droite du widget, *pady* réserve un espace au-dessus et au-dessous du widget.

Prenez le temps d'essayer de bien comprendre ce programme.

Exercice : Écrire une application qui dessine un damier (carrés 'navy' sur fond blanc), ainsi que des pions rouges qui apparaissent au hasard lorsque que l'on clique sur un bouton dans un premier temps, on se contentera du damier).
Le canevas devra avoir une dimension de 300*300.
Les carrés devront avoir un côté égal à 30.
Les pions devront avoir un rayon égal à 10.

Le programme comportera les fonctions suivantes :




```
def damier():
    "dessiner dix lignes de carrés avec décalage alterné"
    ligne=0
    y=0
    while ligne<10:
        if ligne%2==0:          # une fois sur deux, on
            x=0                 # commencera la ligne de
        else :                  # carrés avec un décalage
            x=30                # de la taille d'un carré
        ligne_de_carres(x,y)
        y=y+30
        ligne=ligne+1
```

ligne_de_carres(x, y) : dessine une ligne de 5 carrés de couleur 'navy' espacés, en partant de (x,y).

disque(x, y, r, coul) : dessine un disque de centre (x,y), de rayon r et de couleur coul.

pion( ) : dessine un pion au hasard sur le damier (utilisera la fonction disque).

IV-Calculatrice

Taper le programme suivant, l'enregistrer et l'exécuter.


```
# -*- coding:Utf-8 -*-

from Tkinter import*
from math import*

# définition de l'action à effectuer si l'utilisateur actionne
# la touche "enter" alors qu'il édite le champ d'entrée :

def evaluer(event):
    chaine.configure(text='Résultat = '+str(eval(entree.get())))

##### Programme principal #####

fenetre = Tk()
entree = Entry(fenetre, background='white')
entree.bind("<Return>", evaluer)
chaine = Label(fenetre)
entree.pack()
chaine.pack()

fenetre.mainloop()
```

Au début du script, on importe le module math pour que la calculatrice puisse disposer de toutes les fonctions mathématiques et scientifiques usuelles : sinus, cosinus, racine carrée, etc.

La fonction *evaluer( )* sera la commande exécutée par le programme lorsque l'utilisateur actionnera la touche *Return*.
Cette fonction utilise la méthode *configure( )* du widget *chaine*, pour modifier son attribut *text*.
eval( ) évalue (fait le calcul) la chaîne de caractères.
str( ) transforme une expression numérique en chaîne de caractères.

get( ) est une méthode qui permet d'extraire du widget *entree* la chaîne de caractères qui lui a été fournie par l'utilisateur.

entree est un widget de la « classe » *Entry*. Afin que ce widget puisse transmettre au programme l'expression que l'utilisateur y aura encodée, il faut lui associer un événement à l'aide de la méthode *bind( )* (bind signifie « lier » en anglais).
entree.**bind**("<Return>", evaluer) signifie : « lier l'événement 'pression sur la touche <Return>' à l'objet *entree*, le gestionnaire de cet événement étant la fonction *evaluer* ».
L'argument *event* fourni à la fonction *evaluer* est obligatoire dès que l'on utilise la méthode *bind( )*.

V-Détection et positionnement d'un clic de souris

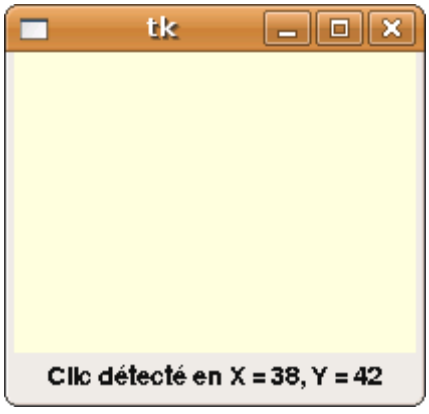
Taper le programme suivant, l'enregistrer et l'exécuter.

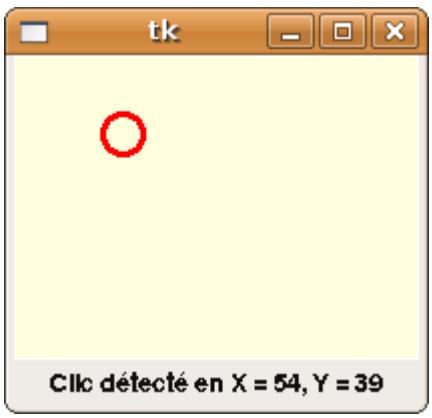
```
# -*- coding:Utf-8 -*-

# Détection et positionnement d'un clic de souris dans une fenêtre :

from Tkinter import*

def pointeur(event):
    chaine.configure(text="Clic détecté en X = "+str(event.x)+", Y = "+str(event.y))

fen=Tk()
cadre=Frame(fen, width=200, height=150, bg="light yellow")
cadre.bind("<Button-1>", pointeur)
cadre.pack()
chaine=Label(fen)
chaine.pack()

fen.mainloop()
```

Le script fait apparaître une fenêtre contenant un cadre (frame) rectangulaire de couleur jaune pâle.

La méthode *bind( )* du widget *cadre* associe l'événement *<clic à l'aide du premier bouton de la souris>* au gestionnaire d'événement « pointeur ».

Ce gestionnaire d'événement peut utiliser les attributs *x* et *y* de l'objet *event* généré automatiquement par Python, pour construire la chaîne de caractères qui affichera la position de la souris au moment du clic.





Exercice : Modifier le script ci-dessus de manière à faire apparaître un petit cercle rouge à l'endroit où l'utilisateur a effectué son clic (il faut d'abord remplacer le widget *Frame* par un widget *Canvas*).

VI-Les classes de widget Tkinter

Il existe 15 classes de base pour les widgets Tkinter :

Button : Un bouton classique, à utiliser pour provoquer l'exécution d'une commande quelconque.
Canvas: Un espace pour disposer divers éléments graphiques. Ce widget peut être utilisé pour dessiner, créer des éditeurs graphiques, et aussi pour implémenter des widgets personnalisés.
Checkbutton : Une « case à cocher » qui peut prendre deux états distincts (la case est cochée ou non). Un clic sur ce widget provoque le changement d'état.
Entry : Un champ d'entrée, dans lequel l'utilisateur du programme pourra insérer un texte quelconque à partir du clavier.
Frame : Une surface rectangulaire dans la fenêtre, où l'on peut disposer d'autres widgets. Cette surface peut être colorée. Elle peut aussi être décorée d'une bordure.
Label : Un texte (ou libellé) quelconque (éventuellement une image).
Listbox : Une liste de choix proposés à l'utilisateur, généralement présentés dans une sorte de boîte. On peut également configurer la Listbox de telle manière qu'elle se comporte comme une série de « boutons radio » ou de cases à cocher.
Menu : Un menu. Ce peut être un menu déroulant attaché à la barre de titre, ou bien un menu « pop up » apparaissant n'importe où à la suite d'un clic.
Menubutton : Un bouton-menu, à utiliser pour implémenter des menus déroulants.
Message : Permet d'afficher un texte. Ce widget est une variante du widget Label, qui permet d'adapter automatiquement le texte affiché à une certaine taille ou à un certain rapport largeur/hauteur.
Radiobutton : Représente (par un point noir dans un petit cercle) une des valeurs d'une variable qui peut en posséder plusieurs. Cliquer sur un « bouton radio » donne la valeur correspondante à la variable, et "vide" tous les autres boutons radio associés à la même variable.
Scale : Vous permet de faire varier de manière très visuelle la valeur d'une variable, en déplaçant un curseur le long d'une règle.
Scrollbar : « ascenseur » ou « barre de défilement » que vous pouvez utiliser en association avec les autres widgets : Canvas, Entry, Listbox, Text.
Text : Affichage de texte formatté. Permet aussi à l'utilisateur d'éditer le texte affiché. Des images peuvent également être insérées.
Toplevel : Une fenêtre affichée séparément, « par-dessus ».

Ces classes de widgets intègrent chacune un grand nombre de méthodes. On peut aussi leur associer (lier) des événements, comme déjà vu dans les pages précédentes. Tous ces widgets peuvent être positionnés dans les fenêtres à l'aide de trois méthodes différentes : la méthode *grid( )*, la méthode *pack( )* et la méthode *place( )*.

VII-Méthode *grid( )* pour la disposition des widgets

Taper le programme suivant, l'enregistrer et l'exécuter.

```
# -*- coding:Utf-8 -*-

from Tkinter import*

fen1=Tk()
txt1=Label(fen1, text='Premier champ : ')
txt2=Label(fen1, text='Second : ')
entr1=Entry(fen1, bg='white')
entr2=Entry(fen1, bg='white')
```

```
txt1.grid(row=0)
txt2.grid(row=1)
entr1.grid(row=0, column=1)
entr2.grid(row=1, column=1)
fen1.mainloop()
```

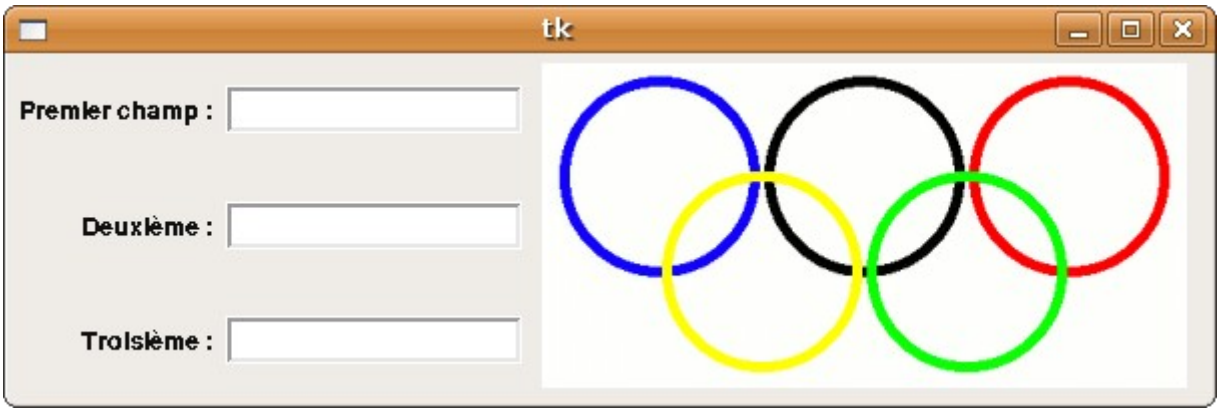
La méthode *grid( )* considère la fenêtre comme un tableau, avec des lignes (*row*) et des colonnes (*column*).

Il est possible d'aligner les widgets avec l'option *sticky* qui peut prendre l'une des quatre valeurs N, S, E, W (les quatre points cardinaux en anglais).

Exercices :

VII-1. Remplacer les deux premières instructions *grid( )* du script par :
txt1.grid(row=0, sticky=E)
txt2.grid(row=1, sticky=E)

VII-2. Le but de cet exercice est d'obtenir la fenêtre ci-dessous :



Le programme comportera entre autres, les parties suivantes :

```
# Création des widgets Label et Entry : À vous de compléter.
```

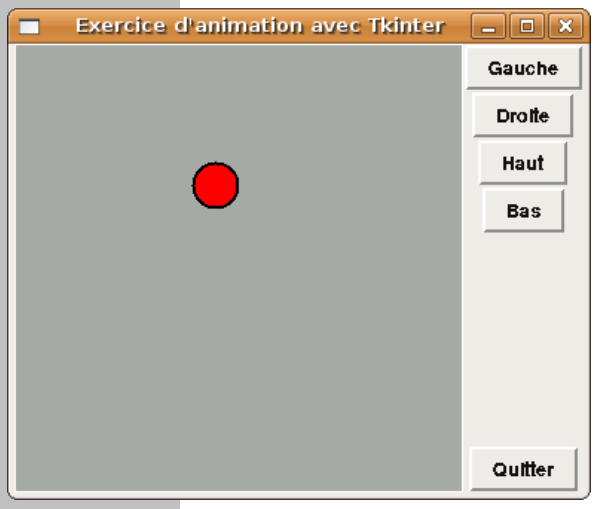
```
# Création d'un widget Canvas contenant une image :
can1=Canvas(fen1, width=320, height=160, bg='white')
anneaux=PhotoImage(file='AnOlympiques.gif')
item=can1.create_image(160, 80, image=anneaux)
```

```
# Mise en place des widgets :
                    à vous de placer les widgets Label et Entry

can1.grid(row=0, column=2, rowspan=3, padx=10, pady=5)
```

Explications :
Tkinter ne permet pas d'insérer directement une image dans une fenêtre. Il faut d'abord installer un canevas, et ensuite positionner l'image dans celui-ci, grâce à l'instruction item=can1.create_image(160, 80, image=anneaux).
Les deux premiers arguments transmis (160,80) indiquent les coordonnées *x* et *y* du canevas où il faut placer le centre de l'image (ici, l'image sera donc centrée dans le canevas).
L'instruction rowspan=3 indique que le canevas pourra « s'étaler » sur trois lignes.
padx et pady indiquent la dimension de l'espace à réserver autour du canevas.

VIII-Modification des propriétés d'un objet – Animation

Taper le programme suivant, l'enregistrer et l'exécuter.




```
# -*- coding:Utf-8 -*-

from Tkinter import*

# Procédure générale de déplacement :
def avance(gd, hb):
    global x1, y1
    x1, y1 = x1+gd, y1+hb
    can1.coords(oval1, x1, y1, x1+30, y1+30)

# Gestionnaire d'événements :
def depl_gauche():
    avance(-10, 0)

def depl_droite():
    avance(10, 0)

def depl_haut():
    avance(0, -10)

def depl_bas():
    avance(0, 10)

##### Programme principal #####

# Les variables suivantes seront utilisées de manière globale :
x1, y1 = 10, 10     # coordonnées initiales

# Création du widget "maître" :
fen1 = Tk()
fen1.title("Exercice d'animation avec Tkinter")

# Création des widgets "esclaves"
can1 = Canvas(fen1, bg='dark gray', height=300, width=300)
oval1=can1.create_oval(x1,y1,x1+30,y1+30,width=2,fill='red')
can1.pack(side=LEFT)
Button(fen1,text='Quitter',command=fen1.quit).pack(side=BOTTOM)
Button(fen1,text='Gauche',command=depl_gauche).pack()
Button(fen1,text='Droite',command=depl_droite).pack()
Button(fen1,text='Haut',command=depl_haut).pack()
Button(fen1,text='Bas',command=depl_bas).pack()

# Démarrage du réceptionnaire d'événement :
fen1.mainloop()

fen1.destroy()
```

La fonction *avance( )* redéfinit les coordonnées de l'objet « cercle coloré » (oval1) à chaque fois que l'on clique sur un des boutons. Ce qui provoque son animation.

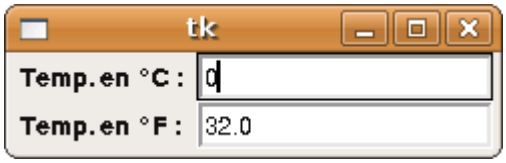
Les boutons ont été définis de manière plus compact (pas d'utilisation de variables).

Exercices :

VIII-1. Modifier le programme précédent de manière à ce que le cercle oval1 se place à l'endroit où l'on clique avec la souris.

VIII-2. Écrire un programme qui fasse apparaître une fenêtre avec un canevas (100*150). Dans ce canevas, placer un petit cercle (de rayon 15) censé représenter une balle. Sous le canevas, placer un bouton. Chaque fois que l'on clique sur le bouton, la balle doit avancer d'une petite distance (10) vers la droite, jusqu'à ce qu'elle atteigne l'extrémité du canevas. Si l'on continue à cliquer, la balle doit alors revenir en arrière jusqu'à l'autre extrémité, et ainsi de suite.

VIII-3. Écrire un programme qui fasse la conversion des degrés Celsius vers les degrés Fahrenheit en tapant la touche *Return*, et vice-versa.

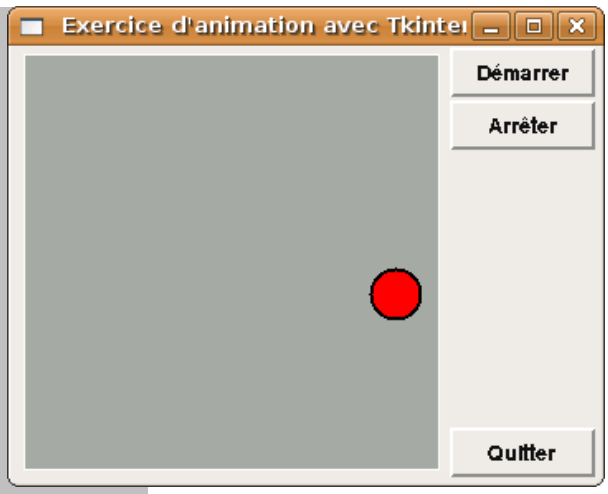


On utilisera la formule : $T_F = T_C \times 1{,}8 + 32$.

Vous aurez besoin de la méthode *get( )* du widget *Entry* (voir la calculatrice), ainsi que des méthodes *delete(0,END)* pour effacer un champ du début à la fin, et *insert(0,text)* pour insérer le texte *text* à partir du début du champ.

IX-Animation automatique – Récursivité

Taper le programme suivant, l'enregistrer et l'exécuter.




```
# -*- coding:Utf-8 -*-

from Tkinter import*

# définition des gestionnaires d'événements

def move():
    "déplacement de la balle"
    global x1, y1, dx, dy, flag
    x1, y1 = x1+dx, y1+dy
    if x1>210 :
        x1, dx, dy = 210, 0, 15
    if y1>210:
        y1, dx, dy = 210, -15, 0
    if x1<10:
        x1, dx, dy = 10, 0, -15
    if y1<10:
        y1, dx, dy = 10, 15, 0
    can1.coords(oval1,x1,y1,x1+30,y1+30)
    if flag>0:
        fen1.after(50,move)     # boucler après 50 millisecondes

def stop_it():
    "arrêt de l'animation"
    global flag
    flag=0

def start_it():
    "démarrage de l'application"
    global flag
    if flag==0:         # pour ne lancer qu'une seule boucle
        flag=1
```

```
    move()

##### Programme principal #####
# Les variables suivantes sont utilisées de manière globale :
x1, y1 = 10, 10     # coordonnées initiales
dx, dy = 15, 0      # 'pas' du déplacement
flag=0              # commutateur

# Création du widget "parent" :
fen1=Tk()
fen1.title("Exercice d'animation avec Tkinter")

# Création des widgets "enfants" :
can1=Canvas(fen1,bg='dark gray',height=250, width=250)
can1.pack(side=LEFT,padx=5,pady=5)
oval1=can1.create_oval(x1,y1,x1+30,y1+30,width=2,fill='red')
bou1=Button(fen1,text='Quitter',width=8,command=fen1.quit)
bou1.pack(side=BOTTOM)
bou2=Button(fen1,text='Démarrer',width=8,command=start_it)
bou2.pack()
bou3=Button(fen1,text='Arrêter',width=8,command=stop_it)
bou3.pack()

# Démarrage du réceptionnaire d'événements :
fen1.mainloop()

fen1.destroy()
```

La méthode *after( )* déclenche l'appel d'une fonction après qu'un certain laps de temps se soit écoulé (temps en millisecondes).
Ici la méthode *after( )* se trouve dans la fonction *move( )*. Elle appelle la fonction *move( )* elle-même. Cette technique de programmation très puissante est appelée « récursivité » : la fonction s'appelle elle-même. Attention, pour que le programme ne boucle pas indéfiniment, il faut mettre en place un moyen pour l'interrompre.

À chaque itération de la boucle, le contenu de la variable *flag* est testé (instruction *if*). Si le contenu de la variable *flag* est à 0, alors le bouclage ne s'effectue plus et l'animation s'arrête.
Un premier clic sur « Démarrer » assigne une valeur non nulle à la variable *flag*, puis provoque immédiatement un appel de la fonction *move( )*. Celle-ci s'exécute et continue à s'appeler elle-même toutes les 50 millisecondes, tant que *flag* ne revient pas à 0. Si l'on continue à cliquer sur le bouton « Démarrer », la fonction *move( )* ne peut plus être appelée tant que *flag* vaut 1. On évite ainsi le démarrage de plusieurs boucles concurrentes.
Le bouton « Arrêter » remet *flag* à 0 et la boucle s'interrompt.

Exercices :
IX-1. Dans la fonction *start_it( )*, supprimer l'instruction **if** flag==0: (et l'indentation des deux lignes suivantes). Cliquer plusieurs fois sur le bouton « Démarrer ». Expliquer ce qui se passe :

.....

.....

IX-2. Modifier le programme de telle façon que la balle change de couleur à chaque « virage ».
Vous aurez besoin de l'instruction : can1.**itemconfigure**(oval1,**fill**='green')

IX-3. Programme de jeu :
Une balle se déplace au hasard sur un canevas, à vitesse faible. Le joueur doit essayer de cliquer sur cette balle à l'aide de la souris. S'il y arrive, il gagne un point, la balle change de couleur et se déplace désormais un peu plus vite, et ainsi de suite. Arrêter le jeu au bout de 10 clics et afficher le score atteint.

Variantes : La balle peut devenir plus petite à chaque fois qu'elle est « attrapée » et changer de direction.

IX-4. Programme de jeu :
Écrire un programme pour le jeu du serpent : un « serpent » (constitué d'une ligne de quatre carrés) se déplace sur le canevas dans l'une des quatre directions : gauche, droite, haut, bas. Le joueur peut à tout moment changer la direction suivie par le serpent à l'aide des touches fléchées du clavier. Sur le canevas se trouve également des « proies » (des petits cercles fixes disposés au hasard). Il faut diriger le serpent de manière à ce qu'il « mange » les proies sans arriver en contact avec les bords du canevas. À chaque fois qu'une proie est mangée, le serpent s'allonge d'un carré, le joueur gagne un point, et une nouvelle proie apparaît ailleurs. La partie s'arrête lorsque le serpent touche l'une des parois, ou lorsqu'il a atteint une certaine taille.

Perfectionnement : La partie s'arrête également si le serpent se « recoupe ».