Tkinter 8.4 reférence: Une Interface Utilisateur Graphique (GUI) pour Python

par Michel AUBRY (Traducteur) (site perso de Michel AUBRY) John W. Shipman (Auteur)

Date de publication : 15 janvier 2010

Dernière mise à jour :

Description de l'ensemble des composants graphiques (widgets) de Tkinter pour construire des interfaces utilisateur graphiques (GUIs) dans le langage de programmation Python.

La publication originale (en anglais) est disponible sur le Web et aussi en format PDF . Veuillez transmettre tous commentaires à tcc-doc@nmt.edu.

1 - Qu'est-ce que Tkinter?

Tkinter est un ensemble de composants graphiques (widget) permettant de construire des interfaces utilisateur graphiques (GUIs) Python. Ce document contient seulement les caractéristiques (fonctions) de base.

Ce document s'applique au Python 2.5 et Tkinter 8.4 pour système X Window sous Linux. Votre version peut varier.

Références :

- *An Introduction to Tkinter* (1) par Fredrik Lundh.
- *Python and Tkinter Programming* par John Grayson (Manning, 2000, ISBN 1-884777-81-3).
- *Python 2.2 quick reference* (2) : Informations générales sur le langage Python.

Nous commencerons par regarder la partie visible de Tkinter : création des widgets et positionnement à l'écran. Plus tard nous parlerons de la façon de connecter le "front panel" de l'application à la logique.

2 - Une application minimale

Voici un petit programme Tkinter contenant seulement un bouton de Sortie :

```
<paragraph>#!/usr/local/bin/python ***1***

 from Tkinter import *   ***2***

class Application(Frame): ***3***

def __init__(self, master=None):</paragraph>
Frame.__init__(self, master) ***4***

self.grid() ***5***

self.createWidgets()</paragraph>
def createWidgets(self):</paragraph>
self.quitButton = Button ( self, text="Quit",</paragraph>
command=self.quit )  ***6***

self.quitButton.grid()  ***7***

app = Application() ***8***

app.master.title("Sample application")   ***9***

app.mainloop()  ***10***
```

1 Permet l'exécution du script, à supposer que sur votre système, le chemin de l'interpréteur Python soit /usr/local/bin/python.

2 Importe le paquet Tkinter entier dans votre programme.

3 Votre classe d'application hérite de la classe Maître de Tkinter.

4 Appelle le constructeur pour la classe parentale, Frame.

5 Nécessaire pour actualiser l'écran.

6 Créé un bouton étiqueté "Quit".

7 Place le bouton.

8 Le programme principal commence ici par l'initialisation de la classe Application.

9 Cet appel de méthode Indique le titre de la fenêtre Sample application.

10 Commence l'application principale, mets en position d'attente des événements de clavier et souris.

3 - Definitions

Avant de commencer, définissons certains des termes communs.

window

Ce terme a des significations différentes dans des contextes différents, mais en général il se réfère à un secteur rectangulaire quelque part sur votre écran.

top-level window

Une fenêtre existe indépendamment de votre écran. Elle aura le cadre standard et les commandes liés à votre système. Vous pouvez le déplacer sur votre bureau. Vous pouvez généralement la redimensionner, bien que votre application puisse l'empêcher.

widget

Terme générique pour n'importe laquelle des composantes d'une application dans une interface utilisateur graphique. Exemples de widget : boutons, radiobuttons, champs de texte (text fields), cadres (frames) et étiquettes de texte (text labels).

frame

Dans Tkinter, le widget Cadre est l'unité de base d'organisation pour des dispositions complexes. Un cadre est un secteur rectangulaire qui peut contenir d'autres widgets.

child, parent

Quand n'importe quel widget est créé, une relation parent-enfant est créée. Par exemple, si vous placez une étiquette de texte à l'intérieur d'un cadre, le cadre est le parent de l'étiquette.

4 - Gestion

Plus tard nous verrons les widgets, la construction de votre GUI. Comment positionner les widgets dans une fenêtre?

Bien qu'il y ait trois différents méthodes dans Tkinter, l'auteur préfère fortement la méthode .grid() pour à peu près tout. Cette méthode traite chaque fenêtre ou cadre comme une table - grille de rangées et de colonne.

- Une *cellule* est le secteur à l'intersection d'une rangée et d'une colonne.

- La largeur de chaque colonne est la largeur de la cellule la plus large dans cette colonne.

- La hauteur de chaque rangée est la hauteur de la plus grande cellule dans cette rangée.

 Pour les widgets qui ne remplissent pas la cellule entière, vous pouvez spécifier ce qui arrive à l'espace supplémentaire. Vous pouvez ou laisser l'espace supplémentaire à l'extérieur du widgets, ou étendre le widgets pour l'adapter, dans la dimension horizontale ou verticale.

 Vous pouvez combiner plusieurs cellules dans un secteur plus grand, un processus appelé enjambant (*spanning)*.

Quand vous créez un widget, il n'apparaît pas tant que vous ne l'enregistrez avec une méthode. Aussi, la construction et le placement d'un widget sont un processus à deux étapes qui donne quelque chose comme cela :

thing = Constructor(parent, ...)

thing.grid(...)

ou « Constructor » est une classe de widget comme Bouton, Cadre, etc et « parent » est le widget parent dans lequel ce widget enfant est construit. Tous les widgets ont une méthode .grid() que vous pouvez utiliser pour indiquer sa position.

4-1 - La méthode .grid()

Pour montrer un widget w sur votre écran :

w.grid(*option=value*, ...)

Cette méthode enregistre un widget w avec la méthode .grid() - si vous ne faites pas cela, le widget existera intérieurement, mais ne sera pas visible sur l'écran.

Le tableau qui suit montre les options de la méthode .grid ():

column	Le numéro de colonne où vous voulez accrocher le widget, en comptant à partir de zéro. La valeur par défaut est le zéro.
columnspan	Normalement un widget occupe seulement une cellule dans la table. Cependant, vous pouvez saisir plusieurs cellules d'une rangée et les fusionner dans une plus grande cellule avec l'option columnspan suivi du nombre de cellules. Par exemple *w*.grid(row=0, column=2, columnspan=3) positionne le widget w dans une cellule qui recouvre les colonnes 2, 3 et 4 de rangée 0
Ipadx	Ajout de x à la largeur intérieure du widget.
Ipady	Ajout de y à la hauteur intérieure du widget.
padx	Ajout de x à la largeur extérieure du widget.
pady	Ajout de y à la hauteur extérieure du widget.
row	Le numéro de rangée dans lequel vous voulez insérer le widget, en comptant à partir de zéro. La valeur par défaut est la 1ere rangée inoccupée dans le sens croissant.
rowspan	Normalement un widget occupe seulement une cellule dans la table. Cependant, vous pouvez fusionner plusieurs cellules adjacentes d'une même colonne avec l'option rowspan suivi du nombre de cellules. Cette option peut-être combinée avec

	l'option columnspan pour fusionner un bloc de cellules. Par exemple *w*.grid(row=3, column=2, rowspan=4, columnspan=5) placera le widget w dans un secteur formé en fusionnant 20 cellules, qui couvre les rangée 3 à 6 et les colonnes 2 à 6
sticky	Cette option détermine comment allouer l'espace supplémentaire de la cellule, qui n'est pas utilisé par le widget. Voir ci-dessous.

 Si vous préciser pas d'attribut sticky, par défaut le widget est centré dans la cellule.

 Vous pouvez placer le widget dans un coin de la cellule en utilisant sticky=NE (supérieur droit), SE (inférieur droit), SW (inférieur gauche), ou NW (supérieur gauche).

 Vous pouvez placer le widget centré le long d'un coté de la cellule en utilisant sticky=N (supérieur), E (droit), S (inférieur), ou W (gauche).

 Utilisez sticky=N+S pour tendre le widget verticalement, mais le laisser centré horizontalement.

 Utilisez sticky=E+W pour tendre le widget horizontalement, mais le laisser centré verticalement.

 Utilisez sticky=N+E+S+W pour tendre le widget tant horizontalement que verticalement et remplir la cellule.

 Les autres combinaisons fonctionnent également. Par exemple, Sticky=N+S+W tendra le widget verticalement et le placera contre le mur ouest (gauche).

4-2 - D'autres méthodes de la gestion .grid (grille)

Ces méthodes concernant la grille(grid) sont définies sur tous les widgets :

***w* .grid_bbox (column=None, row=None, col2=None, row2=None)**

Retourne une description 4-tuple de certaines ou toutes les options de grille du widget w. Les deux premiers nombres retournent les coordonnées x et y du coin gauche supérieur du secteur et les deux nombres suivant sont la largeur et la hauteur. Si vous n'indiquez pas les arguments de colonne et de rangée, les informations retournées seront la largeur et la hauteur. Si vous n'indiquez pas non plus les arguments col2 et row2, les informations retournées décrirons l'ensemble colonne, col2 et rangée, row2.

Par exemple, w.grid_bbox (0, 0, 1, 1) retourne l'information sur les quatre cellules, pas une.

***w* .grid_forget()**

Cette méthode fait disparaître de l'écran le widget w. Il existe toujours, mais n'est pas visible. Vous pouvez utiliser .grid () pour le faire apparaître de nouveau, mais il ne conserve pas ses options de grille.

***w* .grid_info()**

Renvoi un dictionnaire dont les clefs sont les noms d'option du widget w, avec les valeurs correspondantes de ces options.

***w* .grid_location(x,y)**

En fournissant les coordonnées (x, y) relatives au widget contenant, cette méthode retourne un tuple (col, row) de description de la cellule du widget qui contient cette coordonnée d'écran.

_w_ .grid_propagate()

Normalement, tous les widgets propagent leurs dimensions, c'est-à-dire qu'ils s'adaptent au contenu. Cependant, parfois vous voulez forcer un widget à une certaine taille, indépendamment de la taille de son contenu. Pour le faire, appelez w.grid_propagate (0) où w est le widget dont la taille vous veut forcer.

_w_ .grid_remove()

Cette méthode ressemble à .grid_forget (), mais ses options de grille sont rappelées, ainsi si vous utilisez .grid() pour le faire apparaître de nouveau, il utilisera les mêmes options de configuration grille.

_w_ .grid_size()

Retourne un 2-tuple contenant respectivement le nombre de colonnes et le nombre de rangées, du widget w.

_w_ .grid_slaves(row=None, column=None)

Retourne une liste des widgets gérés par le widget w. Si on ne fournit aucun argument, vous obtiendrez une liste de tous les widgets gérés. Renseigner l'argument « row » pour choisir seulement les widgets d'une rangée, ou l'argument « column » pour choisir seulement les widget d'une colonne.

4-3 - Configuration de la taille des colonnes et rangées

A moins que vous ne preniez certaines mesures, la largeur d'une colonne à l'intérieur d'un widget donné sera égale à la largeur de sa cellule la plus large et la hauteur d'une rangée sera la hauteur de sa cellule la plus grande. L'attribut sticky sur un widget contrôle seulement où il sera placé s'il ne remplit pas complètement la cellule.

Si vous voulez ignorer ce classement par taille automatique de colonnes et de rangée, utiliser ces méthodes sur le widget parent qui contient les options grille (grid) :

***W* .columnconfigure (*N* , *option* = *value* , ...)**

Dans le paramétrage des options de grille du widget W, configure la colonne N pour que l'option donnée ait la valeur donnée. Voir la table des options ci-dessous.

***W* .rowconfigure (*N* , *option* = *value* , ...)**

Dans le paramétrage des options de grille du widget W, configure la rangée N pour que l'option donnée ait la valeur donnée. Voir la table des options ci-dessous.

Le tableau suivant montre les options de configuration des colonnes et rangées.

minsize	La colonne ou la taille minimale de rangée en pixels. S'il n'y a rien dans la donnée colonne ou rangée, il n'apparaîtra pas, même si vous utilisez cette option.
pad	Un nombre de pixels qui seront ajoutés à la donnée colonne ou rangée, en plus de la

	colonne ou de la rangée de la plus grande cellule.
weight	Pour faire une colonne ou une rangée étirable, utilisez cette option et fournissez une valeur qui donne le poids relatif de cette colonne ou rangée en distribuant l'espace supplémentaire. Par exemple, si un widget w contient une configuration de grille, ces lignes distribueront 3/4 de l'espace supplémentaire à la première colonne et 1/4 à la deuxième colonne : w.columnconfigure(0, weight=3) w.columnconfigure(1, weight=1) Si cette option n'est pas utilisée, la colonne ou la rangée ne s'étirera pas.

4-4 - Faire une fenêtre maître redimensionnable

Voulez-vous laisser l'utilisateur redimensionner votre fenêtre entière d'application et distribuer l'espace supplémentaire parmi ses widgets internes ? Cela exige quelques opérations qui ne sont pas évidentes

Il est nécessaire d'utiliser les techniques de gestion de taille de rangée et de colonne, décrite dans la Section (p.), pour faire le grille de votre widget étirable. Cependant, ce seul n'est pas suffisant.

Considérez la petite application dans la Section (p.), qui contient seulement un bouton de Quit. Si vous exécutez cette application et redimensionnez la fenêtre, le bouton conservera la même taille, centrée dans la fenêtre.

Voici une version remplaçant la méthode . __ createWidgets() dans l'application minimale. Dans cette version, le bouton de Quit remplit toujours tout l'espace disponible.

```
def createWidgets(self):
top=self.winfo_toplevel()      ***1***
top.rowconfigure(0, weight=1)      ***2***
top.columnconfigure(0, weight=1)      ***3***
self.rowconfigure(0, weight=1)      ***4***
self.columnconfigure(0, weight=1)      ***5***
self.quit = Button ( self, text="Quit", command=self.quit )
self.quit.grid(row=0, column=0,      ***6***
sticky=N+S+E+W)
```

1 "La fenêtre de niveau supérieure" est la fenêtre la plus éloignée sur l'écran. Cependant, cette fenêtre n'est pas votre fenêtre D'application - c'est le parent de l'application. Pour obtenir la fenêtre au plus haut niveau, appelez la méthode .winfo_toplevel () sur n'importe quel widget dans votre application; voir section (p.).

2 Cette ligne fait la rangée 0 de la grille de la fenêtre de niveau supérieur étirable.

3 Cette ligne fait la colonne 0 de la grille de la fenêtre de niveau supérieur étirable.

4 Fait la rangée 0 de la grille du widget étirable.

5 Fait la colonne 0 de la grille du widget étirable.

6 L'argument sticky=N+S+E+W fait le bouton s'étendre pour remplir sa cellule.

Il y a encore un changement qui doit être fait. Dans le programme, changez la deuxième ligne comme indiquée :

```
            def __init__(self, master=None):
            Frame.__init__(self, master)
            self.grid(sticky=N+S+E+W)
            self.createWidgets()
            L'argument sticky=N+S+E+W du self.grid () est nécessaire pour que le widget s'étende pour
remplir sa cellule de la fenêtre au plus haut niveau
```

5 - Attributs standards

Avant de regarder les widgets, jetons un coup d'oeil à certains de leurs attributs communs - comme tailles, couleurs et polices de caractères - sont spécifiés.

 Chaque widget a un jeu d'options qui affectent ses apparence et comportement attributs tel que polices de caractères, couleurs, tailles, étiquettes et plus.

 Vous pouvez spécifier des options en envoyant au constructeur du widget des mot-clé comme le texte = " PANIC!" Ou height=20.

 Après avoir créé un widget, vous pouvez plus tard changer n'importe quelle option en utilisant la méthode .config() du widget et récupérer le cadre courant de n'importe quelle option en utilisant la méthode .cget() du widget. Voir section (p.) pour plus de renseignements sur ces méthodes.

5-1 - Dimensions

Diverses longueurs, largeurs et autres dimensions de widget peuvent être décrits dans plusieurs unités différentes.

 Si vous mettez une dimension à un entier, il est supposé être en pixels.

 Vous pouvez spécifier des unités en mettant une dimension à une chaîne contenant un nombre suivi par:

c	Centimètres
i	Pouces (inches)
m	Millimètres
p	Points d'impression (environ 1/72")

5-2 - Système de coordonnée

Comme dans la plupart des systèmes d'exploitation contemporains, l'origine de chaque système de coordonnée est à son coin gauche supérieur, avec la coordonnée x augmentant vers la droite et la coordonnée y augmentant vers le bas:

L'unité de base est le pixel, avec le pixel en haut à gauche ayant des coordonnées (0,0). Les coordonnées que vous spécifiez comme des entiers sont toujours exprimées dans des pixels, mais n'importe quelle coordonnée peut être spécifiée comme une quantité dimensionnée; voir section (p.).

5-3 - Couleurs

Il y a deux façons générales de spécifier des couleurs dans Tkinter.

 Vous pouvez utiliser une chaîne spécifiant la proportion de rouges, vert et bleu en chiffres hexadécimaux :

#rgb	Quatre bits par couleur
#rrggbb	Huit bits par couleur
#rrrgggbbb	Douze bits par couleur

Par exemple, "#fff" blanc, "#000000" noir, "#000fff000" vert pur, et "#00ffff" cyan pur (vert plus bleu).

 Vous pouvez aussi utiliser chacun le nom des couleurs standard localement défini. Les couleurs "white", "black", "red", "green", "blue", "cyan", "yellow", et "magenta" seront toujours disponibles. D'autres noms peuvent fonctionner, selon votre installation locale.

5-4 - Polices de caractères

Selon votre plate-forme, il peut y avoir jusqu'à trois façons de spécifier le style de type.

 Comme un tuple dont le premier élément est la famille de police de caractères, suivie par une taille en points, facultativement suivis par une chaîne contenant un ou plusieurs modificateurs de style. bold, italic, underline, et overstrike

Exemples : ("Helvetica", "16") pour un Helvetica à 16 points; ("Times", "24", "bold italic") pour Times bold italic à 24 points.

 Vous pouvez créer "un objet de police de caractères" en important le module tkFont et en utilisant son constructeur de classe de Police de caractères :

```
import tkFont
font = tkFont.Font ( option, ... )
```

Où les options incluent :

family	Le nom de la police de caractères (une chaîne de caractères string).
size	La taille de la police de caractères (un entier - integer). Pour obtenir une police de caractères n pixels de haut, utilisez -n.
weight	"bold" pour gras, "normal" pour normal.
slant	"italic" pour italic, "roman" pour normal.
underline	1 pour texte souligné, 0 pour normal.
overstrike	1 pour texte surfrappé, 0 pour normal.

Par exemple, pour obtenir un 36-point bold Helvetica italic:

```
helv36 = tkFont.Font ( family="Helvetica",
size=36, weight="bold" )
```

 si vous utilisez le Système X Window, vous pouvez utiliser n'importe lequel des noms de police de caractères X. Par exemple, la police de caractères nommée

"-*-lucidatypewriter-medium-r-*-*-*-140-*-*-*-*-*-*" est la police de caractères de largeur fixe préférée de l'auteur. Utilisez le programme xfontsel pour vous aider à choisir des polices de caractères qui vous plaises.

Pour obtenir une liste de toutes les familles de polices de caractères disponibles sur votre plate-forme, appelez cette fonction

```
tkFont.families()
```

La valeur de retour est une liste de chaîne. *Note*: Vous devez créer votre fenêtre racine avant l'appel de cette fonction.

Ces méthodes sont définies sur toutes les Polices de caractères :

.actual (*option* =None)

Si vous ne passez aucun argument, vous récupérez un dictionnaire des attributs réels de la police de caractères, qui peuvent différer de ceux vous avez demandé. Pour récupérer la valeur d'un attribut, passez son nom comme un argument.

.cget (*option*)

Rend la valeur de l'option donnée.

.configure (*option* , ...)

Utilisez cette méthode pour changer une ou plusieurs options sur une police de caractères. Par exemple, si vous avez un objet de Police de caractères appelé titleFont, si vous appelez titleFont.configure(family="times", size=18), cette police de caractères changera à 18pt Times et n'importe quels widget qui utilise cette police de caractères changera aussi.

.copy()

Rend une copie d'un objet de Police de caractères.

.measure (*text*)

Passez à cette méthode une chaîne et il rendra le nombre de pixels de largeur que la chaîne prendra dans la police de caractères. Attention : quelques caractères inclinés peuvent s'étendre à l'extérieur de ce secteur.

.metrics (*option*)

Si vous appelez cette méthode sans arguments, elle retourne un dictionnaire de toute la métrique de police de caractères. Vous pouvez récupérer la valeur de juste un métrique en passant son nom comme un argument. La métrique inclut :

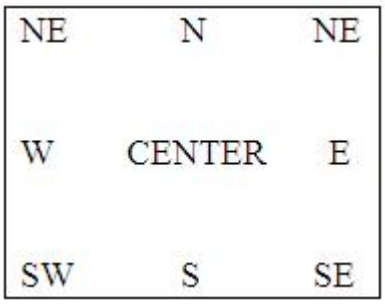
ascent	Nombre de pixels de hauteur entre la ligne des bases et le sommet du signe diacritique le plus haut.
descent	Nombre de pixels de hauteur entre la ligne des bases et le bas du signe diacritique le plus bas.
fixed	Cette valeur est 0 pour une police de caractères de largeur variable et 1 pour une police de caractères monoespacée.
linespace	Nombre de pixels de total de hauteur. C'est l'avancement de type couchent solide dans la police de caractères donnée.

5-5 - Anchors (ancres)

Les constantes sont définies dans le paquet Tkinter que vous pouvez utiliser pour contrôler où les articles sont placés quant à leur contexte. Par exemple, les anchors (ancres) peuvent spécifier où un widget est localisé à l'intérieur d'un cadre quand le cadre est plus grand que le widget.

On donne ces constants comme des points de boussole, où le nord est en haut et l'ouest est à gauche. Nous faisons des excuses aux lecteurs de notre Hémisphère sud pour ce chauvinisme de l'Hémisphère nord (3) .

On montre les constantes anchor dans ce diagramme :



Par exemple, si vous créez un petit widget à l'intérieur d'un grand cadre et utilisez l'option anchor=SE, le widget sera placé dans le coin en bas à droite du cadre. Si vous avez utilisé anchor=N au lieu de cela, on centrerait le widget le long du bord supérieur.

Les ancres sont aussi utilisées pour définir où le texte est placé par raport à un point de référence. Par exemple, si vous utilisez le CENTER comme une ancre de texte, on centrera le texte horizontalement et verticalement autour du point de référence. L'ancre NW placera le texte pour que le point de référence coïncide avec le coin en haut à gauche de la boîte contenant le texte. L'ancre W centrera le texte verticalement autour du point de référence, avec le bord gauche de la boîte de texte passant par ce point, et cetera

5-6 - Styles de Relief

Le *style relief* d'un widget se réfère aux certains effets 3-D simulés autour de l'extérieur du widget. Voici une copie d'écran d'une rangée de boutons exposant tous les styles de relief possibles :

La largeur de ces frontières dépend de l'attribut de borderwidth du widget. Les susdites expositions graphiques qu'ils ressemblent avec une frontière à 5 pixels; la largeur de frontière par défaut est 2.

5-7 - Bitmaps

Pour des options bitmap dans des widgets, on garantit ces bitmaps pour être disponible :

Le graphique ci-dessus montre des widgets de Bouton portant les bitmaps standard. De gauche à droite, il y a "error", "gray75", "gray50", "gray25", "gray12", "hourglass" (le sablier), "info", "questhead", "question", and "warning".

Vous pouvez utiliser vos propres bitmaps. N'importe quel fichier de format .xbm (X bit map) marchera. En place d'un nom bitmap standard, utilisez la chaine "@" suivi par le nom de chemin du fichier de .xbm

5-8 - Curseurs

Il y a ume multitude de curseurs de souris différents disponibles. Dans le tableau suivant, vous verrez leurs noms et graphisme. Le graphique peut varier selon votre système d'exploitation.

 arrow	 man
 based_arrow_down	 middlebutton
 based_arrow_up	 mouse
 boat	 pencil
 bogosity	 pirate
 bottom_left_corner	 plus
 bottom_right_corner	 question_arrow
 bottom_side	 right_ptr
 bottom_tee	 right_side
 box_spiral	 right_tee
 center_ptr	 rightbutton
 circle	 rtl_logo
 clock	 sailboat
 coffee_mug	 sb_down_arrow
 cross	 sb_h_double_arrow
 cross_reverse	 sb_left_arrow
 crosshair	 sb_right_arrow
 diamond_cross	 sb_up_arrow
 dot	 sb_v_double_arrow
 dotbox	 shuttle
double_arrow	sizing
draft_large	spider
draft_small	spraycan
draped_box	star
exchange	target
fleur	tcross

5-9 - Images

Il y a trois méthodes générales pour utiliser des images graphiques dans votre application Tkinter.

 Pour utiliser des images bitmap (à deux couleurs) dans un format de .xbm, voir chapitre 5.9.1, The BitmapImage class.

 Pour utiliser des images multicolores dans un format .gif, .pgm, ou .ppm, voir chapitre 5.9.2, The PhotoImage class .

 La bibliothèque Python (Python Imaging Library ou PIL) supporte des images de formats dans une variété beaucoup plus large. La classe ImageTk est spécifiquement conçue pour utiliser des images dans des applications Tkinter. Voir le document concernant la compilation PIL *Python Imaging Library (PIL) quick reference* (4)

5-9-1 - La classe BitmapImage

Pour montrer une image à deux couleurs dans le format de .xbm, vous aurez besoin de ce constructeur :

```
BitmapImage ( file=f[, background=b][, foreground=c] )
```

Avec *f* le nom du fichier image .xbm

Normalement, les bits de premier plan (foreground) dans l'image seront montrés comme des pixels noirs et les bits d'arrière plan (background) seront transparents. Pour changer ce comportement, utilisez l'option facultative background=b pour colorer les bits d'arrière plan en b et l'option facultative foreground=c pour colorer les bits de premier plan en c. Pour la spécification des couleurs, voir chapitre 5.3, "Couleurs"

Ce constructeur rend une valeur qui peut être utilisée n'importe où Tkinter a une image.

Par exemple, pour montrer une image comme une étiquette, utilisez un widget Etiquette et renseignez l'objet BitmapImage comme valeur d'option de l'image :

```
logo = BitmapImage("logo.xbm", foreground='red')
Label ( image=logo ).grid()
```

5-9-2 - La classe PhotoImage

Pour montrer une image colorée de format .gif, .pgm, ou .ppm, vous aurez besoin de ce constructeur :

```
PhotoImage ( file=f )
```

Avec *f* le nom du fichier image .xbm. Le constructeur retourne une valeur qui peut être utilisée n'importe où Tkinter a une image.

5-10 - Geometry strings

Une "geometry string" est une façon standard de décrire la taille et l'emplacement d'une fenêtre au plus haut niveau sur un bureau.

Une "geometry string" a cette forme générale :

```
"wxh±x±y"
```

Avec :

 w et h donnent la largeur et la hauteur de fenêtre en pixels. Ils sont séparés par le caractère "x".

 Si la partie suivante a la forme +x, il spécifie que le côté gauche de la fenêtre sera à x pixels du côté gauche du bureau. S'il a la forme-x, le côté droit de la fenêtre sera à x pixels du côté droit du bureau.

 Si la partie suivante a la forme +y, il spécifie que le sommet de la fenêtre sera à y pixels au-dessous du sommet du bureau. S'il a la forme -y, le bas de la fenêtre sera à y pixels au-dessus du bord bas du bureau.

Par exemple, une fenêtre créée avec la géométrie "120x50-0+20" aura 120 pixels larges par 50 pixels haut et son coin droit supérieur sera le long du bord droit du bureau et à 20 pixels au-dessous du bord supérieur.

5-11 - Noms de fenêtre

Le terme de fenêtre (window) décrit un secteur rectangulaire sur le bureau.

 La fenêtre au plus haut niveau ou la fenêtre racine (root) est une fenêtre qui a une existence indépendante sous la fenêtre du système. Elle pocède les attributs du système et peut être déplacé et redimensionné indépendamment. Votre application peut utiliser n'importe quel nombre de fenêtres racines.

 Le terme de fenêtre (window) s'applique aussi à n'importe quel widget qui fait partie d'une fenêtre racine.

Tkinter nomme toutes ces fenêtres utilisant un nom/chemin de fenêtre hiérarchique.

 La fenêtre racine se nomme « . ».

 Les fenêtres filles ont un nom de cette forme « .n » ou n est un entier sous forme de chaine de caractères. Par exemple, un fenêtre nommée « .135932060 » est fille de la fenêtre racine (« . »).

 Les fenêtres filles de fenêtres filles ont les noms de la forme "p.n" où p est le nom de la fenêtre parent et n est un certain entier. Par exemple, un fenêtre nommée « .135932060.137304468 » a pour parent « .13593206 », aussi est-elle petite-fille de la fenêtre racine.

 le nom relatif de la fenêtre est la partie droite qui suit le dernier « . » du nom complet (nom/chemin). Pour suivre l'exemple précédent, le nom relatif de la fenêtre petite-fille est « 137304468 ».

Le chemin de n'importe quel widget w peut être donné en appelant str(w).

Voir également Chapitre 25 Méthode Universel pour widget pour voir les commandes utilisées sur les nom de fenêtre, spécialement .winfo_name, .winfo_parent, et .winfo_pathname.

5-12 - Styles de fin et de jonction de ligne

Pour obtenir facilement et proprement des diagrammes, il est bon d'avoir recours au "cap style" (fin de ligne) et "join style" (jonction).

 Le "cap style" d'une ligne est la forme de la fin de la ligne. Les styles sont :

 BUTT: la fin de la ligne est coupée par une ligne droite perpendiculaire passant par le point de fin.

 PROJECTING: la fin de la ligne est coupée par une ligne droite perpendiculaire passant à une distance du point de fin égale à la moitier de l'épaisseur de la ligne.

 ROUND: la fin de la ligne décrit un demi-cercle centré sur le point de fin.

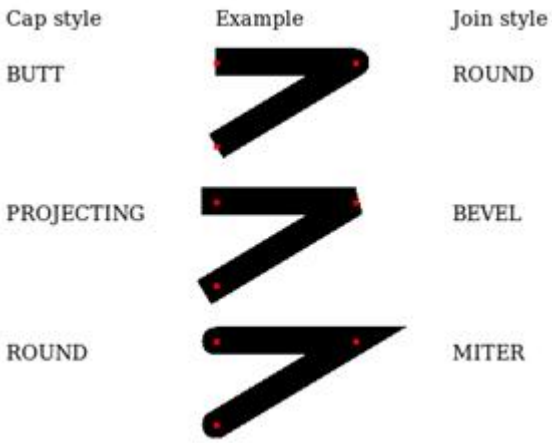
 Le "join style" décrit la forme de l'angle, jonction de deux segments de ligne.

 ROUND: la jonction décrit un demi-cercle centré sur le point d'union des deux segments.

 BEVEL: la jonction se termine par une ligne droite d'un angle intermédiaire à l'union des deux segments.

 MITER: les bords des deux segments se poursuivent pour se rencontrer en un point.

L'illustration montre les diverses options. Les point rouges symbolisent la fin nominale des segments.



5-13 - Modèles de Tirets

Un certain nombre de widgets vous permettent de spécifier un contour. Le tiret (Dash) et les options dashoffset vous donnent le contrôle du modèle des tirets.

dash

Cette option est spécifiée par un tuple d'entiers. Le premier entier spécifie combien de pixels doivent être dessinés. Le second spécifie combien de pixels doivent être sautés avant le prochain départ de dessin, etcetera. Quand tous les entiers du tuple sont épuisés, ils sont réutilisés dans le même ordre jusqu'à la fin de la ligne.

Par exemple, dash= (3,5) produits des tirets alternant 3 pixels dessinés séparés par des écarts de 5 pixels. Une valeur dash= (7,1,1,1) produits un modèle de tiret/point, avec un tiretde 7 pixels de long, 1pixel d'écart, 1 point de 1pixel puis un écart de 1 pixel. Une valeur dash= (5), produits des tirets alternant 5 pixels dessiné et 5 pixels d'écarts.

dashoff

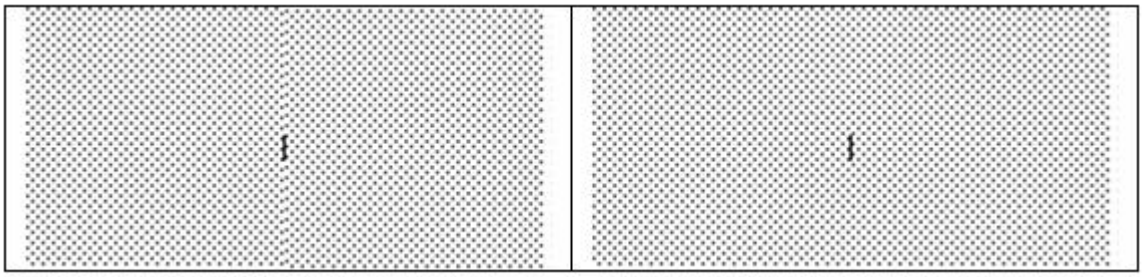
Pour commencer le modèle tiret à un point différent que le début, utilisez une option de dashoff=n, où n est le nombre de pixels pour sauter au début du modèle.

Par exemple, dash= (5, 1, 2, 1) et dashoff=3, le premier modèle produit sera : 2 on, 1 off, 2 on et 1 off. La suite du modèle sera 5 on, 1 off, 2 on et 1 off. Ci-joint la copie d'écran d'une ligne dessinée avec cette combinaison d'options :

5-14 - Correspondance des modèles d'objets graphiques

Cela peut sembler pointilleux, mais si vous dessinez un graphique qui a deux objets avec des modèles différents, un professionnel s'assurera que les modèles s'alignent le long de leur frontière

Voici un exemple. La copie d'écran de gauche montre deux objets adjacents 100 × 100 avec le modèle "gray12", mais l'objet de droite est compensé verticalement par un pixel. La petite ligne noire dans le centre de la figure est positionnée le long de la frontière des objets.



La deuxième copie d'écran est la même, sauf que les deux objets adjacents 100 × 100 ont leurs modèles de trait alignés.

En pratique, cela arrive dans deux situations. L'alignement de grands secteurs est contrôlé par une option nommée offset. Pour des figures avec de grandes surfaces , l'option outlineoffset contrôle leur alignement. Ces deux options ont les valeurs d'une de ces formes :

- "x, y" : Compense les modèles de trait par les valeurs x et y relativent au niveau haut de la fenêtre ou à l'origine du canvas.

- "#x, y" : Pour les objet d'un Canvas, compense les modèles de trait par les valeurs x et y relativent au niveau haut de la fenêtre.

- "ne", "se", "sw", "nw" : Aligne un coin du modèle de trait avec le coin correspondant de l'objet contenant. Par exemple, "ne" signifie que le coin en haut à gauche du modèle de trait coïncide avec le coin en haut à gauche l'objet contenant.

- "n", "e", "s", "w" : Aligne le modèle au centre d'un coté de l'objet contenant. Par exemple, "e" signifie que le centre du modèle de trait coïncide avec le centre du coté droit de l'objet contenant.

- "center" : Aligne le centre du modèle au centre de l'objet contenant.

6 - Le widget Button

Pour créer un bouton poussoir (pushbutton) dans une fenêtre de niveau haut ou un cadre nommé *parent*:

```
w = Button ( parent, option=value, ... )
```

Le constructeur retourne le nouveau widget de Bouton avec ses options :

activebackground	Couleur de fond quand le bouton est sous le curseur.
activeforeground	Couleur de premier plan quand le bouton est sous le curseur.
anchor	Où le texte est positionné sur le bouton. Voir section (p.). Par exemple, anchor=NE placerait le texte au coin supérieur droit du bouton.
bd ou borderwidth	Largeur de la frontière extérieure du bouton; voir section (p.). Par défaut la valeur est deux pixels.
bg ou background	Couleur normale de fond.
bitmap	Nom d'un des bitmaps standard pour indiquer le bouton (au lieu du texte).
command	Fonction ou méthode à être appelée quand le bouton est cliqué.
cursor	Curseur choisit pour indiquer lorsque la souris est sur le bouton.
default	NORMAL par défaut; indiqué DISABLED si le bouton doit être initialement mis hors service (insensible aux clics de souris).
disabledforeground	La couleur de premier plan utilisée quand le bouton est mis hors service.
fg ou foreground	Couleur normale du premier plan (texte).
font	Police de caractères à utilisée pour l'étiquette du bouton.
height	Hauteur du bouton en lignes (pour boutons textuels) ou pixels (pour images).
highlightbackground	Couleur de focus quand le widget n'a pas de focus.
highlightcolor	Couleur de focus quand le widget a le focus.
highlightthickness	Intensité de la couleur de focus.
image	Image positionnée sur le bouton (au lieu du texte).
justify	Justification : LEFT pour justifier à gauche, CENTER pour centrer ou RIGHT pour justifier à droite
overrelief	Le style de relief utilisé tandis que la souris est sur le bouton; le relief par défaut est RAISED. Voir section (p.).
padx	Remplissage supplémentaire gauche et droite du texte. Voir section (p.) pour les valeurs possibles.
pady	Remplissage supplémentaire au dessus et au dessous du texte.
relief	Spécifie le type de relief du bouton (Voir section (p.)). Relief par défaut est RAISED.
repeatdelay	Voir ci-dessous repeatinterval
repeatinterval	Normalement, un bouton n'agit qu'une seule fois quand l'utilisateur actionne le bouton de la souris. Si vous voulez que le bouton agisse à intervalles réguliers tant que le bouton de la souris est maintenu, positionner cette option à un certain nombre de millisecondes d'utilisation entre les répétitions et mettre le repeatdelay

	au nombre de millisecondes d'attente avant le début de répétition. Par exemple, si vous spécifiez "repeatdelay=500, repeatinterval=100" le bouton s'actionnera après une demie seconde puis chaque dixième d'une seconde, jusqu'à ce que l'utilisateur n'appuie plus sur le bouton de la souris. Si l'utilisateur ne maintient pas le bouton de souris au moins le temps de repeatdelay, le bouton fonctionnera normalement.
state	Positionner cette option a DISABLED pour rendre le bouton insensible . A la valeur ACTIVE lorsque la sousis est sur le bouton. Par défaut la valeur est NORMAL.
takefocus	Normalement, le contrôle de clavier agit sur les boutons (voir section (p.) et un caractère espace agit de la même façon qu'un clic de souris, "poussant" le bouton. Vous pouvez paramétrer l'option takefocus a zéro pour empêcher le contrôle de clavier pour le bouton.
text	Texte affiché sur le bouton. Utilisez des retours à la ligne pour montrer des lignes de texte multiples.
textvariable	Un StringVar () qui est associé au texte sur ce bouton. Si la variable est modifiée, la nouvelle valeur est affichée sur le bouton. Voir section (p.).
underline	Par défaut la valeur est -1, signifiant qu'aucun caractère du texte sur le bouton ne sera souligné. Si le nombre est positif, le caractère de texte correspondant sera souligné. Par exemple, underline=1 soulignerait le deuxième caractère du texte du bouton.
width	Largeur du bouton en lettres (pour boutons textuels) ou pixels (pour images).
wraplength	Si cette valeur est positive, les lignes de texte seront enveloppées pour obtenir cette longueur. Ensemble des valeurs possibles, voir section (p.).

Méthodes sur les objets bouton :

.flash()

Fait flasher le bouton plusieurs fois entre des couleurs actives et normales. Laisse le bouton dans l'état d'origine. Ignoré si le bouton est mis hors service.

.invoke()

Appelle le rappel de service du bouton, et renvoi ce que cette fonction retourne. N'a aucun effet si le bouton est hors service ou n'a aucun rappel de service.

7 - Le widget Canvas

Un Canvas est un secteur rectangulaire (canevas) destiné à supporter des images ou d'autres dispositions complexes. Vous pouvez y placer le graphisme, le texte, des widgets, ou des cadres. Voir les sections suivantes pour les méthodes qui créent des objets sur des Canvas :

- .create_arc():Une tranche dellipse. Voir section (p.).
- .create_bitmap():Une image bitmap. Voir section (p.).
- .create_image():Une image graphique. Voir section (p.).
- .create_line():Un ou plusieurs segments de ligne. Voir section (p.).
- .create_oval():Une ellipse; utilisez-le aussi pour dessiner des cercles, qui sont un cas spécial d'ellipse. Voir section (p.).
- .create_polygon(): Un polygone. Voir section (p.).
- .create_rectangle(): Un rectangle. Voir section (p.).
- .create_text(): Texte et annotation. Voir section (p.).
- .create_window(): Une fenêtre rectangulaire. Voir section (p.).

Pour créer un Canvas :

```
w = Canvas ( parent, option=value, ... )
```

Le constructeur retourne le nouveau widget Canvas. Les options supportées :

bd ou borderwidth	Largeur de la frontière extérieure du canvas, voir section (p.).
bg ou background	Couleur normale de fond. Par défaut gris clair, soit "#E4E4E4"
closeenough	Un réel qui spécifie à quelle distance la souris doit être d'un article pour être considéré à l'intérieur. Par défaut la vaelur est 1.0.
confine	Valeur par défaut vrai (True).Si vrai, le Canvas ne peut pas être scrollé à l'extérieur de la scrollregion (voir ci-dessous).
cursor	Curseur choisit dans le Canvas, voir section (p.).
height	Hauteur du Canvas, voir section (p.).
highlightbackground	Couleur de focus quand le Canvas n'a pas de focus. Voir section (p.).
highlightcolor	Couleur de focus quand le widget a le focus.
highlightthickness	Intensité de la couleur de focus.
relief	Spécifie le type de relief du Canvas (voir section (p.)). Relief par défaut est FLAT
scrollregion	Un tuple (w, n, e, s) qui définit dans quel secteur le Canvas peut être scrolled, où w

	est le côté gauche, n le sommet, e le côté droit et s le bas.
selectbackground	La couleur de fond utilisée pour l'affichage des objets choisis.
selectborderwidth	La largeur de la bordure utilisée autour des objets choisis.
selectforeground	La couleur de premier plan utilisée pour l'affichage des objets choisis.
takefocus	Normalement, pour ce widget, le contrôle de clavier (voir section (p.))scroll uniquement la touche de tabulation si le contrôle est paramétré(voir section (p.) pour une vue d'ensemble des contrôles de clavier). Si vous mettez cette option à 1, le contrôle de clavier se fera.
width	Largeur du Canvas (voir section (p.)).
xscrollincrement	Normalement, les Canvas peuvent être scrollé horizontalement vers n'importe quelle position. Vous pouvez obtenir ce comportement en mettant xscrollincrement à zéro. Si vous mettez cette option xscrollincrement à une valeur positive, le canvas peut être placée seulement sur les multiples de valeur et la valeur sera utilisée comme unités de scrolling, comme par exemple, quand l'utilisateur clique sur les flèches en bout d'une scrollbar. Pour plus d'informations sur des unités de scrolling, voir section (p.).
xscrollcommand	Si le canvas peux être scrollé, cet attribut devrait être la méthode .set() de scrollbar horizontale.
yscrollincrement	Travail comme xscrollincrement mais dans le sens vertical.
yscrollcommand	Si le canvas peux être scrollé, cet attribut devrait être la méthode .set()de scrollbar verticale.

7-1 - Coordonnées Canvas

Parce que le canvas peut être plus grand que la fenêtre et équipé de scrollbars pour déplacer le canvas à l'intérieur de la fenêtre, il y a deux systèmes de coordonnée pour chaque canvas :

- Les coordonnées de fenêtre d'un point sont relatives au coin en haut à gauche du secteur où le canvas apparaît.

- Les coordonnées de canvas d'un point sont relatives au coin en haut à gauche du canvas.

7-2 - L'ordre d'exposition d'un Canvas

L'ordre d'exposition se réfère à l'ordre de tous les objets du canvas, depuis l'arrièr plan ("le bas" de l'ordre d'exposition) au premier plan ("le sommet").

S'il y a chevauchement de deux d'objets, celui qui est au-dessus de l'autre dans l'ordre d'exposition signifie qu'il est tout près du premier plan, apparaîtra dans le secteur de chevauchement et obscurcira celui du dessous. Par défaut,

les nouveaux objets sont toujours créés au sommet de l'ordre d'exposition (et donc devant tous les autres objets), mais vous pouvez re-ordonner l'ordre d'exposition.

7-3 - L'Id des objets dans un Canvas

L'ID d'un objet est la valeur rendue par le constructeur pour cet objet. Tout ID est un entier simple et l'ID d'un objet est unique dans ce canvas.

7-4 - Tag (étiquette) dans un Canvas

Une tag (étiquette) est une chaîne que vous pouvez associer a des objets sur le canvas.

- une tag peut être associé à n'importe quel nombre d'objets sur le canvas, y compris zéro.

- un objet peut avoir n'importe quel nombre de tags associées, y compris zéro.

Les tags sont très utilisées. Par exemple, si vous dessinez une carte sur un canvas et s'il y a des objets de texte sur des rivières, vous pourriez attacher la tag "riverLabel" à tous ces objets de texte. Cela vous permettrait d'exécuter des opérations simultanées sur tous les objets portant cette Tag, comme le changement de couleur ou la suppression.

7-5 - Les arguments TagOrId

Un argument tagOrId spécifie un ou plusieurs objets sur le canvas.

- si un argument tagOrId est un entier, il est traité comme un ID et s'applique seulement à l'objet unique avec cet ID. Voir section (p.).

- si un tel argument est une chaîne, c'est interprétée comme une étiquette et choisit tous les objets qui ont cette étiquette (s'il y en a). Voir section (p.).

7-6 - Methode des Canvas

Tous les canvas supportent ces méthodes :

.addtag_above (*newTag* , *tagOrId*)

Attachés une nouvelle étiquette à l'objet juste au-dessus de celui indiqué par *tagOrId* dans l'ordre d'exposition. L'argument newTag est l'étiquette que vous voulez attacher, comme une chaîne.

.addtag_all (*newTag*)

Attachés l'étiquette donnée *newTag* à tous les objets sur le canvas.

.addtag_below (*newTag* , *tagOrID*)

Attachés une nouvelle étiquette à l'objet juste au-dessous de celui indiqué par *tagOrId* dans l'ordre d'exposition. L'argument newTag est une tag string.

.addtag_closest (*newTag* , *x* , *y* , *halo* =None, *start* =None)

Ajoute une tag à l'objet le plus proche pour examiner la coordonnée (x, y). S'il y a deux ou plus objets à la même distance, le plus haut dans la liste d'exposition est choisi.

Utilisez l'argument de *halo* pour augmenter la taille efficace du point. Par exemple, une valeur de 5 traiterait n'importe quel objet de 5 pixels de (x, y) de chevauchement.

Si on passe un ID dans l'argument *start*, applique le tag à tous les objets de plus haut niveau que *start* dans la l'ordre d'exposition.

.addtag_enclosed (*newTag* , *x1* , *y1* , *x2* , *y2*)

Ajoutez l'étiquette *newTag* à tous les objets complètement présent dans le rectangle dont le coin supérieur gauche est en (x1, y1) et dont le coin inférieur droit est en (x2, y2).

.addtag_overlapping (*newTag* , *x1* , *y1* , *x2* , *y2*)

Comme la méthode précédente, mais affecte tous les objets qui partagent au moins un point avec le rectangle donné.

.addtag_withtag (*newTag* , *tagOrId*)

Ajoute la tag *newTag* à l'objet ou aux objets indiqués par *tagOrId*.

.bbox (tagOrId=None)

Rend un tuple (x1, y1, x2, y2) description d'un rectangle qui inclut tous les objets indiqués par *tagOrId*. Si l'argument est omis, rend un rectangle incluant tous les objets sur le canvas. Le coin supérieur gauche du rectangle est en (x1, y1) et le coin inférieur droit est en (x2, y2).

.canvasx (*screenx* , gridspacing=None)

Traduit une coordonnée de fenêtre x *screenx* en une coordonnée de canvas. Si *gridspacing* est fourni, la coordonnée de canvas est arrondie au plus proche multiple de cette valeur.

.canvasy (*screeny* , gridspacing=None)

Traduit une coordonnée de fenêtre y *screeny* en une coordonnée de canvas. Si *gridspacing* est fourni, la coordonnée de canvas est arrondie au plus proche multiple de cette valeur.

.coords (*tagOrId* , *x0* , *y0* , *x1* , *y1* , ..., *xn* , *yn*)

Si vous passez seulement l'argument *tagOrId*, rend un tuple des coordonnées de l'objet le plus bas indiqué par cet argument. Le nombre de coordonnées dépend du type d'objet. Dans la plupart des cas ce sera un 4-tuple (x1, y1, x2, y2) decrivant la boîte de limitation de l'objet

Vous pouvez déplacer un objet en passant de nouvelles coordonnées.

.dchars (*tagOrId* , first=0, last=first)

Supprime des caractères d'un article de texte ou des articles. Les caractères entre *first* et *last* inclus sont supprimés, où ces valeurs peuvent être des entier ou la chaine "e*nd*" pour signifier la fin du texte. Par exemple, pour un canvas C et un article I, C.dchars (I, 1, 1) enlèvera le deuxième caractère.

.delete (*tagOrId*)

Supprime le ou les objets pointés par *tagOrId*. Il n'y a pas d'erreur si aucun article ne correspond à tagOrId.

.dtag (tagOrId, tagToDelete)

Enlève l'étiquette indiquée par *tagToDelete* de l'objet ou des objets indiqués par *tagOrId*.

.find_above (tagOrId)

Rend le numéro d'ID de l'objet juste au-dessus de celui indiqué par *tagOrId*. Si plusieurs objets correspondent vous obtenez l'ID du plus haut. Rend un tuple vide si vous passez l'ID de l'objet le plus haut.

.find_all()

Rend une liste des numéro d'ID pour tous les objets du canvas, du plus bas au plus haut.

.find_below (*tagOrId*)

Rend le numéro d'ID de l'objet juste au-dessous de celui indiqué par *tagOrId*. Si plusieurs objets correspondent vous obtenez l'ID du plus bas. Rend un tuple vide si vous passez l'ID de l'objet le plus bas.

.find_closest (*x* , *y* , *halo* =None, *start* =None)

Rend un tuple singleton contenant numéro d'ID de l'objet le plus proche du point (*x, y*). S'il n'y a aucun objet correspondant, rend un tuple vide.

Utilisez l'argument *halo* pour augmenter la taille du point. Par exemple, *halo*=5 traiterait n'importe quel objet en 5 pixels (*x, y*) comme le chevauchement.

Si on passe un numéro d'ID à l'argument *start*, cette méthode retourne l'objet rang le plus élevé qui est au-dessous de *start* dans l'ordre d'exposition.

.find_enclosed (*x1* , *y1* , *x2* , *y2*)

Rend une liste des numéros d'ID de tous les objets qui contenu complètement dans le rectangle dont le coin en haut à gauche est (*x1, y1*) et le coin en bas à droite est (*x2, y2*).

.find_overlapping (*x1* , *y1* , *x2* , *y2*)

Comme la méthode précédente, mais retourne une liste une liste des numéros d'ID de tous les objets qui partagent au moins un point avec le rectangle donné.

.find_withtag (*tagOrId*)

Retourne la liste des numéros d'ID de tous les objets spécifiés par *tagOrId.*

.focus (*tagOrId=None*)

Déplace le focus à l'objet indiqué par tagOrId. S'il y a plusieurs objets, déplace le focus au premier dans l'ordre d'exposition qui autorise un curseur d'insertion. S'il aucun article n'est possible, ou si le canvas n'a pas de focus, le focus ne se déplace pas.

Si l'argument est omis, renvoi l'ID de l'objet qui a le focus, ou renvoi "" si aucun objet n'a de focus.

.gettags (*tagOrId*)

Si tagOrId est un ID objet, retourne une liste de toutes les étiquettes associées à cet objet. Si l'argument est une étiquette, retourne toutes les étiquettes de l'objet le plus bas qui a cette étiquette.

.icursor (*tagOrId, index*)

On suppose que l'article choisi permet l'insertion de texte et a le focus, positionne le curseur d'insertion à *index*, qui peut être un entier ou la chaîne "end". N'a aucun effet autrement.

.index (*tagOrId, specifier*)

Renvoi l' entier index du *specifier* donné dans l'article de texte pointé par *tagOrId* (le plus bas, si *tagOrId* spécifie plusieurs objets). La valeur rendue est un entier correspondant à sa position, avec la convention de habituelle Python, où 0 est la position avant le premier caractère.

L'argument *specifier* peut être:

 INSERT, pour obtenir la position du curseur insertion.

 END, pour obtenir la position qui suit le dernier caractère.

 SEL_FIRST, pour obtenir la position du début de la sélection de texte. Tkinter lèvera une exception TclError si l'article de texte ne contient pas de sélection de texte.

 SEL_LAST, pour obtenir la position qui suit la fin de la sélection de texte. Tkinter lèvera une exception TclError si l'article de texte ne contient pas de sélection de texte.

 Une chaine de la forme "@*x, y*". pour obtenir le caractère du caractère contenant les coordonnées (*x,y*) du Canvas. Si ces coordonnées sont au dessus ou à gauche de l'article de texte, la méthode retourne 0; si les coordonnées sont au-dessous ou à droite de l'article, la méthode renvoi l'index de la fin de l'article.

.insert (*tagOrId, specifier, text*)

Insert la chaîne fourni dans l'objet ou les objets spécifiés par *tagOrId*, à la position fourni par *specifier*.

L'argument *specifier* peut être:

 n'importe lequel des mots-clés INSERT, END, SEL_FIRST, ou SEL_LAST. Voir le méthode précedente pour l'interprétation de ces codes.

 la position souhaitée de l'insertion en utilisant la convention normalisée de Python pour la position dans une chaîne.

.itemcget (*tagOrId, option*)

Retourne la valeur de l'option de configuration de l'objet choisi (ou l'objet le plus bas si tagOrId en spécifie plus d'un). C'est semblable à la méthode .cget () pour des objets Tkinter.

.itemconfigure (*tagOrId, option,*)

Si aucun argument d'option n'est fourni, rend un dictionnaire dont les clés sont les options de l'objet indiqué par tagOrId (ou l'objet le plus bas si tagOrId en spécifie plus d'un).

Pour changer l'option de configuration de l'article indiqué, fournissez un ou plusieurs arguments de mot-clé sous forme *option=value*.

.move (*tagOrId, xAmount, yAmount*)

Déplace les articles indiqués par *tagOrId* en ajoutant *xAmount* à leurs coordonnées en *x* et *yAmount* à leurs coordonnées en *y*.

.postscript (*option,* □)

Produit une représentation Abrégée Post-script du contenu actuel du canvas. Les différentes options supportées :

colormode	Utiliser "color" pour colorée, "gray" pour échelle de gris, ou "mono" pour noir et blanc.
file	Si renseigné, nomme un fichier où le PostScript sera écrit. Si on ne donne pas cette option, le PostScript est retourné dans une chaîne.
height	La taille en y utilisée dans le canvas pour écrire. Par défaut c'est tout.
rotate	Si faux, la page sera rendue dans l'orientation portrait; si vrai, en paysage.
x	La coordonnée extrême gauche du secteur du canvas pour imprimer.
y	La coordonnée la plus haute du secteur du canvas pour imprimer.
width	La taille en x utilisée dans le canvas pour écrire. Par défaut c'est tout.

.scale (*tagOrId, xOffset, yOfffset, xScale, yScale*)

Mesure la distance de tous les objets à partir d'un point P = (*xOffset, yOffset*). Les facteurs d'échelle *xScale* et *yScale* sont basés sur une valeur de 1.0, qui ne signifie aucune graduation. Chaque point des objets pointés par tagOrId est déplacé pour que sa distance x de P soit multipliée par xScale et sa distance y soit multipliée par yScale.

Cette méthode ne changera pas la taille d'un article de texte, mais peut le déplacer.

.scan_drago (*x, y, gain=10.0*)

Voir la méthode .scan_mark ()ci-dessous.

.scan_mark (*x, y*)

Cette méthode est utilisée pour mettre en oeuvre le fast scrolling d'un Canvas.Nous présumerons que l'utilisateur pressera, maintiendra un bouton de souris et déplacera ensuite la souris horizontalement et verticalement à une vitesse qui dépend de la distance sur laquelle la souris s'est déplacée.

Pour mettre en oeuvre cette fonction, lier l'événement appui du bouton de la souris à un entraîneur scan_mark (x, y) où x et y sont les coordonnées actuelles de la souris.Liez l'événement <Mouvement> à un entraîneur scan_dragto (*x, y, gain*) où x et y sont les coordonnées actuelles de la souris , tant que le bouton de souris est maintenu.

L'argument *gain* contrôle la vitesse de balayage. Cet argument a une valeur par défaut de 10.0. Utilisez un plus grand nombre pour un balayage plus rapide.

.select_adjust (*oid , specifier*)

Ajuste les frontières de la sélection de texte pour inclure la position donnée par l'argument de *specifier*, dans l'article de texte avec ID de l'objet *oid*.

La sélection actuelle d'ancre est aussi mise à la position indiquée. Voir la méthode *select_from* ci-dessous.

.select_clear ()

Supprime la sélection de texte actuelle. S'il n'y a aucune sélection, ne fait rien.

.select_from (*oid* , *specifier*)

Cette méthode met sélection d'ancre à la position donnée par l'argument *specifier*, dans l'article de texte dont l' ID est donné par *oid*.

Le texte sélectionné sur une le canvas donné est spécifié par trois positions : la position de début, la position de fin et la sélection d'ancre, qui peut être n'importe où à l'interieur de ces deux positions.

Pour changer la position du texte sélectionné, utilisez cette méthode en association avec les méthodes *select_adjust*, *select_from* et *select_to* (q.v)..

.select_item ()

S'il y a une sélection de texte sur ce canvas, retourne l'ID objet de l'article de texte contenant la sélection. S'il n'y a aucune sélection, cette méthode ne retourne None.

.select_to (*oid* , *specifier*)

Cette méthode change la sélection de texte pour qu'il inclus la sélection d'ancre et la position donnée par *specifier* dans l'article de texte dont ID est donne dans *oid.* Pour les valeurs de *specifier*, voir la méthode *insert* ci-dessus.

. tag_bind (*tagOrId, sequence=None, function=None,add=None*)

Lie des événements aux objets du Canvas. Pour l'objet ou les objets pointé par tagOrId, associe la fonction *function* avec l'événement *sequence*. Si l'argument passé est une chaîne commençant avec "+", la nouvelle attache est ajoutée aux attaches existantes pour la *sequence* donnée, autrement la nouvelle attache remplace pour la *sequence* donnée.

Pour des informations générales sur des attaches d'événement, voir section (p.).

Notez que les attaches sont appliquées aux articles qui ont cette étiquette au moment de l'appel de la méthode tag_bind. Si les étiquettes sont enlevées postérieurement de ces articles, les attaches persisteront sur ces articles. Si l'étiquette que vous spécifiez est appliquée postérieurement aux articles qui n'avaient pas cette étiquette quand vous avez appelé tag_bind, cette attache ne sera pas appliquée aux articles nouvellement étiquetés.

.tag_lower (*tagOrId, belowThis*)

Déplace l'objet ou les objets pointés par *tagOrId* dans la liste d'affichage à une position juste au-dessous du premier ou seul objet specifié par l' ID *belowThis*.

S'il y a des plusieurs articles avec l'étiquette *tagOrId*, leur ordre d'empilemant relatif est préservé.

Cette méthode n'affecte pas les canvas de fenêtre. Pour changer un ordre d'entassement d'un article de fenêtre, utilisez une méthode inférieur ou d'ascenseur sur la fenêtre.

.tag_raise (*tagOrId, aboveThis*)

Déplace l'objet ou les objets pointés par *tagOrId* dans la liste d'affichage à une position juste au-dessus du premier ou seul objet specifié par l' ID *belowThis*.

S'il y a des plusieurs articles avec l'étiquette *tagOrId*, leur ordre d'empilemant relatif est préservé.

Cette méthode n'affecte pas les canvas de fenêtre. Pour changer un ordre d'entassement d'un article de fenêtre, utilisez une méthode inférieur ou d'ascenseur sur la fenêtre.

.tag_unbind (*tagOrId, sequence, funcId=None*)

Enlève des attaches pour l'entraîneur *funcId* et l'ordre *sequence* de l'objet canvas ou les objets pointés par tagOrId. Voir section (p.).

.type (*tagOrId*)

Renvoi le type du premier ou seul objet pointé par *tagOrId*. La valeur de retour sera une des chaîne "arc", "bitmap", "image", "line", "oval", "polygon", "rectangle", "text", ou "window".

.xview (*MOVETO, fraction*)

Cette méthode fait défiler le Canvas relatif à son image et est destinée à être lier à l'option de commande de scrollbar. Le Canvas est scrollé horizontalement à une position donnée par *offset*, où 0.0 déplace le canvas à sa position extrême gauche et 1.0 à sa position extrême droite.

.xview (*SCROLL, n, what*)

Cette méthode déplace le Canvas de gauche ou droite : l'argument *what* spécifie l'unité du déplacement et peut prendre les valeurs UNITS ou PAGES et n spécifie le nombre d'unités du déplacement du Canvas à droite (ou gauche, si négatif).

La valeur du mouvement pour UNITS est fournis par l'option *xscrollincrement*; voir section (p.).

Pour des mouvements par PAGES, n est multiplié par à neuf dizièmes de la largeur du canvas.

.xview_moveto (*fraction*)

Cette méthode fait défiler le Canvas dans la même façon que .xview (*MOVETO, fraction*).

.xview _scroll (*n, what*)

Identique à .xview (*SCROLL, n, what*).

.yview (*MOVETO, fraction*)

Le défilement vertical équivalent à .xview (*MOVETO, fraction*).

. yview (*SCROLL, n, what*)

Le défilement vertical équivalent à .xview (*SCROLL, n, what*).

.yview_moveto (*fraction*)

Le défilement vertical équivalent à .xview_moveto (*fraction*).

. yview_scroll (*n, what*)

Le défilement vertical équivalent à .xview (*SCROLL, n, what*) et .xview (*SCROLL, n, what*).

7-7 - Canvas Objets arc

Un Canvas objet d'arc, en sa forme la plus générale, est une tranche en forme de coin extraite d'une ellipse. Cela inclut des ellipses entières et des cercles comme des cas spéciaux. Voir section (p.) pour connaître la géométrie des ellipses.

Pour créer un objet arc sur un Canvas ***C*** utiliser :

```
id=C.create_arc(x0,y0,x1,y1,option,□)
```

Le constructeur renvoi l'ID du nouvel objet arc sur le Canvas *C*.

Le point (x0, y0) est le coin supérieur gauche et (x1, y1) le coin inférieur droit d'un rectangle dans lequel l'ellipse est contenue. Si ce rectangle est carré, vous obtenez un cercle.

Les différentes options supportées :

activedash activefill activeoutline activeoutlinestipple activestipple activewidth	Ces options s'appliquent quand l'arc est dans l'état ACTIF, c'est-à-dire quand la souris est sur l'arc. Par exemple, l'option *activefill* spécifie la couleur intérieure quand l'arc est actif. Pour connaître les valeurs utilisées, voir *dash*, *fill*, *outline*, *outlinestipple*, *stipple* et *width*, respectivement.
dash	Modèle de tiret pour le contour. Voir section (p.).
dashoffset	Compensation de modèle de tiret pour le contour. Voir section (p.).
disableddash disabledfill disabledoutline Disabledoutlinestipple Disabledstipple Disabledwidth	Ces options s'appliquent quand l'arc est dans l'état INACTIF.
extent	Largeur de la tranche en degrés. La tranche commence à l'angle donné par l'option de *start* et s'étend en sens inverse des aiguilles d'une montre de *extent* degrés.
fill	Par défaut, l'intérieur d'un arc est transparent et *fill* = "" sélectionnera ce comportement. Vous pouvez aussi paramétrer cette option à n'importe quelle couleur et l'intérieur de l'arc sera rempli de cette couleur.
offset	Compensation de modèle de tiret pour l'intérieur de l'arc. Voir section (p.).
outline	La couleur de la frontière extérieure de la tranche. Par défaut noir.
outlineoffset	Compensation de modèle de tiret pour le contour. Voir section (p.).
outlinestipple	Si l'option *outline* est utilisée, cette option spécifie qu'un bitmap est utilisée pour le tiret de la frontière. Par défaut noir et le défaut

	peut être paramétré en mettant outlinestipple = "".
start	Angle de départ pour la tranche, en degrés, mesuré de direction +x. Si omis, vous obtenez l'ellipse entière.
state	Cette option est *NORMAL* par défaut. Elle peut être positionné à *HIDDEN* pour rendre l'arc invisible ou *DISABLE* pour griser l'arc et le rendre insensible aux événements.
stipple	Un bitmap indiquant comment l'intérieur de l'arc sera remplit. Par défaut *stipple= ""* (solide). Vous voudrez probablement quelque chose comme *stipple= "gray25"*. N'a aucun effet si l'option *fill* n'a pas été paramétré à une couleur.
style	Par défaut dessine l'arc entier; pour ce style, utiliser *style=PIESLICE*. Pour dessiner seulement l'arc circulaire, utiliser *style=ARC*. Pour dessiner l'arc circulaire et sa corde (ligne droite connectant les 2 pointes de l'arc), utiliser *style=CHORD*.
tags	Si une simple chaîne, l'arc est étiqueté avec chaîne. Utiliser un tuple de chaîne pour accrocher à l'arc des étiquettes multiples. Voir section (p.).
width	Largeur de la frontière extérieure de l'arc. Par défaut 1 pixel.

7-8 - Canvas Objets bitmap

Un Canvas Objet Bitmap est montré par deux couleurs, la couleur de fond (valeur 0) et la couleur de premier plan (valeur 1).

Pour créer un objet bitmap sur un Canvas ***C*** utiliser :

```
id=C.create_bitmap (x,y,*option□)
```

Le constructeur renvoi l'ID du nouvel objet bitmap sur le Canvas *C*.

Les valeurs *x* et *y* indique la position du bitmap

Les différentes options supportées :

activebackground activebitmap activeforeground	Ces options spécifient les valeurs de **background**, **bitmap** et **foreground** quand le bitmap est actif, c'est-à-dire quand la souris est sur le bitmap.
anchor	Le bitmap est placé par rapport au point (*x*, *y*).Par défaut, anchor=CENTER, c'est-à-dire que le bitmap est centré par rapport au point (*x*,*y*). Voir section (p.). Par exemple, si vous spécifiez anchor=NE le bitmap sera placé

	pour que le point (x, y) soit localisé au nord-est (coin droit supérieur) du bitmap.
background	La couleur des bits 0 du bitmap. Par défaut background="", c'est à dire transparent.
bitmap	Le bitmap affiché; voir section (p.).
disabledbackground	Ces options spécifient les valeurs de **background**, **bitmap** et **foreground** quand le bitmap est inactif.
disabledbitmap	
disabledforeground	
foreground	La couleur des bits 1 du bitmap. Par défaut foreground="black", c'est à dire noir.
state	Par défaut state=NORMAL. Positionner à DISABLE pour le griser et le rendre insensible aux événements. Positionner à HIDDEN pour le rendre invisible.
tags	L'étiquette ou les étiquettes associées à l'objet, comme une chaine ou un tuple de chaines. Voir section (p.).

7-9 - Canvas Objets image

Pour créer un objet image sur un Canvas ***C*** utiliser :

```
id=C.create_image (x, y, option□)
```

Le constructeur renvoi l'ID du nouvel objet image sur le Canvas *C*.

Les valeurs *x* et *y* indique la position relative de l'image.

Les différentes options supportées :

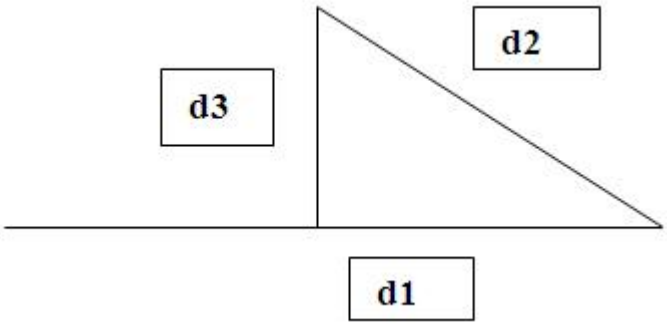
activeimage	L'image affichée quand la souris est sur l'objet. Pour les différentes options, voir ci-dessous **image**.
anchor	Par défaut, anchor=CENTER, c'est-à-dire que l'image est centré par rapport au point (*x*,*y*). Voir section (p.). Par exemple, si vous spécifiez anchor=S l'image sera placé pour que le point (x, y) soit localisé au centre de bord supérieur de l'image.
disabledimage	L'image affichée quand l'objet est inactif. Pour les différentes options, voir ci-dessous **image**.
image	L'image affichée. Voir section (p.) pour plus d'informations sur la façon de créer les images.
state	Par défaut state=NORMAL. Positionner à DISABLE pour le griser et le rendre insensible aux événements. Positionner à HIDDEN pour le rendre invisible.
tags	L'étiquette ou les étiquettes associées à l'objet, comme une chaine. Voir section (p.).

7-10 - Canvas Objets ligne

En général, une ligne peut consister en un certain nombre de segments connectés bout à bout et chaque segment peut être droit ou courbé. Pour créer un objet ligne sur un Canvas ***C*** utiliser :

```
id=C.create_ligne (x0, y0, x1, y1, □, option,□)
```

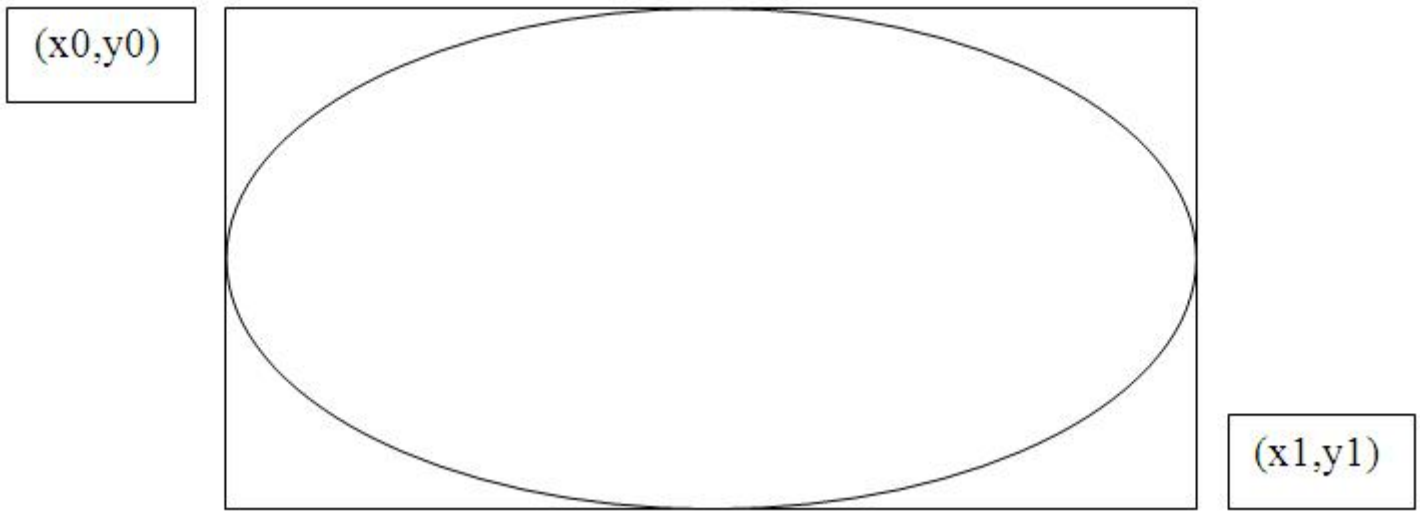
La ligne passe par la série de points (x0, y0), (x1, y1), □ (xn, yn). Les différentes options supportées :

activedash activefill activestipple activewidth	Ces options spécifient le dash, fill, stipple, width lorsque la ligne est active, c'est-à-dire quand la souris est sur l'objet.
arrow	Par défaut la ligne n'a aucune pointe de flèche.Mettre arrow=FIRST pour obtenir une pointe de flèche à l'extremité de la ligne (x0, y0). Mettre arrow=LAST pour obtenir une pointe de flèche à l'extremité de fin. Mettre arrow=BOTH pour obtenir une pointe de flèche à chaque extrémité.
arrowshape	Un tuple (d1, d2, d3) qui décrit la forme des pointes de flèche en complément de l'option **arrow**. Par défaut (8,10,3). 
capstyle	Vous pouvez spécifier la forme des fins de la ligne avec cette option; Voir section (p.). Par défaut à BUTT.
dash	Pour produire une ligne discontinue, spécifiez cette option; Voir section (p.). Par défaut une ligne continue.
dashoffset	Si vous spécifiez un modèle de tiret, par défaut le modèle indiqué doit commencer au début de la ligne. L'option dashoffset vous permet de spécifier que le début du modèle de tiret commence à une distance donnée du le début de la ligne. Voir section (p.).
disableddash disabledfill disabledstipple disabledwidth	Le dash, fill, stipple, width lorsque la ligne est inactive.
fill	La couleur utilisée pour dessiner la ligne. Par défaut fill="black" (noir).
joinstyle	Pour les lignes qui sont composées de plus qu'un segment de ligne, cette option

	contrôle l'apparence de la jonction entre des segments. Voir section (p.). Par défaut ROUND.
offset	Pour des lignes composées, le but de cette option est de correspondre au modèle de l'article avec ceux des objets adjacents. Voir section (p.).
smooth	Si true (vrai), la ligne est dessinée comme une série de splines paraboliques. Par défaut est false (faux) qui dessine la ligne comme un ensemble de segments droits.
splinesteps	Si l'option **smooth** est vraie, chaque spline est rendu comme un certain nombre de segments de ligne droits. L'option splinesteps spécifie le nombre de segments utilisé pour créer chaque section de la ligne; Par défaut splinesteps=12.
state	Par défaut state=NORMAL. Positionner à DISABLE pour le griser et le rendre insensible aux événements. Positionner à HIDDEN pour le rendre invisible.
stipple	Pour dessiner une ligne stippled, associer cette option à un bitmap qui spécifie le modèle de stippling, comme le trait ponctuel = "gray25". Voir section (p.) pour les différentes valeurs.
tags	L'étiquette associée à l'objet, comme une chaine. Voir section (p.).
Width	L'épaisseur de la ligne. Par défaut 1 pixel. Voir section (p.).

7-11 - Canvas Objets oval

Les ovals, mathématiquement, sont des ellipses, y compris les cercles comme un cas spécial. L'ellipse s'inscrit dans un rectangle défini par les coordonnées (x0, y0) du coin en haut à gauche et les coordonnées (x1, y1) du coin en bas droit :



L'ovale coïncidera avec les lignes supérieures et gauches de cette boîte, mais ira juste à l'intérieur des côtés bas et droite.

Pour créer une ellipse sur un Canvas ***C*** utiliser :

```
id=C.create_oval (x0, y0, x1, y1, option,□)
```

qui renvoi l'ID du nouvel objet oval sur le Canvas *C*.

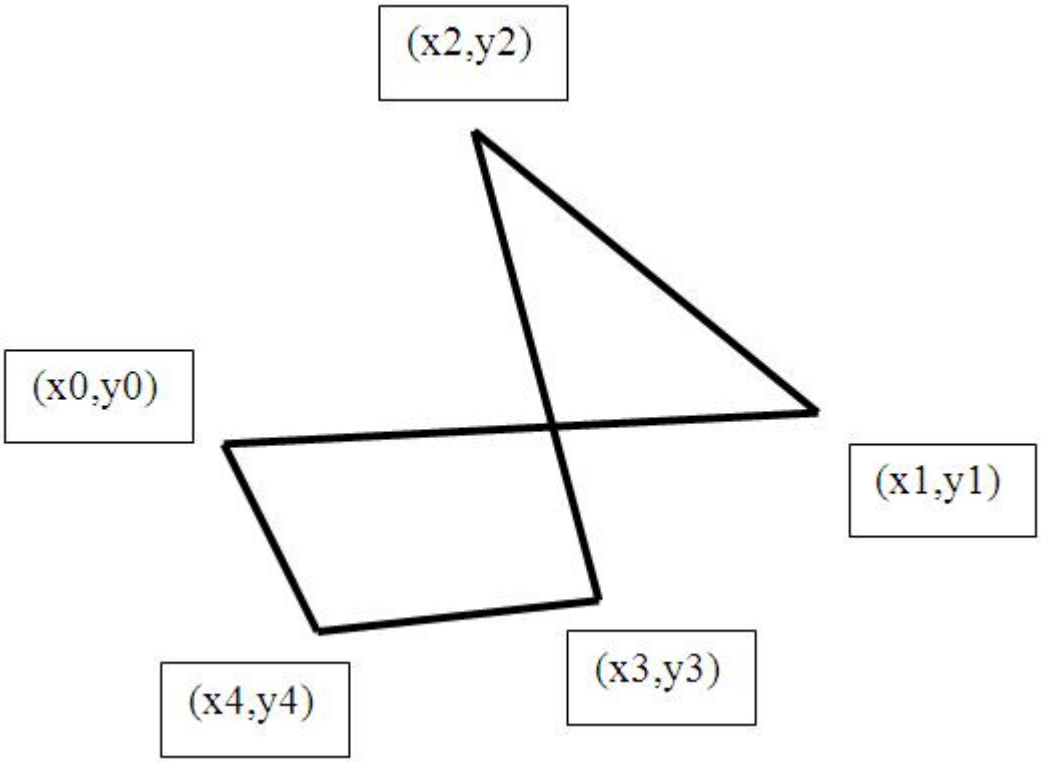
Les options pour oval :

activedash activefill activeoutline activeoutlinestipple activestipple activewidth	Ces options spécifient le modèle de tiret, la couleur de remplissage, le modèle de trait ponctuel, le modèle de trait ponctuel intérieur et la largeur qui seront utilisé quand l'ovale est dans l'état ACTIF, c'est-à-dire quand la souris est sur l'ovale. Pour les valeurs, voir dash, fill, outline, outlinestipple, stipple et width.
dash	Pour produire une ligne discontinue autour l'ovale, spécifiez cette option; Voir section (p.).
dashoffset	En utilisant l'option de dash, l'option dashoffset est utilisée pour changer l'alignement du modèle de tiret à la frontière de l'ovale. Voir section (p.).
disableddash disabledfill disabledoutline disabledoutlinestipple disabledstipple disabledwidth	Ces options spécifient l'apparence des ovales quand l'état de l'objet est DISABLED.
fill	Par défaut l'apparence intérieur d'un oval est transparente et une valeur fill = "" sélectionnera ce comportement. Vous pouvez utiliser cette option pour remplir l'intérieur de l'ellipse de n'importe quelle couleur ; voir section (p.).
offset	Compensation de modèle de trait de l'intérieur. Voir section (p.).
outline	La couleur de la frontière extérieure de l'ellipse. Par défaut outline="BLACK" (noir).
outlineoffset	Compensation de modèle de trait de la frontière.Voir section (p.).
stipple	Un bitmap indiquant comment l'intérieur de l'ellipse sera dessinée. Par défaut stipple="" qui signifie une couleur unie. Une valeur typique "gray25". N'a aucun effet si l'option **fill** n'est pas paramétré à une couleur. Voir section (p.).
outlinestipple	Le modèle de trait utilisé pour la frontière. Pour les valeurs, voir stipple ci-dessus.
state	Par défaut state=NORMAL. Positionner à DISABLE pour le rendre insensible aux

	événements. Positionner à HIDDEN pour le rendre invisible.
tags	L'étiquette associée à l'objet, comme une chaine. Voir section (p.).
width	Epaisseur de la frontière autour de l'extérieur de l'ellipse. Par défaut 1 pixel. Voir section (p.). Positionné à zéro, le trait disparaît. Positionné à zéro avec un fill transparent , fait totalement disparaître l'ellipse.

7-12 - Canvas Objets polygone

Comme montré, un polygone a deux parties : son contour et son intérieur. Sa géométrie est spécifiée comme une série de sommets [(x0, y0), (x1, y1), □ (xn, yn)], mais le périmètre réel inclut encore un segment de (xn, yn) à (x0, y0). Dans cet exemple, il y a cinq sommets :



Pour créer une polygone sur un Canvas ***C*** utiliser :

```
id=C.create_polygon (x0, y0, x1, y1,□, option,□)
```

Le constructeur renvoi l'ID du nouvel objet polygone sur le Canvas *C*. Les options :

activedash activefill activeoutline activeoutlinestipple	Ces options spécifient l'apprence du polygone lorsqu'il est à l'état ACTIF, c'est-à-dire quand la souris dessus. Pour les valeurs, voir dash, fill, outline, outlinestipple, stipple et width.

activestipple	
activewidth	
dash	Utilisé pour produire une ligne discontinue autour du polygone; Voir section (p.).
dashoffset	Utilisez l'option dashoffset débuter le dash a n'importe quel point du cycle en dehors du début. Voir section (p.).
disableddash	Ces options spécifient l'apparence du polygone à l'état DISABLED.
disabledfill	
disabledoutline	
disabledoutlinestipple	
disabledstipple	
disabledwidth	
fill	Vous pouvez colorier l'intérieur en positionnant cette option à une couleur. Par défaut l'apparence intérieur d'un polygone est transparent et une valeur fill = "" sélectionnera ce comportement.; Voir section (p.).
joinstyle	Cette option contrôle l'apprence des intersections entre deux segments adjacents du polygone . Voir section (p.).
offset	Compensation de modèle de trait de l'intérieur. Voir section (p.).
outline	La couleur de la frontière extérieure. Par défaut outline=" " qui la fait être transparente.
outlineoffset	Compensation de modèle de trait de la frontière.Voir section (p.).
outlinestipple	Utilisez cette option pour obtenir une frontière à traits ponctuels autour du polygone. La valeur d'option doit être un bitmap. Voir section (p.).
smooth	Le contour par défaut utilise des lignes droites pour connecter les sommets; utilisez smooth=0 pour obtenir ce comportement. Si vous utilisez smooth=1, vous obtenez une courbe de spline continue. De plus, si vous mettez smooth=1, vous pouvez faire n'importe quel segment directement en faisant un double des coordonnées à chaque fin de ce segment.
splinesteps	Si l'option smooth est vraie, chaque spline est rendu comme un certain nombre de segments de lignes droites. L'option splinesteps spécifie le nombre de segments utilisés pour reunir chaque section de ligne. Par défaut splinesteps=12
state	Par défaut state=NORMAL. Positionner à HIDDEN pour le rendre invisible ou positionner le à DISABLED pour le rendre insensible aux événements.
stipple	Un bitmap indiquant comment l'intérieur du polygone sera dessiné. Par défaut stipple="" qui signifie une couleur unie. Une valeur typique "gray25". N'a aucun effet si l'option

	fill n'est pas paramétré à une couleur. Voir section (p.).
tags	L'étiquette associée à l'objet, comme une chaine. Voir section (p.).
width	Epaisseur de la frontière. Par défaut 1. Voir section (p.).

7-13 - Canvas Objets rectangle

Chaque rectangle est spécifié comme deux points : (x0, y0) est le coin en haut à gauche et (x1, y1) est l'emplacement du pixel juste à l'extérieur du coin en bas à droite.

Par exemple, le rectangle indiqué par le coin en haut à gauche (100,100) et le coin en bas à droite (102,102) est une surface de deux pixels par deux pixels, y compris le pixel (101,101), mais n'incluant pas (102,102).

Les rectangles sont dessinés en deux parties :

 Le contour se trouve à l'intérieur du rectangle sur ses côtés supérieurs et gauches, mais à l'extérieur du rectangle sur son coté bas et droit. L'apparence par défaut de la frontière est un pixel de large et noir.

Par exemple, considérez un rectangle avec le coin en haut à gauche (10,10) et le coin en bas à droite (11,11). Si vous demandez aucune frontière (width=0) et remplissage vert (fill='green '), vous obtiendrez un pixel vert à (10,10). Cependant, si vous demandez les mêmes options avec une frontière noire (width=1), vous obtiendrez quatre pixels noirs à (10,10), (10,11), (11,10) et (11,11).

 Le remplissant est le secteur à l'intérieur du contour. Son apparence par défaut est transparente.

Pour créer un rectangle sur un Canvas ***C*** utiliser :

```
id=C.create_rectangle (x0, y0, x1, y1, option,)
```

Le constructeur renvoi l'ID du nouvel objet rectangle sur le Canvas *C*. Les options disponibles:

activedash	Ces options spécifient l'apprence du rectangle lorsqu'il est à l'état ACTIF, c'est-à-dire quand la souris dessus. Pour les valeurs, voir dash, fill, outline, outlinestipple, stipple et width.
activefill	
activeoutline	
activeoutlinestipple	
activestipple	
activewidth	
dash	Utilisé pour produire une ligne discontinue autour du polygone; Voir section (p.).
dashoffset	Utilisez l'option dashoffset débuter le dash a n'importe quel point du cycle. Voir section (p.).
disableddash	Ces options spécifient l'apparence du rectangle à l'état DISABLED.
disabledfill	
disabledoutline	
disabledoutlinestipple	
disabledstipple	
disabledwidth	
fill	Par défaut l'apparence intérieur d'un rectangle est transparent et une valeur fill = "" sélectionnera ce comportement.Vous

	pouvez colorier l'intérieur en positionnant cette option à une couleur; Voir section (p.).
offset	Utilisez cette option pour changer la compensation du modèle de trait intérieur. Voir section (p.).
outline	La couleur de la frontière extérieure. Par défaut outline="BLACK" (noir).
outlineoffset	Utilisez cette option pour ajuster la compensation du modèle de trait dans le contour ; Voir section (p.).
outlinestipple	Utilisez cette option pour obtenir une frontière à traits ponctuels autour. La valeur d'option doit être un bitmap. Voir section (p.).
state	Par défaut state=NORMAL. L'état est ACTIVE lorsque la souris est sur le rectangle. Positionner le à DISABLED pour le griser et le rendre insensible aux événements.
stipple	Un bitmap indiquant comment l'intérieur du rectangle sera dessiné. Par défaut stipple="" qui signifie une couleur unie. Une valeur typique "gray25". N'a aucun effet si l'option **fill** n'est pas paramétré à une couleur. Voir section (p.).
tags	L'étiquette associée à l'objet, comme une chaine. Voir section (p.).
width	Epaisseur de la frontière. Par défaut 1 pixel. Utiliser width=0 pour rendre la frontiere invisible. Voir section (p.).

7-14 - Canvas Objets texte

Vous pouvez afficher une ou plusieurs lignes de texte sur un Canvas ***C*** en créant un objet texte:

```
id=C.create_text (x, y, option,□)
```

Le constructeur renvoi l'ID du nouvel objet texte sur le Canvas *C*. Les options disponibles:

activefill	La couleur de texte utilisée quand le texte est actif, c'est-à-dire quand la souris est dessus. Pour des valeurs d'option, voir **fill** ci-dessous.
activestipple	Le modèle de trait ponctuel utilisé quand le texte est actif. Pour les valeurs d'option, voir **stipple** ci-dessous
anchor	Par défaut, anchor=CENTER, c'est-à-dire que le texte est centré verticalement et horizontalement par rapport au point (*x*,*y*). Voir section (p.). Par exemple, si vous spécifiez anchor=SW le texte sera placé

	pour que le point (x, y) soit localisé au sud-ouest (coin inférieur gauche) du texte.
disabledfill	La couleur de texte utilisée quand le texte est DISABLED (inactif) Pour des valeurs d'option, voir **fill** ci-dessous.
disabledstipple	Le modèle de trait ponctuel utilisé quand le texte est inactif. Pour les valeurs d'option, voir **stipple** ci-dessous
fill	Par défaut le texte est noir mais vous pouvez le mettre à n'importe quelle couleur en positionnant l'option fill. Voir section (p.).
font	Si vous n'aimez pas la police de caractères par défaut, mettez cette option à n'importe quelle valeur de police de caractères. Voir section (p.).
justify	Pour l'affichage du texte multi-ligne, cette option contrôlent comment les lignes sont justifiées : LEFT(par défaut) (gauche), CENTER (centré), ou RIGHT (droit).
offset	La compensation de trait à être utilisée dans le rendu du texte. Voir section (p.).
state	Par défaut state=NORMAL. Positionner à DISABLED pour le rendre insensible aux événements ou à HIDDEN pour le rendre invisible.
stipple	Un bitmap indiquant comment texte sera dessiné. Par défaut stipple="" qui signifie une couleur unie. Une valeur typique "gray25". Voir section (p.).
tags	L'étiquette associée à l'objet, comme une chaine. Voir section (p.).
text	Le texte à afficher dans l'objet, comme une chaîne. Utiliser le caractère de retour à la ligne ("\n") pour forcer des sauts de ligne.
width	Si vous ne spécifiez pas d'option width (largeur), le texte sera mis à l'intérieur d'un rectangle aussi long que la ligne la plus longue. Cependant, vous pouvez aussi mettre l'option width (largeur) à une dimension et si nécessaire chaque ligne du texte sera tronquée même au milieu d'un mot, à la largeur indiquée. Voir section (p.).

Vous pouvez changer le texte affiché dans un objet texte.

 Pour récupérer le texte d'un article avec l'ID *I* sur un Canvas *C*, appelez

C.itemcget (*I*, "texte").

 Pour remplacer le texte dans un article avec l'ID *I* sur un Canvas *C* avec le texte d'une chaîne *S*, appelez C.itemconfigure (*I*, text=*S*)

Un certain nombre de méthodes de Canvas vous permettent de manipuler des articles de texte. Voir section (p.) spécialement **dchars**, **focus**, **icursor**, **index**, et **insert**.

7-15 - Canvas Objets fenêtre

Vous pouvez placer n'importe widget Tkinter sur un Canvas en utilisant un Canvas objet fenêtre. Une fenêtre est un secteur rectangulaire qui peut contenir un widget Tkinter. Le widget doit être l'enfant de la même fenêtre au plus haut niveau que le Canvas, ou l'enfant d'un certain widget situé dans la même fenêtre au plus haut niveau.

Si vous voulez mettre des objets multi-widget complexes sur unCanvas, vous pouvez utiliser cette méthode de placer un widget cadre sur le Canvas et placer ensuite d'autres widget à l'intérieur de ce cadre.

Pour créer une nouvelle fenêtre sur un Canvas ***C*** utiliser :

```
id=C.create_window (x, y, option,□)
```

Cela renvoi l'ID du nouvel objet fenêtre sur le Canvas *C*. Les options :

anchor	Par défaut, anchor=CENTER, c'est-à-dire que la fenêtre est centrée par rapport la position (*x*,*y*). Voir section (p.). Par exemple, si vous spécifiez anchor=E la fenêtre sera placé pour que le point (x, y) soit localisé au milieu de son bord droit (est).
height	La hauteur du secteur réservé pour la fenêtre. Si omis, la fenêtre sera de taille pour adapter la hauteur du widget contenu. Voir section (p.).
state	Par défaut state=NORMAL. Positionner à DISABLED pour la rendre insensible aux événements ou à HIDDEN pour la rendre invisible.
tags	L'étiquette associée à l'objet, comme une chaine. Voir section (p.).
width	La largeur du secteur réservé pour la fenêtre. Si omis, la fenêtre sera de taille pour adapter la largeur du widget contenu.
window	Utilisez window=w où w est le widget vous voulez placer sur le Canvas. Si c'est omis initialement, vous pouvez plus tard appeler C.itemconfigure (*id*, window=*w*) pour placer le wiget *w* sur le Canvas, où *id* est l'ID de l'objet fenêtre.

8 - Le widget Checkbutton

Le but d'un widget checkbutton (parfois appelé "checkbox") est de permettre à l'utilisateur de lire et sélection un choix bilatéral. Le graphique ci-dessus montre à quoi ressemble des checkbuttons à l'état off (0) et on (1) dans une mise en oeuvre : c'est une copie d'écran de deux checkbuttons utilise la police de caractères Times à 24 points.

L'indicateur (*indicator*) est la partie du checkbutton qui montre son état et l'étiquette (*label*) est le texte qui apparaît à côté.

 Vous devrez créer une variable de contrôle, une instance de la classe IntVar, donc votre programme peut mettre en doute et positionner l'état du checkbutton. Voir section (p.).

 Vous pouvez aussi utiliser des attaches d'événement pour réagir aux actions de l'utilisateur sur le checkbutton; Voir section (p.).

 Vous pouvez mettre hors de service un checkbutton. Cela change son apparence à grisé et le rend insensible à la souris.

 Vous pouvez vous débarrasser de l'indicateur du checkbutton et faire de l'ensemble du widget un bouton poussoir "push-push" qui semble enfoncé quand il est actionné et semble levé quand il est éteint.

Pour créer un checkbutton dans une fenêtre parent existante ou cadre parent :

```
w=Checkbutton (parent, option,)
```

Le constructeur renvoi un nouveau widget checkbutton . Les options disponibles:

activebackground	La couleur de fond quand le checkbutton est sous le curseur. Voir section (p.).
activeforeground	La couleur de premier plan quand le checkbutton est sous le curseur
anchor	Si le widget peuple un espace plus grand que ses besoins, cette option spécifient où le checkbutton sera positionné dans cet espace.Par défaut, anchor=CENTER. Voir section (p.). Par exemple, si vous spécifiez anchor=NW le widget sera placé dans le coin supérieur gauche.
bg ou background	La couleur normale de fond affichée derrière l'étiquette et l'indicateur. Voir section (p.). Pour l'option bitmap, cela spécifie la couleur affichée pour le bit 0 dans le bitmap.
bitmap	Pour afficher une image monochrome sur un bouton, positionner cette option avec un bitmap; Voir section (p.).
bd ou borderwidth	La taille de la frontière autour de l'indicateur. Par défaut deux pixels. Voir section (p.).
command	Une procédure appelée chaque fois l'utilisateur change l'état de ce checkbutton.
compound	Utilisez cette option pour afficher tant un texte qu'un graphique, qui peut être un bitmap ou une image, sur le bouton. Des valeurs permises décrivent la position relative des graphiques par rapport au texte et peuvent être BOTTOM, TOP, LEFT, RIGHT, or CENTER. Par exemple, compound=LEFT positionne le graphique à gauche du texte
cursor	Si vous mettez un nom de curseur à cette option (voir section (p.)) le curseur de souris

	change pour le modèle sélectionné quand il est sur le checkbutton.
disabledforeground	La couleur de premier plan rend le texte de checkbutton hors service. Par défaut la couleur de premier plan est à trait ponctuel.
font	La police de caractères utilisée pour le texte. Voir section (p.).
fg ou foreground	La couleur du texte. Pour l'option bitmap, cela spécifie la couleur affichée pour le bit 1 dans le bitmap.
height	Le nombre de ligne de texte sur le checkbutton. Par défaut 1.
highlightbackground	La couleur du focus hightlight quand le checkbutton n'a pas de focus. Voir section (p.).
highlightcolor	La couleur du focus hightlight quand le checkbutton a le focus.
highlightthickness	Intensité de la couleur de focus. Par défaut 1. Positionner à 0 pour supprimer l'affichage du focus hightlight.
image	Pour afficher une image graphique sur le bouton, mettez cette option à un objet image. Voir section (p.).
indicatoron	Normalement un checkbutton affiche comme indicateur une boîte qui montre si le checkbutton est sélectionné ou non. Vous pouvez obtenir ce comportement en mettant indicatoron=1. Cependant, si vous mettez indicatoron=0, l'indicateur disparaît et le widget entier devient un bouton poussoir "push-push" qui semble enfoncé quand il est actionné et levé quand il est éteint. Vous pouvez vouloir augmenter la valeur de borderwidth pour mieux voir l'état d'un tel contrôle.
justify	Si le texte contient plusieurs lignes, cette option indique comment justifier le texte : CENTER (centré), LEFT (à gauche), ou RIGHT (à droite).
offrelief	Par défaut, checkbuttons utilise le style RAISED quand le bouton est off; Utilisez cette option pour spécifier un style différent quand le bouton est off. Voir section (p.).
offvalue	Normalement, la variable de contrôle associée d'un checkbutton's sera mise à 0 quand il est off. Vous pouvez fournir une valeur différente de l'état en mettant offvalue à cette valeur.
onvalue	Normalement, la variable de contrôle associée d'un checkbutton's sera mise à 1 quand il est on. Vous pouvez fournir une

	valeur différente de l'état en mettant onvalue à cette valeur.
overrelief	Utilisez cette option pour spécifier un style de relief à affiché quand la souris est sur le checkbutton; voir section (p.).
padx	Combien d'espace à gauche et à droite entre le checkbutton et le texte. Par défaut 1 pixel. Voir section (p.).
pady	Combien d'espace en haut et en bas entre le checkbutton et le texte. Par défaut 1 pixel.
relief	Avec la valeur par défaut, relief=FLAT, le checkbutton ne ressort pas de son contexte. Vous pouvez positionner cette option à n'importe lequel des autres styles (voir section (p.)) ou utilisez relief=SOLID, qui vous donne un cadre noir solide autour.
selectcolor	La couleur du checkbutton quand il est actionné. Par défaut selectcolor = "red" (rouge).
selectimage	Si vous mettez cette option à une image, cette image apparaîtra dans le checkbutton quand il est actionné. Voir section (p.).
state	Par défaut state=NORMAL. Positionner à DISABLED pour la rendre insensible aux événements. Si le curseur est sur le checkbutton, l'état est ACTIF.
takefocus	Par défaut le contrôle de clavier agit sur le checkbutton (voir section (p.)). Si vous mettez takefocus=0, le focus ne sera pas actionné.
text	L'étiquette afficher près du checkbutton. Utiliser le caractère de retour à la ligne ("\n") pour forcer des sauts de ligne.
textvariable	Si vous devez changer l'étiquette sur un checkbutton pendant l'exécution, mettre à cette variable de contrôle un StringVar pour le gérer (voir section (p.)). A chaque changement de valeur de la variable de contrôle, l'annotation du checkbutton's changera automatiquement aussi.
underline	Avec la valeur par défaut de-1, aucun des caractères du texte de l'étiquette n'est souligné. Indiquer par cette option, la position du caractère à souligner en comptant à partir de zéro.
variable	La variable de contrôle qui suit à la trace l'état actuel du checkbutton ; voir section (p.). Normalement cette variable est un IntVar et 0 signifie off et 1 on, mais voir les options offvalue et onvalue ci-dessus.
width	La largeur par défaut d'un checkbutton est fonction de la taille de l'image ou du texte affiché. Vous pouvez forcer le largeur à un certain nombre de caractères et le checkbutton aura toujours la taille indiquée.
wraplength	Normalement, les lignes ne sont pas enveloppées. Vous pouvez forcer cette

	option à un certain nombre de caractères et toutes les lignes seront brisées avec au maximum ce nombre de caractères.

Les méthodes checkbutton incluent :

.deselect ()

Désélectionne le checkbutton.

.flash ()

Flashe le checkbutton à plusieurs reprises entre ses couleurs actives et normales, mais le laisse à l'état précédent.

.invoke ()

Vous pouvez appeler cette méthode pour obtenir les mêmes actions que si l'utilisateur clique sur le checkbutton pour changer son état.

.select ()

Sélectionne le checkbutton.

.toggle ()

Désélectionne le checkbutton s'il est sélectionné et sélectionne le checkbutton s'il est déselectionné.

9 - Le widget de saisie

Le but d'un widget de saisie est de laisser l'utilisateur voir et modifier une ligne simple de texte.

 Si vous voulez afficher les lignes multiples de texte qui peuvent être éditées, voir section (p.).

 Si vous voulez afficher une ou plusieurs lignes de texte sans être modifié par l'utilisateur, voir section (p.).

Quelques définitions :

 La *sélection* est une région du texte mise en évidence dans un widget de saisie, s'il y en a une.

Typiquement la sélection est faite par l'utilisateur avec la souris et le texte choisi est copié dans le presse-papiers du système. Cependant, Tkinter vous permet de contrôler si vraiment le texte choisi est copié dans le presse-papiers. Vous pouvez aussi choisir de mettre le texte dans une entrée contrôlée par le programme.

 Le *curseur d'insertion* montre où le nouveau texte sera inséré. Il est affiché seulement quand l'utilisateur clique sur la souris quelque part dans le widget. Il apparaît d'habitude comme une ligne verticale clignotante à l'intérieur du widget. Vous pouvez personnaliser son apparence de plusieurs façons.

 On donne des positions dans le texte affiché du widget comme un *index*. Il y a plusieurs façons de spécifier un index :

 Comme l'index Python normal, commençant de 0.

 La constante END se réfère à la position après le texte existant.

 La constante INSERT se réfère à la position actuelle du curseur d'insertion.

 La constante ANCHOR se réfère au premier caractère de la sélection, s'il y a une sélection.

 Vous devez comprendre que la position du caractère dans le widget correspond à une position de souris donnée. Pour simplifier ce processus, vous pouvez utiliser comme index une chaîne de la forme "@n", où n est la distance horizontale en pixels entre le bord gauche du widget de saisie et la souris. Un tel index fera correspondre le caractère à cette position de souris horizontale.

Pour créer un wiget de saisie dans une fenêtre ou un cadre nommé parent :

```
w = Entry (parent, option,)
```

Le constructeur renvoi le nouveau widget de saisie. Les options disponibles:

bg ou background	La couleur normale de fond affichée dans l'aire de saisie. Par défaut gris clair.
bd ou borderwidth	La taille de la frontière autour de l'aire de saisie ; voir section (p.). Par défaut deux pixels.
cursor	Le curseur utilisé quand la souris est dans le widget de saisie; voir section (p.).
disabledbackground	La couleur de fond quand le widget est à l'état DISABLED. Pour les valeurs d'option, voir *bg* ci-dessus.
disabledforeground	La couleur de premier plan quand le widget est à l'état DISABLED. Pour les valeurs d'option, voir *fg* ci-dessous.
exportselection	Par défaut, si vous sélectionnez le texte dans un widget de saisie, il est automatiquement exporté au presse-papiers. Pour éviter cette exportation, utilisez exportselection = 0.
fg ou foreground	La couleur d'affichage du texte. Par défaut noir.
font	La police de caractères utilisée pour le texte saisie. Voir section (p.).
highlightbackground	La couleur du focus hightlight quand le widget n'a pas de focus. voir section (p.).
highlightcolor	La couleur du focus hightlight quand le widget a le focus.
highlightthickness	Intensité de la couleur de focus.
insertbackground	Par défaut, le curseur d'insertion (qui montre le point dans le texte où la nouvelle entrée au clavier sera insérée) est noir. Pour obtenir une couleur différente du curseur d'insertion, mettez insertbackground à n'importe quelle couleur; voir section (p.).
insertborderwidth	Par défaut, le curseur d'insertion est un rectangle simple. Vous pouvez obtenir le curseur avec l'effet de relief RAISED (voir section (p.)) en mettant insertborderwidth à la dimension de la frontière 3-d. Si vous le faites, assurez-vous que l'attribut

	d'insertwidth est au moins deux fois cette valeur.
insertofftime	Par défaut, le curseur d'insertion clignote. Vous pouvez paramétrer insertofftime à une valeur en millisecondes pour spécifier combien de temps le curseur d'insertion s'éteint. Par défaut 300. Si vous utilisez insertofftime=0, le curseur d'insertion ne clignotera pas du tout.
insertontime	Semblable à insertofftime, cet attribut spécifie combien de temps le curseur d'insertion s'allume. Par défaut 600 (millisecondes).
insertwidth	Par défaut, le curseur d'insertion est 2 pixels de large. Vous pouvez l'ajuster en mettant insertwidth à n'importe quelle valeur.
justify	Cette option contrôle comment le texte est justifié quand le texte ne remplit pas la largeur du widget. Les valeurs possibles sont: LEFT par défaut (à gauche), CENTER (centré), ou RIGHT (à droite).
readonlybackground	La couleur de fond affichée quand l'option d'état du widget est "readonly".
relief	Sélectionne les effets tridimensionnels autour de l'entrée de texte. Voir section (p.). Par défaut, relief=SUNKEN.
selectbackground	La couleur de fond utilisée à l'affichage du texte sélectionné. Voir section (p.).
selectborderwidth	La largeur de la frontière utilisé autour du texte sélectionné. Par défaut un pixel.
selectforeground	La couleur de premier plan du texte sélectionné
show	Normalement, les caractères saisispar l'utilisateur apparaissent. Pour faire une saisie "password" (mot de passe) qui affiche chaque caractère comme un astérisque, mettre show = "*".
state	Utilisez cet attribut pour mettre hors de service le widget de saisie, pour que l'utilisateur ne puisse rien y taper. Mettre state=DISABLED pour mettre le widget hors service, state=NORMAL pour permettre la saisie de l'utilisateur. Votre programme peut aussi découvrir si le curseur est actuellement sur le widget en interrogeant cet attribut; Il aura la valeur ACTIVE quand la souris est dessus. Vous pouvez aussi positionner cette option "disabled", qui ressemble à l'état DISABLED (hors service), mais le contenu du widget peut toujours être sélectionné ou copié.
takefocus	Par défaut le contrôle de clavier agit sur le widget de saisie. Positionner cette option à 0 pour sortir le widget de la séquence. Pour une explication du focus, voir section (p.)).
textvariable	Pour être capable pour récupérer le texte actuel de votre widget de saisie, vous devez

	indiquer une variable de contrôle StringVar à cette option; voir section (p.). Vous pouvez récupérer le texte utilisant v.get (), ou le mettre utilisant v.set (), où *v* est la variable de contrôle associée.
width	La taille de l'entrée en caractères. Par défaut 20. Pour des polices de caractères proportionnelles, la longueur physique du widget sera basée sur la largeur moyenne des caractères.
xscrollcommand	Si vous vous attendez à ce que les utilisateurs entrent plus de texte que la taille visible du widget, vous pouvez lier votre widget de saisie avec un scrollbar (barre de défilement). Positionner cette option à la méthode .set du scrollbar. Pour plus d'information voir section (p.).

Les méthodes sur des objets de saisie incluent :

.delete (*first* , *last* =None)

Supprime des caractères du widget, commençant par celui à l'index *first*, jusqu'au caractère *last* sans l'inclure. Si le deuxième argument est omis, seul le caractère à la position *first* est supprimé.

.get()

Renvoi le texte actuel de l'entrée qui est une chaîne.

.icursor (*index*)

Positionne le curseur d'insertion juste avant le caractère à l'*index* donné.

.index (*index*)

Change le contenu de l'entrée pour que le caractère à l'*index* donné soit le caractère visible extrême gauche. N'a aucun effet si le texte rempli entièrement l'entrée.

.insert (*index* , *s*)

Insere la chaîne *s* avant le caractère à l'*index* donné.

.scan_dragto (*x*)

Voir la méthode scan_mark ci-dessous.

.scan_mark (*x*)

Utilisez cette option pour paramétrer le balayage rapide du contenu du widget de saisie qui a un scrollbar qui supporte le scrolling horizontal.

Pour mettre en oeuvre cette fonction, liez l'événement de bouton en bas de la souris à un entraîneur appellé scan_mark (*x*), où *x* est la position actuelle de la souris. Liez alors l'événement <Motion> à un entraîneur appellé scan_dragto (*x*), où *x* est la position actuelle de la souris. La méthode scan_dragto fait défiler le contenu du widget

de saisie continuellement à un taux proportionnel à la distance horizontale entre la position au moment de l'appel de scan_mark et la position actuelle.

.select_adjust (*index*)

Cette méthode est utilisée pour s'assurer que la sélection inclut le caractère à l'*index* spécifié. Si la sélection inclut déjà ce caractère, rien ne se produit. Sinon, la sélection est étendue de sa position actuelle pour inclure la position de l'*index*.

.select_clear()

Vide la sélection. S'il n'y a pas actuellement de sélection, n'a aucun effet.

.select_from (*index*)

Met la position d'index de l'ancre (ANCHOR) au caractère choisi par l'index et sélectionne ce caractère.

.select_present()

S'il y a une sélection, renvoi true (vrais), sinon retourne false (faux).

.select_range (*start* , *end*)

Met la sélection dans le contrôle de programme. Sélectionne le texte commençant à l'index *start*, jusqu'au caractère à l'index *end*, sans l'inclure. La position de début doit être avant la position de fin. Pour choisir tout le texte dans un widget de saisie e, utilisez e.select_range (0, END).

.select_to (*index*)

Sélectionne tout le texte depuis la position de l'ancre (ANCHOR) jusqu'au caractère à l'*index* donné, mais sans l'inclure.

.xview (*index*)

Pareil à .xview (). Cette méthode est utile dans la jonction du widget de saisie à scrollbar horizontal. Voir section (p.).

.xview_moveto (*f*)

Place le texte dans l'entrée pour que le caractère à la position *f* soit placé sur le bord gauche de la fenêtre. L'argument *f* doit être dans [0,1], où 0 représente le bord gauche du texte et 1 la fin à droite.

.xview_scroll (*number* , *what*)

L'argument *what* doit être UNITS, pour faire défiler par des largeurs de caractère, ou PAGES, pour faire défiler par des gros morceaux de la taille du widget de saisie. Le *number* est positif pour faire défiler de gauche à droite, négatif pour faire défiler de droite à gauche. Par exemple, pour un widget de saisie e, e.xview_scroll (-1, PAGES) déplaceraient le texte d'une page à droite et e.xview_scroll (4, UNITS) déplacerait le texte de quatre caractères à gauche.

9-1 - Scroller un widget de saisie

La fabrication d'un widget de saisie déroulant exige un petit code supplémentaire de votre part pour adapter le rappel de service du widget Scrollbar aux méthodes disponibles sur le widget de saisie. Voici quelques fragments de code illustrant l'installation. D'abord, la création et la jonction du widget de saisie et du widget Scrollbar :

```
self.entry = Entry ( self, width=10 )
self.entry.grid(row=0, sticky=E+W)
self.entryScroll = Scrollbar ( self, orient=HORIZONTAL, command=self.__scrollHandler )
self.entryScroll.grid(row=1, sticky=E+W)
self.entry["xscrollcommand"] = self.entryScroll.set
```

Voici la fonction d'adaptateur mentionnée ci-dessus :

```
def __scrollHandler(self, *L):
op, howMany = L[0], L[1]
if op == "scroll":
units = L[2]
self.entry.xview_scroll ( howMany, units )
elif op == "moveto":
self.entry.xview_moveto ( howMany )
```

10 - Le widget Cadre

Un cadre est essentiellement juste un conteneur pour d'autres widgets.

 La fenêtre racine de votre application est essentiellement un cadre.

 Chaque cadre a sa disposition de grille propre, aussi le positionnement de widget dans chaque cadre travail indépendamment.

 Les widgets cadres sont un outil de valeur dans la fabrication de votre application modulaire. Vous pouvez grouper un ensemble de widget liés dans un widget composé en les mettant dans un cadre. Mieux encore, vous pouvez déclarer une nouvelle classe qui hérite du Cadre, y ajoutant votre interface propre. C'est une bonne façon de cacher au monde extérieur les détails d'interactions dans un groupe de widgets liés.

Pour créer un wiget cadre dans une fenêtre ou un cadre nommé parent :

```
w = Frame (parent, option,)
```

Le constructeur renvoi le nouveau widget cadre. Les options disponibles:

bg ou background	La couleur de fond du cadre. Voir section (p.).
bd ou borderwidth	La taille de la frontière du cadre. Par défaut 0 (pas de frontière). Voir section (p.).
cursor	Le curseur utilisé quand la souris est sur le cadre; voir section (p.).
height	La dimension verticale du nouveau cadre. Ce sera ignoré à moins que vous n'appliquiez aussi .grid_propagate (0) au cadre; voir section (p.).
highlightbackground	La couleur du focus hightlight quand le cadre n'a pas de focus. voir section (p.).
highlightcolor	La couleur du focus hightlight quand le cadre a le focus.
highlightthickness	Intensité de la couleur de focus.
padx	Normalement, un Cadre va autour de son contenu. Pour ajouter les pixels N d'espace

	horizontal à l'intérieur du cadre, mettez padx=N.
pady	Utilisé pour ajouter l'espace vertical à l'intérieur d'un cadre. Voir padx ci-dessus.
relief	Le relief par défaut pour un cadre est FLAT (plat), qui signifie que le cadre s'harmonisera avec son environnement. Pour mettre une frontière autour d'un cadre, mettez son borderwidth à une valeur positive et mettez son relief à un des types de relief standard; voir section (p.).
takefocus	Normalement, les widgets cadres ne sont pas concernés par le focus (voir section (p.) pour une vue d'ensemble de ce sujet). Cependant, vous pouvez mettre takefocus=1 si vous voulez que le cadre contrôle l'entrée au clavier. Pour manipuler une telle saisie, vous devrez créer des attaches pour des événements de clavier; voir section (p.) pour plus d'information sur événements et attaches.
width	La dimension horizontale du nouveau cadre. Voir section (p.). Cette valeur sera ignoré à moins que vous n'appelez aussi .grid_propagate (0) au cadre; voir section (p.).

11 - Le widget Etiquette

Les widget étiquette peuvent afficher une ou plusieurs lignes de texte dans le même style, ou un bitmap ou une image. Pour créer un wiget étiquette dans une fenêtre ou un cadre nommé parent :

```
w = Label (parent, option,□)
```

Le constructeur renvoi le nouveau widget étiquette. Les options disponibles:

activebackground	La couleur de fond affichée quand la souris est sur le widget.
activeforeground	La couleur de premier plan affichée quand la souris est sur le widget.
anchor	Cette option contrôle le positionnement du texte si le widget a plus d'espace que les besoins du texte. Par défaut, anchor=CENTER, qui centre le texte dans l'espace disponible. Pour les autres valeurs voir section (p.). Par exemple, si vous spécifiez anchor=NW le texte sera placé

	au coin supérieur gauche de l'espace disponible.
bg ou background	La couleur de fond de l'étiquette. Voir section (p.).
bitmap	Positionnez cette option égale à un bitmap ou un objet image et l'étiquette l'affichera graphique. Voir section (p.) et section (p.).
bd ou borderwidth	Largeur de la frontière autour de l'étiquette; voir section (p.).Par défaut 2 pixels.
compound	Si vous voulez que le widget étiquette affiche tant du texte qu'un graphique (bitmap ou une image), l'option composée spécifie l'orientation relative du graphique par rapport au texte. Les valeurs peuvent être n'importe lequel de LEFT, RIGHT, CENTER, BOTTOM, ou TOP. Par exemple, si vous spécifiez compound=BOTTOM, le graphique sera affiché au-dessous du texte.
cursor	Le curseur qui apparaît quand la souris est sur cette étiquette. Voir section (p.).
disabledforeground	La couleur de premier plan affichée quand l'état du widget est DISABLED (hors service).
font	Si vous affichez du texte dans cette étiquette (avec l'option *text* ou *textvariable*, l'option *font* spécifie dans quelle police de caractères que le texte sera affiché. Voir section (p.).
fg ou foreground	Si vous affichez du texte ou un bitmap dans cette étiquette, cette option spécifie la couleur du texte.Pour l'option bitmap, cela spécifie la couleur affichée pour le bit 1 dans le bitmap. Voir section (p.).
height	Hauteur de l'étiquette en lignes (pas pixels!). Si cette option n'est pas mise, l'étiquette sera de taille pour adapter son contenu.
highlightbackground	La couleur du focus highlight quand le widget n'a pas de focus.
highlightcolor	La couleur du focus highlight quand le widget a un focus.
highlightthickness	Intensité de la couleur de focus.
image	Pour afficher une image statique dans le widget étiquette, mettez cette option à un objet image. Voir section (p.).
justify	Spécifie comment les lignes multiples de texte seront alignées l'une par rapport à l'autre : LEFT pour aligner à gauche,

	CENTER pour centré (par défaut), ou RIGHT pour justifié à droite.
padx	Espace supplémentaire à gauche et à droite du texte dans le widget. Par défaut 1.
pady	Espace supplémentaire dessus et dessous le texte dans le widget. Par défaut 1.
relief	Spécifie l'apparence d'une frontière décorative autour de l'étiquette. Par défaut FLAT (plat) ; voir section (p.).
state	Par défaut un widget étiquette est à l'état NORMAL. Positionner à DISABLED pour le rendre insensible aux événements. L'état est ACTIVE quand la souris est sur le widget.
takefocus	Normalement, le focus ne fait pas de contrôle pour des widget étiquette; voir section (p.). Si vous voulez un contrôle du focus mettre takefocus=1.
text	Pour afficher une ou plusieurs lignes de texte dans un widget étiquette, mettre dans cette option chaîne contenant le texte. Le retour à la ligne ("\n") force un saut de ligne.
textvariable	Pour rattacher le texte affiché dans un widget étiquette à une variable de contrôle de classe StringVar. Voir section (p.).
underline	Vous pouvez afficher un souligné (_) au dessous de le *n*ieme lettre du texte, en comptant de 0, en mettant cette option à *n*. Par défaut underline=-1, qui signifie aucun soulignage.
width	Largeur de l'étiquette en caractères (pas pixels!). Si cette option n'est pas mise, l'étiquette sera de taille pour s'adapter son contenu.
wraplength	Vous pouvez limiter le nombre de caractères à chaque ligne en mettant cette option au nombre désiré. La valeur par défaut, 0, signifie que les lignes seront brisées seulement aux retours à la ligne.

Il n'y a aucune méthode spéciale pour des widget étiquette autres que les méthodes communes (voir section (p.)).

12 - Le widget Cadre Etiquette

Le widget Cadre Etiquette, comme le widget Cadre, est un conteneur spatial un secteur rectangulaire qui peut contenir d'autres widgets. Cependant, à la différence du widget Cadre, le widget Cadre Etiquette vous permet d'afficher une étiquette dans la partie de la frontière autour du secteur.

Voici un exemple d'un widget Cadre Etiquette contenant deux wigets Bouton. Notez que l'étiquette "Important controls" interrompt la frontière. Ce widget illustre le relief par défaut GROOVE (voir section (p.)) et lancre d'étiquette par défaut 'nw', qui place l'étiquette au sommet gauche du cadre.

Pour créer un wiget Cadre Etiquette dans une fenêtre ou un cadre nommé parent :

```
w = LabelFrame (parent, option,□)
```

Le constructeur renvoi le nouveau widget Cadre Etiquette. Les options disponibles:

bg ou background	La couleur de fond de l'étiquette. Voir section (p.).
bd ou borderwidth	Largeur de la frontière autour de l'étiquette; voir section (p.). Par défaut 2 pixels.
cursor	Le curseur qui apparaît quand la souris est sur ce widget. Voir section (p.).
fg ou foreground	La couleur de l'étiquette texte.
height	Hauteur de l'étiquette. Ignoré à moins que vous n'appelliez aussi .grid_propagate (0) au cadre; voir section (p.).
highlightbackground	La couleur du focus highlight quand le widget n'a pas de focus.
highlightcolor	La couleur du focus highlight quand le widget a un focus.
highlightthickness	Intensité de la couleur de focus.
labelanchor	Utilisez cette option pour spécifier la position de l'étiquette à la frontière du widget. La position par défaut est 'nw', qui place l'étiquette au bord gauche de la frontière supérieure. Pour les neuf positions d'étiquette possibles, référez-vous à ce diagramme :
labelwidget	Au lieu d'une étiquette de texte, vous pouvez utiliser n'importe quel widget comme étiquette en passant ce widget en valeur de cette option. Si vous fournissez tant un labelwidget qu'une option texte, l'option de texte est ignorée.
padx	Espace supplémentaire à gauche et à droite du texte dans le widget. Valeur en pixels.
pady	Espace supplémentaire dessus et dessous le texte dans le widget. Valeur en pixels.
relief	Cette option contrôle l'apparence de la frontière autour de l'extérieur du widget. Le

	style par défaut est la GROOVE; voir section (p.).
takefocus	Normalement, le focus ne fait pas de contrôle pour ce widget; fournissez la valeur True (vrai) à cette option pour obtenir un contrôle de focus. Voir section (p.).
text	Texte de l'étiquette.
width	La dimension horizontale du nouveau cadre. Sera ignoré à moins que vous n'appelliez aussi .grid_propagate (0) au cadre; voir section (p.).

13 - Le widget Listbox

Le but d'un widget listbox est d'afficher un ensemble de lignes de texte. Généralement ils sont destinés à permettre à l'utilisateur de choisir un ou plusieurs articles d'une liste. Toutes les lignes de texte utilisent la même police de caractères. Si vous avez besoin de quelque chose ressemblant à un éditeur de texte,voir section (p.).

Pour créer un widget listbox dans une fenêtre ou un cadre nommé parent :

```
w = Listbox (parent, option,□)
```

Le constructeur renvoi le nouveau widget Lisbox. Les options disponibles:

activestyle	Cette option spécifie l'apparence de la ligne active. Il peut avoir n'importe laquelle de ces valeurs : **"underline"** La ligne active est soulignée. C'est l'option par défaut. **"dotbox"** La ligne active est entourée d'une ligne pointillée sur tous les quatre côtés. **"none"** On ne donne aucune apparence spéciale à la ligne active.
bg ou background	La couleur de fond de la listbox.
bd ou borderwidth	Largeur de la frontière autour de la listbox. Par défaut 2 pixels. Voir section (p.).
cursor	Le curseur qui apparaît quand la souris est sur la listbox. Voir section (p.).
disabledforeground	La couleur de premier plan affichée quand l'état du widget est DISABLED (hors service).
exportselection	Par défaut, l'utilisateur sélectionner le texte avec la souris et le texte choisi sera exporté

	au presse-papiers. Pour mettre hors service ce comportement, utilisez exportselection=0.
font	La police de caractères utilisée dans la listbox. Voir section (p.).
fg ou foreground	La couleur du texte de la listbox. Voir section (p.).
height	Nombre de lignes (pas pixels!) affichées dans la listebox. Par défaut 10.
highlightbackground	La couleur du focus highlight quand le widget n'a pas de focus. Voir section (p.).
highlightcolor	La couleur du focus highlight quand le widget a un focus.
highlightthickness	Intensité de la couleur de focus.

listvariable	Un Variable chaîne (StringVar) qui est connecté à la liste complète des valeurs dans la listebox (voir section (p.)).
	Si vous appelez la méthode .get () du listvariable, vous récupérerez une chaîne de la forme "('*v0*', '*v1*'...)", où chaque *vi* est le contenu d'une ligne de la listebox.
	Pour changer le jeu entier de lignes dans la listebox, appelez .set(*s*) du listvariable, où *s* est une chaîne contenant les valeurs de ligne avec des espaces entre eux.
	Par exemple, si listCon est un StringVar associé à l'option listvariable d'une listebox, cet appel ferait la listebox contenir trois lignes :
	`listCon.set("fourmi abeille cigale")`
	Cet appel rendrait la chaîne "('fourmi', 'abeille', 'cigale')" :
	`listCon.get()`

relief	Choisit la frontière tridimensionnelle des effets. Par défaut SUNKEN ; voir section (p.).
selectbackground	La couleur de fond pour utiliser l'affichage le texte choisi.
selectborderwidth	La largeur de la frontière utilisée autour du texte choisi. Par défaut on montre l'article choisi dans un bloc solide de couleur selectbackground; Si vous augmentez le selectborderwidth, les entrées sont déplacées plus loin à part et les entrées choisies en relief RAISED (voir section (p.)).
selectforeground	La couleur de premier plan utilisée pour l'affichage du texte choisi.
selectmode	Détermine combien d'articles peuvent être choisis et comment les traînées de souris affectent la sélection : □ **BROWSE**: Normalement, vous pouvez seulement choisir une ligne d'une liste. Si vous cliquez sur un article et traînez ensuite à une ligne différente, la sélection suivra la souris. C'est la valeur par défaut.

	□ **SINGLE**: Vous pouvez seulement choisir une ligne et vous ne pouvez pas traîner la souris - partout où vous cliquez sur le bouton 1, cette ligne est choisie. □ **MULTIPLE**: Vous pouvez choisir n'importe quel nombre de lignes à la fois. Le clic sur n'importe quelle ligne bascule le choix. □ **EXTENDED**: Vous pouvez choisir n'importe quel groupe adjacent de lignes immédiatement en cliquant sur la première ligne et traînant à la dernière ligne.
state	Par défaut un widget listbox est à l'état NORMAL. Positionner à DISABLED pour le rendre insensible aux événements.
takefocus	Normalement, le focus contrôle le widget listbox; fournissez la valeur 0 à cette option pour obtenir l'interruption du contrôle de focus. Voir section (p.).
width	Largeur de l'étiquette en caractères (pas pixels!). La largeur est basée sur la moyenne des caractères, donc quelques chaînes dans des polices de caractères proportionnelles ne peuvent pas aller à cette longueur . Par défaut 20.
xscrollcommand	Si vous voulez permettre à l'utilisateur de faire défiler la liste horizontalement, vous pouvez lier votre widget listbox avec scrollbar horizontal. Positionner cette option avec la méthode .set du scrollbar. Voir section (p.).
yscrollcommand	Si vous voulez permettre à l'utilisateur de faire défiler la liste verticalement, vous pouvez lier votre widget listbox avec scrollbar vertical. Positionner cette option avec la méthode .set du scrollbar. Voir section (p.).

Un jeu spécial de formes d'index est utilisé pour beaucoup de méthodes sur des objets listebox :

□ Si vous spécifiez un index (un entier), il se réfère à la ligne dans la listebox avec cet index, comptant de 0.

□ L'index END se réfère à la dernière ligne dans la listebox.

□ L'index ACTIVE se réfère à la ligne choisie. Si la liste permet des sélections multiples, il se réfère à la ligne qui était dernière choisie.

□ Un index chaîne de la forme "@x, y" se réfère à la ligne la plus proche des coordonnés (x, y) relative au coin gauche supérieur du widget.

Les méthodes sur des objets de listebox incluent :

. activate (*index*)

Sélectionne la ligne spécifiée par l'index donné.

. bbox (*index*)

Renvoi la bounding box de la ligne indiquée par l'index sous forme d'un 4-tuple (*xoffset, yoffset, largeur, hauteur*), où le pixel gauche supérieur de la boîte est (*xoffset, yoffset*) et la largeur et la hauteur donnés en pixels. La valeur de largeur rendue inclut seulement la partie de la ligne occupée par le texte.

Si la ligne indiquée par l'index n'est pas visible, cette méthode rend None. Si c'est partiellement visible, la bounding box rendue peut s'étendre à l'extérieur du secteur visible.

. curselection ()

Rend un tuple le contenant des numéros de ligne de l'élément ou des éléments choisis, en comptant depuis 0. Si rien n'est choisi, rend tuple vide.

. delete (*first, last=None*)

Supprime les lignes dont les indexs sont dans la gamme [*first, last*], y compris le dernier (contrairement à l'idiome de Python habituel, où l'effacement s'arrête juste avant le dernier index), comptant de 0. Si le deuxième argument est omis, la ligne seule avec l'index *first* est supprimée.

. get (*first, last=None*)

Rend un tuple contenant le texte des lignes avec leur index de *first* à *last* compris. Si le deuxième argument est omis, rend le texte de la seule ligne *first*.

. index (*i*)

Si possible, place la partie visible de la listebox pour que la ligne contenant l'index *i* soit au sommet du widget.

. insert (*index, *elements*)

Insére une ou plusieurs nouvelles lignes dans la listebox avant la ligne indiquée par l'index. Utiliser END comme premier argument si vous voulez ajouter les nouvelles lignes à la fin de la listebox.

. itemcget (*index, option*)

Récupère une des valeurs d'option pour une ligne spécifique dans la listebox. Pour les valeurs d'option, voir itemconfig ci-dessous. Si l'option donnée n'a pas été renseignée pour la ligne donnée, la valeur rendue sera une chaîne vide.

. itemconfig (*index, option=value,* □)

Changez une option de la configuration pour la ligne indiquée par l'index. Les noms d'option incluent :

background

La couleur de fond de la ligne donnée.

foreground

La couleur du texte de la ligne donnée.

selectbackground

La couleur de fond de la ligne sélectionnée.

select foreground

La couleur du texte de la ligne sélectionnée.

. nearest (*y*)

Renvoi l'index de la ligne visible la plus proche de la coordonnée *y* du widget listebox.

.scan_dragto(*x* , *y*))

Voir scan_mark ci-dessous.

.scan_mark(*x* , *y*)

Utilisez cette méthode pour mettre en oeuvre le balayage rapide stable d'une listebox. Pour obtenir cette fonction, liez un certain événement de bouton de souris à un entraîneur qui appelle scan_mark avec la position de souris actuelle. Liez alors l'événement <Motion> (Mouvement) à un entraîneur qui appelle scan_dragto avec la position de souris actuelle et la listebox sera scrollée à un taux proportionnel à la distance entre la position enregistrée par scan_mark et la position actuelle.

.see (*index*)

Ajuste la position de la listebox pour que la ligne mentionnée par l'index soit visible.

.selection_anchor (*index*)

Place "la sélection de l'ancre" sur la ligne choisie par l'argument *index*. Une fois que cette ancre a été placée, vous pouvez vous y référer avec ANCHOR.

Par exemple, pour une listebox nommée lbox, cet ordre choisirait des lignes 3, 4 et 5 :

```
lbox.selection_anchor(3)
lbox.selection_set(ANCHOR,5)
```

.selection_clear (*first* , *last* =None)

Déselectionne toutes les lignes entre les indices *first* et *last*, compris. Si le deuxième argument est omis, déselectionne la ligne avec l'index *first*.

.selection_includes (*index*)

Retourne 1 si la ligne avec l'index donné est sélectionné, sinon rend 0.

.selection_set (*first* , *last* =None)

Sélectionne toutes les lignes entre les indices *first* et *last*, compris. Si le deuxième argument est omis, sélectionne la ligne avec l'index *first*.

.size()

Renvoi le nombre le lignes de la listbox.

.xview()

Pour faire la listebox déroulante horizontalement, mettez l'option de commande de scrollbar horizontal associée à cette méthode.Voir section (p.).

.xview_moveto (*fraction*)

Fait défiler la listebox pour que la *fraction* extrême gauche de la largeur de sa ligne la plus longue soit à l'extérieur du côté gauche de la listebox. *fraction* est dans la gamme [0,1].

.xview_scroll (*number* , *what*)

Fait défiler la listebox horizontalement. Pour l'argument *what*, utilisez UNITS pour faire défiler par caractères, ou PAGES pour faire défiler par pages, c'est-à-dire par largeur de la listebox. L'argument *number* indique de combien faire défiler; les valeurs négatives déplacent le texte vers la droite dans la listebox, les valeurs positives à gauche.

.yview()

Pour faire la listebox déroulante verticalement, mettez l'option de commande de scrollbar verticale associée à cette méthode.Voir section (p.).

.yview_moveto (*fraction*)

Fait défiler la listebox pour que la *fraction* supérieur de la largeur de sa ligne la plus longue soit à l'extérieur du côté gauche de la listebox. *fraction* est dans la gamme [0,1].

.yview_scroll (*number* , *what*)

Fait défiler la listebox verticalement. Pour l'argument *what*, utilisez UNITS pour faire défiler par lignes, ou PAGES pour faire défiler par pages, c'est-à-dire par hauteur de la listebox. L'argument *number* indique de combien faire défiler; les valeurs négatives déplacent le texte de haut en bas dans la listebox, les valeurs positives en haut.

13-1 - Scroller un widget Listbox

Voici un fragment de code illustrant la création et la jonction d'une listebox tant à un scrollbar horizontal que vertical.

```
self.yScroll = Scrollbar ( self, orient=VERTICAL )
self.yScroll.grid ( row=0, column=1, sticky=N+S )
self.xScroll = Scrollbar ( self, orient=HORIZONTAL )
self.xScroll.grid ( row=1, column=0, sticky=E+W )
self.listbox = Listbox ( self,
xscrollcommand=self.xScroll.set,
yscrollcommand=self.yScroll.set )
>self.listbox.grid ( row=0, column=0, sticky=N+S+E+W )
self.xScroll["command"] = self.listbox.xview
self.yScroll["command"] = self.listbox.yview
```

14 - Le widget Menu

Des menus "déroulants" sont une façon populaire de présenter à l'utilisation un certain nombre de choix, cependant il faut prendre l'espace minimal sur l'écran de l'application quand l'utilisateur n'a pas à faire de choix.

· Un *menubutton* est la partie qui apparaît toujours sur l'application.

· Un *menu* est la liste des choix qui apparaît seulement après clics de l'utilisateur sur le menubutton.

· Pour sélectionner un choix, l'utilisateur peut traîner la souris depuis le *menubutton* vers le bas sur un des choix. Alternativement, il peut cliquer et ouvrir le menubutton : les choix apparaîtront et resteront jusqu'à ce que l'utilisateur clique sur un d'entre eux.

· La version Unix de Tkinter supporte au moins les menus ouvrant "tear-off menus". Si vous le souhaitez une ligne pointillée apparait au-dessus des choix. L'utilisateur peut cliquer sur cette ligne pour extraire le menu : une nouvelle fenêtre séparée et indépendante apparaît contenant les choix.

Voir section (p.), ci-dessous, pour savoir comment créer un menubutton et le connecter à un widget menu. D'abord regardons le widget Menu, qui affiche la liste des choix.

Les choix affichés sur un menu peuvent être:

· une commande simple : une chaîne de caractères que l'utilisateur peut sélectionner pour exécuter une opération.

· une *cascade* : une chaîne de caractères ou une image que l'utilisateur peut sélectionner pour montrer un autre menu de choix.

· un Checkbutton (voir section (p.)).

· un groupe de radiobuttons (voir section (p.)).

Pour créer un widget menu, vous devez d'abord avoir créé un Menubutton, que nous appellerons mb :

```
= Menu ( mb, option, ... )
```

w
Le constructeur renvoi le nouveau widget Menu. Les options disponibles:

activebackground	La couleur de fond qui apparaîtra sur un choix quand il est sous la souris. Voir section (p.).
activeborderwidth	Spécifie la largeur d'une frontière dessinée autour d'un choix quand il est sous la souris. Par défaut 1 pixel.Voir section (p.).
activeforeground	La couleur de premier plan qui apparaîtra sur un choix quand il est sous la souris.
bg ou background	La couleur de fond pour les choix qui ne sont pas sous la souris.
bd ou borderwidth	La largeur de la frontière autour de l'ensemble des choix; voir section (p.). Par défaut 1 pixel.
cursor	Le curseur qui apparaît quand la souris est sur les choix, mais seulement quand le menu a été dévelopé. Voir section (p.).
disabledforeground	La couleur du texte pour des articles dont l'état est DISABLED (hors service).
font	La police de caractères par défaut pour choix textuels. Voir section (p.).
fg ou foreground	La couleur de premier plan utilisée pour les choix qui ne sont pas sous la souris.
postcommand	Vous pouvez mettre cette option à une procédure, et cette procédure sera appelée chaque fois quelqu'un appellera ce menu.
relief	le relief par défaut est RAISED. Voir section (p.).
selectcolor	Spécifie la couleur affichée dans les checkbuttons et radiobuttons quand ils sont sélectionnés.
tearoff	Normalement, un menu peut être étiré (ouvert) : la première position (la position 0) dans la liste des choix est occupée par l'élément d'ouverture du menu et les choix supplémentaires sont étirés à partir de la position 1. Si vous mettez tearoff=0, le menu n'aura pas de fonction d'étirement et les choix seront ajoutés en commençant à la position 0.
tearoffcommand	Si vous voulez que votre programme soit notifié quand les clics d'utilisateur se font sur la position d'ouverture dans un menu, mettre cette option à votre procédure. Il sera appelé avec deux arguments : l' ID de la fenêtre parentale et l'ID de la nouvelle fenêtre du menu déroulant.
title	Normalement, le titre d'étirement de la fenêtre de menu sera le même que le texte du menubutton ou de la cascade qui mène à ce menu. Si vous voulez changer le titre de cette fenêtre, renseignez l'option *title* avec cette chaîne.

Ces méthodes sont disponibles sur des objets Menu. Ceux qui créent des choix sur le menu ont leurs options particulières propres; voir section (p.).

.add (*kind* , *coption* , ...)

Ajoute un nouvel élément de type *kind* comme choix disponible suivant dans ce menu. L'argument *kind* peut être n'importe lequelle de "cascade", "checkbutton", "command", "radiobutton", ou "separator". Selon l'argument *kind*, cette méthode est équivalente à .add_cascade (), .add_checkbutton (), et cetera; référez-vous à ces méthodes ci-dessous pour plus de détails.

.add_cascade (*coption* , ...)

Ajoute un nouvel élément en cascade comme choix disponible suivant dans ce menu. Utilisez l'option de menu dans cet appel pour connecter la cascade au menu du niveau suivant, un objet de type le Menu.

.add_checkbutton (*coption* , ...)

Ajoute un nouveau checkbutton comme choix disponible suivant dans ce menu. Les options vous permettent de paramétrer le checkbutton aussi bien que si vous paramétriez un objet Checkbutton; voir section (p.).

.add_command (*coption* , ...)

Ajoutez une nouvelle commande comme choix disponible suivant dans ce menu. Utilisez l'option d'étiquette, de bitmap, ou d'image pour placer du texte ou une image sur le menu; utilisez l'option de commande pour connecter ce choix à une procédure qui sera appelée quand ce choix est sélectionné.

.add_radiobutton (*coption* , ...)

Ajoute un nouveau radiobutton comme choix disponible suivant dans ce menu. Les options vous permettent de paramétrer le radiobutton aussi bien que si vous paramétriez un objet objet Radiobutton; voir section (p.).

.add_separator()

Ajoute un séparateur après la derniere option actuellement définie. C'est juste une ligne horizontale que vous pouvez utiliser pour faire ressortir les groupes de choix. Les séparateurs sont comptés comme des choix, aussi si vous avez déjà trois choix et vous ajoutez un séparateur, le séparateur occupera la position 3 (en comptant de 0).

.delete (*index1* , *index2* =None)

Cette méthode supprime les choix numérotés d'index1 à index2 compris. Pour supprimer un choix, omettez l'argument index2. Vous ne pouvez pas utiliser cette méthode pour supprimer un choix d'étirement , mais vous pouvez le faire en paramétrant l'option de l'objet menu tearoff à 0.

.entrycget (*index* , *coption*)

Pour récupérer la valeur actuelle de *coption* d'un choix, appelez cette méthode avec *index* renseigné avec l'index du choix et *coption* renseigné du nom de l'option désirée.

.entryconfigure (*index* , *coption* , ...)

Pour changer la valeur actuelle de *coption* pour un choix, appelez cette méthode avec *index* renseigné avec l'index de ce choix et un ou plusieurs arguments *coption=value*.

.index (*i*)

Renvoi la position du choix indiqué par l'index *i*. Par exemple, vous pouvez utiliser .index(END) pour trouver l'index du dernier choix (ou None s'il n'y a aucun choix).

.insert_cascade (*index* , *coption* , ...)

Insère une nouvelle cascade à la position donnée par l'index, en comptant de 0. Tous les choix qui suivent cette position se décale vers le bas. Les options sont identiques à celle de .add_cascade (), ci-dessus.

.insert_checkbutton (*index* , *coption* , ...)

Insère un nouveau checkbutton à la position *index*. Les options sont identiques à celle de .add_checkbutton (), ci-dessus.

.insert_command (*index* , *coption* , ...)

Insère une nouvelle commande à la position *index*. Les options sont identiques à celle de .add_command (), ci-dessus.

.insert_radiobutton (*index* , *coption* , ...)

Insère un nouveau radiobutton à la position *index*. Les options sont identiques à celle de .add_radiobutton (), ci-dessus.

.insert_separator (*index*)

Insère un nouveau séparateur à la position *index*.

.invoke (index)

Appelle la commande associé au choix à la position *index*. Si c'est un checkbutton, son état est basculé entre set et clear; si c'est un radiobutton, ce choix est sélectionné.

.post(*x* , *y*)

Affiche ce menu à la position (x, y) de la fenêtre de racine.

.type (*index*)

Rend le type du choix indiqué par *index* : "cascade", "checkbutton", "command",

"radiobutton", "separator", or "tearoff".

.yposition (*n*)

Pour le choix de menu énième, rend la compensation verticale en pixels relative au sommet du menu. Le but de cette méthode est de vous permettre de placer un menu contextuel précisément par rapport à la position actuelle de la souris.

14-1 - Options de création d'article de menu (coption)

Partout où les méthodes de menu décrites ci-dessus permettent un coption, vous pouvez appliquer une valeur à n'importe laquelle des options ci-dessous en utilisant le nom d'option comme un argument mot-clé avec la valeur désirée.Par exemple, pour faire apparaitre le texte d'une commande avec des lettres rouges, utilisez "foreground='red '" comme option à l'appel de méthode de la commande.

accelerator	Pour afficher une combinaison de frappe "d'accélérateur" sur le côté droit d'un choix de menu, utilisez l'option "accelerator=*s*" où *s* est une chaîne contenant les caractères à afficher. Par exemple, pour indiquer qu'une commande a Contrôle-X comme accélérateur, utiliser l'option "accelerator = ' ^ X '". Notez que cette option ne met pas en oeuvre l'accélérateur; utilisez la frappe de touche pour le faire.
activebackground	La couleur de fond utilisée pour les choix quand ils sont sous la souris.
activeforeground	La couleur de premier plan utilisée pour les choix quand ils sont sous la souris.
background	La couleur de fond utilisée pour les choix quand ils ne sont pas sous la souris. Notez que cela ne peut pas être abrégé *bg*.
bitmap	Affichez un bitmap pour ce choix; voir section (p.).
columnbreak	Normalement tous les choix sont affichés dans une longue colonne. Si vous mettez columnbreak=1, ce choix commencera une nouvelle colonne à droite de celui contenant le choix précédent.
columnbreak	Utilisez l'option d'utilisation "columnbreak=True" pour commencer une nouvelle colonne de choix avec ce choix.
command	Une procédure appelée quand ce choix est activé.
compound	Si vous voulez afficher tant du texte qu'un graphique (bitmap ou une image) sur un choix de menu, utilisez ce coption pour spécifier l'emplacement du graphique par rapport au texte. Les valeurs peuvent être LEFT, RIGHT, TOP, BOTTOM, CENTER, ou NONE. Par exemple, une valeur de "compound=TOP" placerait le graphique au-dessus du texte.
font	La police de caractères utilisée pour le texte d'étiquette. Voir section (p.).
foreground	La couleur de premier plan utilisée pour les choix quand ils ne sont pas sous la souris. Notez que cela ne peut pas être abrégé *fg*.
hidemargin	Par défaut, une petite marge sépare les choix adjacents dans un menu. Utilisez le coption "hidemargin=True" pour supprimer cette marge. Par exemple, si vos choix sont

	des échantillons colorés sur une palette, cette option rendra adjacent les échantillons.
image	Affichez une image pour ce choix; Voir section (p.).
label	La chaîne de caractères affichée pour ce choix.
menu	Cette option est utilisée seulement pour des choix en cascade. Indiquer un objet de Menu qui affiche le niveau suivant de choix.
offvalue	Normalement, la variable de contrôle pour un checkbutton est mise à 0 quand le checkbutton est débranché. Vous pouvez changer cette valeur en mettant cette option à la valeur désirée. Voir section (p.).
onvalue	Normalement, la variable de contrôle pour un checkbutton est mise à 1 quand le checkbutton est branché. Vous pouvez changer cette valeur en mettant cette option à la valeur désirée.
selectcolor	Normalement, la couleur affichée dans un checkbutton ou radiobutton est rouge. Vous pouvez changer cette valeur en mettant cette option à la couleur désirée; voir section (p.).
selectimage	Utilisez l'option d'image pour afficher un graphique au lieu du texte sur un menu radiobutton ou checkbutton. Si vous utilisez selectimage=*I*, l'image *I* est affichée quand l'article est sélectionné.
state	Normalement, tous les choix réagissent aux clics de souris, mais vous pouvez mettre state=DISABLED pour griser et rendre insensible un choix. Ce coption sera ACTIF quand la souris est sur le choix.
underline	Normalement aucune des lettres de l'étiquette n'est soulignée. Positionner cette option à l'index d'une lettre pour souligner cette lettre.
value	Spécifie la valeur de la variable de contrôle associée (voir section (p.)) pour un radiobutton. Cela peut être un entier si la variable de contrôle est un IntVar, ou une chaîne si la variable de contrôle est un StringVar.
variable	Pour checkbuttons ou radiobuttons, cette option doitt être paramétré avec la variable de contrôle associée au checkbutton ou au groupe de radiobuttons. Voir section (p.).

15 - Le widget MenuButton

Un menubutton est la partie d'un menu déroulant qui reste l'écran tout le temps. Chaque menubutton est associé à un widget Menu (voir ci-dessus) qui peut afficher les choix de ce menubutton quand l'utilisateur clique dessus.

Pour créer un widget menubutton dans une fenêtre ou un cadre nommé parent :

```
w = Menubutton (parent, option,□)
```

Le constructeur renvoi le nouveau widget menubutton. Les options disponibles:

activebackground	La couleur de fond quand la souris est sur le menubutton. voir section (p.).
activeforeground	La couleur de premier plan quand la souris est sur le menubutton.
anchor	Cette option contrôle où le texte est placé si le widget a plus d'espace que les besoins de texte. Par défaut anchor=CENTER qui centre le texte. Pour les diverses option voir section (p.). Par exemple, si vous utilisez anchor=W, le texte est centré contre le côté gauche du widget.
bg ou background	La couleur de fond quand la souris n'est pas sur le menubutton.
bitmap	Pour afficher un bitmap sur le menubutton, positionnez cette option avec le nom d'un bitmap; voir section (p.).
bd ou borderwidth	Largeur de la frontière autour du menubutton. Par défaut deux pixels. Voir section (p.).
compound	Si vous spécifiez en même temps du texte et un graphique (bitmap ou image), cette option spécifie où le graphique apparaît par rapprot au texte. Les différentes valeurs sont NONE (par défaut), TOP, BOTTOM, LEFT, RIGHT, et CENTER. Par exemple, compound=RIGHT place le graphique à droite du texte. Si vous spécifiez compound=NONE, le graphique est affiché, mais le texte (s'il y en a un) ne l'est pas.
cursor	Le curseur qui apparaît quand la souris est sur ce menubutton. Voir section (p.).
direction	Normalement, le menu apparaîtra au-dessous du menubutton. Mettre direction=LEFT pour afficher le menu à gauche du bouton; utiliser direction=RIGHT pour afficher le menu à droite du bouton; ou

	paramétrer direction ='above ' pour placer le menu au-dessus du bouton.
disabledforeground	La couleur de premier plan affichée sur ce menubutton quand il est mis hors de service.
fg ou foreground	La couleur de premier plan quand la souris n'est pas sur le menubutton.
font	Spécifie la police de caractères utilisée pour afficher le texte; voir section (p.).
height	La hauteur du menubutton en lignes de texte (pas pixels!). Par défaut s'adapte à son contenu.
highlightbackground	Couleur de focus quand le menubutton n'a pas de focus. Voir section (p.).
highlightcolor	Couleur de focus quand le menubutton a le focus.
highlightthickness	Intensité de la couleur de focus.
image	Pour afficher une image sur le menubutton, paramétrer cette option avec l'objet image. Voir section (p.).
justify	Cette option contrôle où le texte est placé quand il ne remplit pas le menubutton : utilisez justify=LEFT pour justifier à gauche le texte (par défaut); utilisez justify=CENTER pour le centrer, ou justify=RIGHT pour justifier à droite.
menu	Pour associer le menubutton à un ensemble de choix, paramétrer cette option avec l'objet Menu contenant ces choix. Cet objet de menu doit avoir été créé en passant menubutton associé au constructeur comme premier argument. Voir ci-dessous exemple montrant comment associer un menubutton et un menu.
padx	Combien d'espace à gauche et à droite du texte du menubutton. Par défaut 1.
pady	Combien d'espace au dessus et en dessous du texte du menubutton. Par défaut 1.
relief	Normalement, menubutton a un apparence de relief (RAISED). Pour tout autre effet voir section (p.).
state	Normalement, un menubutton répond à la souris. state=DISABLED rend le menubutton insensible et gris.
takefocus	Normalement, le contrôle de clavier n'agit pas sur les menubuttons (voir section (p.)) vous pouvez mettre takefocus=True si vous voulez que le menubutton contrôle l'entrée au clavier.
text	Pour afficher le texte sur le menubutton, paramétrer cette option avec la chaîne contenant le texte voulu. Dans la chaîne les retours à la ligne ("\n") causeront des sauts de ligne.
textvariable	Vous pouvez associer une variable de contrôle de classe StringVar avec le menubutton. La modification de cette

	variable de contrôle changera le texte affiché. Voir section (p.).
underline	Normalement, le texte n'est pas souligné sur le menubutton. Pour souligner un des caractères, mettez cette option à l'index de ce caractère.
width	Largeur du menubutton en caractères (pas pixels!). Si cette option n'est pas positionnée, l'étiquette sera de taille à s'adapter son contenu.
wraplength	Normalement, les lignes ne sont pas enveloppées. Vous pouvez mettre cette option à un certain nombre de caractères et toutes les lignes seront brisées au plus à ce nombre.

```
Voici un petit exemple montrant la création
 d'un menubutton et son menu associé avec deux cases à cocher :</paragraph>
self.mb = Menubutton ( self, text="condiments", relief=RAISED )
self.mb.grid()
self.mb.menu = Menu ( self.mb, tearoff=0 )
self.mb["menu"] = self.mb.menu
self.mayoVar = IntVar()
self.ketchVar = IntVar()
self.mb.menu.add_checkbutton ( label="mayo", variable=self.mayoVar )>
self.mb.menu.add_checkbutton ( label="ketchup", variable=self.ketchVar )
```

Cet exemple crée un menubutton étiquetés condiments. Quand ils sont cliqués, les deux checkbuttons étiqueté mayo et ketchup se rabaisseront.

16 - Le widget Message

Ce widget est semblable au widget Etiquette (voir section (p.)), mais il est destiné à afficher des messages sur des lignes multiples. Tout le texte sera affiché dans la même police de caractères; si vous devez afficher le texte avec plus d'une police de caractères, voir section (p.).

Pour créer un widget Message dans une fenêtre ou un cadre nommé parent :

```
w = Message(parent, option,□)
```

Le constructeur renvoi le nouveau widget message. Les options disponibles:

aspect	Utilisez cette option pour spécifier la proportion de la largeur à la hauteur en pourcentage. Par exemple, aspect=100 vous donnerait un texte du message dans un carré; Avec aspect=200, le secteur de texte serait deux fois plus large que haut. La valeur par défaut est 150, c'est-à-dire le texte sera dans une boîte 50 % plus large que haute.
bg ou background	La couleur de fond derrière le texte; voir section (p.).
bd ou borderwidth	Largeur de la frontière autour du widget; voir section (p.). Par défaut deux pixels. Cette

	option est visible seulement quand l'option de relief n'est pas FLAT (plat).
cursor	Spécifie le curseur qui apparaît quand la souris est sur le widget; voir section (p.).
font	Spécifie que la police de caractères utilisée pour afficher le texte dans le widget ; voir section (p.).
fg ou foreground	Spécifie la couleur de texte; voir section (p.).
highlightbackground	Couleur de focus quand le widget n'a pas de focus.
highlightcolor	Couleur de focus quand le widget a le focus.
highlightthickness	Intensité de la couleur de focus.
justify	Utilisez cette option pour spécifier comment les lignes multiples de texte sont alignées. L'utilisation justifie = LEFT pour obtenir une marge gauche alignée; justify=CENTER pour centrer chaque ligne; et justify=RIGHT pour obtenir une marge droite alignée.
padx	Utilisez cette option pour ajouter l'espace supplémentaire à l'intérieur du widget à gauche et à droite du texte. La valeur est en pixels.
pady	Utilisez cette option pour ajouter l'espace supplémentaire à l'intérieur du widget au dessus et en dessous du texte. La valeur est en pixels.
relief	Cette option spécifie l'apparence de la frontière autour de l'extérieur du widget; voir section (p.). Le style par défaut est FLAT (plat).
takefocus	Normalement, le contrôle de clavier n'agit pas sur les widgets Message (voir section (p.)). Utiliser takefocus=True si vous voulez que le widget contrôle l'entrée au clavier.
text	La valeur de cette option est le texte à être affiché à l'intérieur du widget.
textvariable	Si vous voulez être capables de changer le message par programme, associez cette option avec un StringVar (voir section (p.)). La valeur de cette variable est le texte à être affiché. Si vous spécifiez en même temps les option *texte* et *textvariable*, l'option *texte* est ignorée.
width	Utilisez cette option pour spécifier la largeur du secteur de texte dans le widget, en pixels. La largeur par défaut dépend du texte affiché et de la valeur de l'option *aspect*.

17 - Le widget OptionMenu

Le but de ce widget est d'offrir un jeu fixe de choix à l'utilisateur dans un menu déroulant.

Les illustrations ci-dessus montrent un OptionMenu dans deux états. L'exemple de gauche montre le widget en sa forme initiale. L'exemple de droite montre a quoi il ressemble quand la souris a cliqué dessus et déplacé au choix "boat".

Pour créer un widget OptionMenu dans une fenêtre ou un cadre nommé parent :

```
w = OptionMenu (parent, variable, choice1, choice2, ...)
```

Le constructeur renvoi le nouveau widget message. La *variable* est un StringVar (voir section (p.)) qui est associé au widget et les arguments restants sont les choix à être affiché dans le widget en tant que des chaînes.

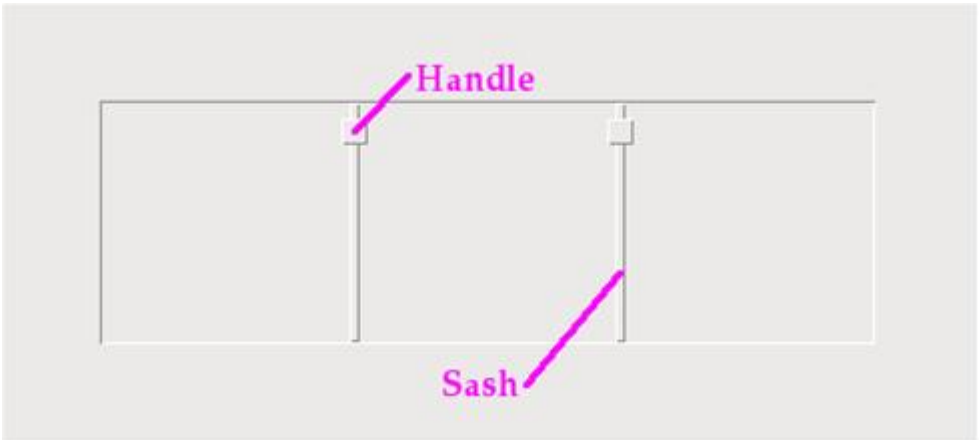
Ce petit bout de code ci-dessus a été créé pour l'illustrer :

```
optionList = ("train", "plane", "boat")
self.v = StringVar()
self.v.set(optionList[0])
self.om = OptionMenu ( self, self.v, *optionList )
```

Pour découvrir quel choix est actuellement sélectionné dans un widget OptionMenu, la méthode .get () sur la variable de contrôle associée rendra ce choix en chaîne.

18 - Le widget PanedWindows (fenêtre à carreau)

Le but du widget PanedWindow est de donner à l'utilisateur de l'application un peu de contrôle sur le partage de l'espace dans la l'application. Un PanedWindow est quelque peu comme un Cadre : c'est un conteneur pour des widgets enfants. Chaque widget PanedWindow contient une pile horizontale ou verticale de widgets enfants. En utilisant la souris, l'utilisateur peut déplacer les frontières entre les widgets enfants dans les deux sens.



 Vous pouvez afficher des poignées (*handle)* dans le widget. Une poignée est un petit carré que l'utilisateur peut traîner avec la souris.

 Vous pouvez faire des ceintures (*sash*) visibles. Une ceinture est une barre placée entre les widgets enfants.

 Un carreau (*pane*) est le secteur occupé par un widget enfant.

Pour créer un widget PanedWindow dans une fenêtre ou un cadre nommé parent :

```
w = PanedWindow ( parent, option, ... )
```

Le constructeur renvoi le nouveau widget PanedWindow. Les options disponibles:

bg ou background	La couleur de fond affichée derrière les widgets enfants; voir section (p.).
bd ou borderwidth	Largeur de la frontière extérieur du widget; voir section (p.). Par défaut deux pixels.
cursor	Le curseur affiché quand la souris est sur le widget; voir section (p.).
handlepad	Utilisez cette option pour spécifier la distance entre la poignée et la fin de la ceinture. Pour orient=VERTICAL, c'est la distance entre la fin gauche de la ceinture et la poignée; pour orient=HORIZONTAL, c'est la distance entre le sommet de la ceinture et la poignée. La valeur par défaut est huit pixels; pour d'autres valeurs, voir section (p.).
handlesize	Utilisez cette option pour spécifier la taille de la poignée, qui est toujours un carré; voir section (p.). La valeur par défaut est huit pixels.
height	Spécifie la hauteur du widget; voir section (p.). Si vous ne spécifiez pas cette option, la hauteur est décidée par la hauteur des widgets enfants.
opaqueresize	Cette option contrôle comment une opération redimensionnante fonctionne. Pour la valeur par défaut, opaqueresize=True, le redimensionnement est fait continuellement comme on traîne la ceinture. Si cette option est mise à False (faux), la ceinture (et les widgets enfants adjacents) reste en place jusqu'à ce que l'utilisateur ne relache le bouton de souris et ensuite il saute à la nouvelle position.
orient	Pour empiler les widgets enfants horizontalement, utilisez orient=HORIZONTAL. Pour les empiler verticalement, utilisez orient=VERTICAL.
relief	Sélectionne le style de relief de la frontière exterieure; voir section (p.). Par défaut FLAT.
sashpad	Utilisez cette option pour répartir l'espace supplémentaire de chaque côté de chaque

	ceinture. Par défaut zéro; pour d'autres valeurs, voir section (p.).
sashrelief	Cette option spécifie que le style de relief utilisépour les ceintures; voir section (p.). Le style par défaut est FLAT.
sashwidth	Spécifie la largeur de la ceinture; voir section (p.). La largeur par défaut est deux pixels.
showhandle	Utilisez showhandle=True pour afficher les poignées. Pour la valeur par défaut, False, l'utilisateur peut toujours utiliser la souris pour déplacer les ceintures. La poignée est simplement une réplique visuelle.
width	Largeur du widget; voir section (p.). Si vous ne spécifiez pas de valeur, la largeur sera définie par les tailles des widgets enfants.

Pour ajouter des widgets enfants à un PanedWindow, créez les widgets enfants comme les enfants du parent PanedWindow, mais plutôt que d'utiliser la méthode .grid () pour les enregistrer, utilisez la méthode .add () sur PanedWindow.

Voici les méthodes sur des widgets PanedWindow.

.add (*child* [, *option* = *value*] ...)

Utilisez cette méthode pour ajouter le widget enfant donné comme l'enfant suivant de ce PanedWindow. Créez d'abord le widget enfant avec le PanedWindow comme son widget parent, mais n'appelez pas la méthode .grid () pour l'enregistrer. Appelez alors .add(enfant) et l'enfant apparaîtra à l'intérieur du PanedWindow dans la position disponible suivante.

Associé à chaque enfant il y a un jeu d'options de configuration qui contrôlent sa position et son apparence. Voir section (p.). Vous pouvez fournir ces options de configuration comme mot-clé argument à la méthode .add ().Vous pouvez aussi mettre ou changer leurs valeurs n'importe quand avec la méthode .paneconfig (), ou récupérer la valeur courante de n'importe laquelle de ces options en utilisant la méthode .panecget (); ces méthodes sont décrites ci-dessous.

.forget (*child*)

Enlève le widget enfant.

.identify (*x* , *y)*

Pour un emplacement donné (x, y) dans les coordonnées de fenêtre, cette méthode rend une valeur qui décrit la caractéristique à cet emplacement.

· Si la caractéristique est une fenêtre enfant, la méthode rend une chaîne vide.

· Si la caractéristique est une ceinture, la méthode rend un tuple (*n*, 'sash') où *n* est 0 pour la première ceinture, 1 pour le deuxième, et cetera.

· Si la caractéristique est une poignée, la méthode rend un tuple (*n*, 'handle') où *n* est 0 pour la première poignée, 1 pour le deuxième, et cetera.

.panecget (*child* , *option*)

Cette méthode récupère la valeur d'une option de configuration de widget enfant, où *child* est le widget enfant et *option* est le nom de l'option en chaîne. Pour la liste des options de configuration de widget enfant, voir section (p.).

.paneconfig (*child* , *option* = *value* , ...)

Utilisez cette méthode pour configurer des options pour des widgets enfant. Les options sont décrites dans la section (p.).

.panes()

Cette méthode rend une liste des widgets enfant, en ordre de gauche à droite (pour orient=HORIZONTAL) ou du haut en bas (pour orient=VERTICAL).

.remove (*child*)

Enlève l'enfant donné; c'est la même action que le .forget ().

.sash_coord (*index*)

Cette méthode rend l'emplacement d'une ceinture. L'argument *index* choisit la ceinture : 0 pour la ceinture entre les deux premiers enfants, 1 pour la ceinture entre le deuxième et troisième enfant, et ainsi de suite. Le résultat est un tuple (x, y) contenant les coordonnées du coin gauche supérieur de la ceinture.

.sash_place (*index* , *x* , *y*)

Utilisez cette méthode de replacer la ceinture sélectionnée par *index* (0 pour la première ceinture, et cetera). Le coordonnées x et y spécifient la nouvelle position voulue du coin gauche supérieur de la ceinture. Tkinter ignore la coordonnée orthogonale de l'orientation du widget : utilisez la valeur de x pour replacer la ceinture pour orient=HORIZONTAL et utiliser la coordonnée y pour déplacer la ceinture pour l'option orient=VERTICAL.

18-1 - Options de configuration des widgets PanedWindows

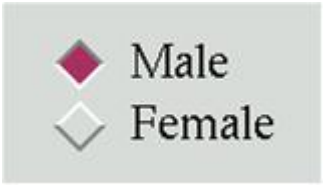
Chaque enfant d'un PanedWindow a un jeu d'options de configuration qui contrôlent sa position et son apparence. On peut fournir ces options au moment d'ajouter un enfant avec la méthode .add (), ou positionner avec la méthode .paneconfig (), ou vérifier avec la méthode .panecget () décrites ci-dessus.

after	Normalement, quand vous .add () un nouvel enfant à un PanedWindow, le nouvel enfant est ajouté après n'importe quel widget enfant existant. Au lieu de cela vous pouvez utiliser l'option after=w pour insérer le nouveau widget à une position juste après un widget enfant existant w.
before	Quand vous utilisez l'option before=w dans un appel à la méthode .add (), le nouveau widget est placé juste avant le widget enfant existant w.
height	Cette option spécifie la hauteur voulue du widget enfant; voir section (p.).
minsize	Utilisez cette option pour spécifier une taille minimale pour le widget enfant dans le sens de l'orientation de PanedWindow. Pour orient=HORIZONTAL, c'est la largeur minimale; pour orient=VERTICAL, c'est

	la hauteur minimale. Pour les valeurs possibles, voir section (p.).
Padx	La quantité d'espace supplémentaire à ajouter à gauche et à droite du widget enfant; voir section (p.).
Pady	La quantité d'espace supplémentaire à ajouter au dessus et au dessous du widget enfant; voir section (p.).
sticky	Cette option fonctionne comme l'argument collant de la méthode .grid (); voir section (p.). Il spécifie comment placer un widget enfant si le carreau est plus grand que le widget. Par exemple, sticky=NW placerait le widget dans le coin supérieur gauche du carreau ("nord-ouest").
Width	Largeur désirée du widget enfant; voir section (p.).

19 - Le widget RadioButton

Un widget Radiobutton est un ensemble de widgets liés qui permettent à l'utilisateur de choisir un seul choix. Chaque radiobutton consiste en deux parties, l'*indicator* (indicateur) et le *label* (étiquette) :



 L'indicateur est la partie en losanges qui devient rouge dans l'article choisi.

 L'étiquette est le texte, bien que vous puissiez utiliser une image ou un bitmap comme étiquette.

 Si vous préférez, vous pouvez vous passer de l'indicateur. Cela fait ressembler le radiobuttons aux boutons "de poussée de poussée", avec l'entrée choisie apparaissant enfoncé et le reste apparaissant en relief.

 Pour former plusieurs radiobuttons dans un groupe fonctionnel, créez une simple variable de contrôle (voir section (p.), ci-dessous) et mettre la variable d'option de chaque radiobutton à cette variable.

La variable de contrôle peut être IntVar ou un StringVar. Si deux ou plus radiobuttons partagent la même variable de contrôle, positionnant n'importe laquel d'entre eux positionnera les autres.

 Chaque radiobutton dans un groupe doit avoir une valeur d'option unique du même type que la variable de contrôle. Par exemple, un groupe de trois radiobuttons pourrait partager un IntVar et avoir les valeurs de 0, 1 et 99. Ou vous pouvez utiliser un StringVar qui contrôle la variable et donnent les valeur d'options de radiobuttons comme "too hot "(trop chaud), "too cold"(trop froid) et "just right"(juste parfait).

Pour créer un widget Radiobutton dans une fenêtre ou un cadre nommé parent :

```
cw = Radiobutton ( parent, option, ... )
```

Le constructeur renvoi le nouveau widget Radiobutton. Les options disponibles:

activebackground	La couleur de fond quand la souris est sur le radiobutton. Voir section (p.).
activeforeground	La couleur de premier plan quand la souris est sur le radiobutton.
anchor	Si le widget peuple un espace plus grand que ses besoins, cette option spécifient où le radiobutton sera positionné dans cet espace. Par défaut anchor=CENTER. Pour d'autres valeurs de positionnement, voir section (p.). Par exemple, si vous mettez anchor=NE, le radiobutton sera placé dans le coin supérieur droit de l'espace disponible.
bg ou background	La couleur normale de fond derrière l'indicateur et l'étiquette.
bitmap	Pour afficher une image monochrome sur un radiobutton, positionner cette option avec un bitmap; Voir section (p.).
bd ou borderwidth	La taille de la frontière autour de la partie de l'indicateur lui-même. Par défaut deux pixels. Voir section (p.).
command	Une procédure appelée chaque fois que l'utilisateur change l'état de ce radiobutton.
compound	Si vous spécifiez en même temps du texte et un graphique (bitmap ou image), cette option spécifie où le graphique apparaît par rapport au texte. Les valeurs possibles sont NONE (valeur par défaut), le TOP (haut), BOTTOM (bas), LEFT (gauche), RIGHT (droit) et CENTER (centre). Par exemple, compound=BOTTOM placerait le graphique au-dessous du texte. Si vous spécifiez compound=NONE, le graphique est affiché, mais le texte (s'il y en a un) ne l'est pas.
cursor	Si vous paramétrez cette option avec un nom de curseur (voir section (p.)), le curseur de souris changera de modèle quand il est sur le radiobutton.
disabledforeground	La couleur de premier plan utilisée quand le bouton est mis hors service. Par défaut une version achurée de la couleur de premier plan par défaut.
font	La police de caractères utilisée pour le texte. Voir section (p.).
fg or foreground	La couleur utilisée pour le texte.
height	Le nombre de lignes (pas pixels) de texte sur le radiobutton. Par défaut 1.
highlightbackground	Couleur de focus quand le widget n'a pas de focus.
highlightcolor	Couleur de focus quand le widget a le focus.
highlightthickness	Intensité de la couleur de focus. Par défaut 1. Positionner highlightthickness = 0 pour supprimer l'affichage du focus.
image	Pour afficher une image graphique au lieu du texte pour ce radiobutton, parametrez cette option avec un objet image. Voir section (p.). L'image apparaît quand le radiobutton n'est

	pas choisi; comparez avec selectimage, ci-dessous.
indicatoron	Normalement un radiobutton affiche son indicateur. Si vous mettez cette option au zéro, l'indicateur disparaît et le widget entier devient un bouton "de poussée de poussée" qui semble en relief quand il est libre et enfoncé quand il est sélectionné. Vous pouvez vouloir augmenter la valeur de borderwidth pour redre plus visible un tel contrôle.
justify	Si le texte contient des lignes multiples, cette option commandes le texte est justifié : CENTER (par défaut), LEFT, ou RIGHT.
offrelief	Si vous supprimez l'indicateur en affirmant indicatoron=False, l'option offrelief spécifie le style de relief affiché quand le radiobutton n'est pas choisi. La valeur par défaut est RAISED. Voir section (p.).
overrelief	Spécifie le style de relief affiché quand la souris est sur le radiobutton.
padx	Combien d'espaces à gauche et à droite entre le bord du radiobutton et le texte. Par défaut 1.
pady	Combien d'espaces au dessus et en dessous entre le bord du radiobutton et le texte. Par défaut 1.
relief	Par défaut, un radiobutton aura le relief FLAT, donc il n'est pas ressorti de son plan. Voir section (p.) pour plus d'informations. Vous pouvez aussi utiliser relief=SOLID, qui affiche un cadre noir solide autour du radiobutton.
selectcolor	La couleur du radiobutton quand il est sélectionné. Par défaut rouge.
selectimage	Si vous utilisez l'option d'image pour afficher un graphique au lieu du texte quand le radiobutton n'est pas choisi, vous pouvez mettre l'option selectimage à une image différente qui sera affichée quand le radiobutton est sélectionné. Voir section (p.).
state	Par défaut state=NORMAL, mais vous pouvez mettre state=DISABLED pour le griser et le rendre insensible. Si le curseur est sur le radiobutton, l'état est ACTIF.
takefocus	Par défaut, le contrôle de clavier agit sur un radiobutton (voir section (p.)). Si vous mettez takefocus=0, le contrôle de clavier ne se fera pas pour le radiobutton.
Text	L'étiquette affichée à côté du radiobutton. Utiliser le retour à la ligne ("\n") pour afficher les lignes multiples de texte.
textvariable	Si vous devez changer l'étiquette sur un radiobutton pendant l'exécution, mettre

	à cette option un StringVar pour gérer sa valeur (voir section (p.)).
underline	Avec la valeur par défaut de-1, aucun des caractères du texte de l'étiquette n'est souligné. Paramétrer cette option à l'index d'un caractère du texte (comptant depuis zéro) pour souligner ce caractère.
value	Quand un radiobutton est allumé par l'utilisateur, sa variable de contrôle est mise à son option de valeur courante. Si la variable de contrôle est un IntVar, donne à chaque radiobutton dans le groupe une option de valeur d'entier différente. Si la variable de contrôle est un StringVar, donne une option de valeur de chaîne différente à chaque radiobutton.
variable	La variable de contrôle que ce radiobutton partage avec les autres radiobuttons dans le groupe; voir section (p.). Cela peut être IntVar ou un StringVar.
width	La largeur par défaut d'un radiobutton est définie par la taille de l'image ou du texte affiché. Vous pouvez paramétrer cette option à un certain nombre de caractères (pas les pixels) et le radiobutton aura toujours la taille définie par ce nombre de caractères.
wraplength	Normalement, les lignes ne sont pas enveloppées. Vous pouvez forcer cette option à un certain nombre de caractères et toutes les lignes seront brisées avec au maximum ce nombre de caractères.

Méthodes sur les widgets Radiobutton :

.deselect()

Eteint le radiobutton.

.flash()

Fait flasher le radiobutton plusieurs fois entre des couleurs actives et normales, mais laisse le bouton dans l'état d'origine.

.invoke()

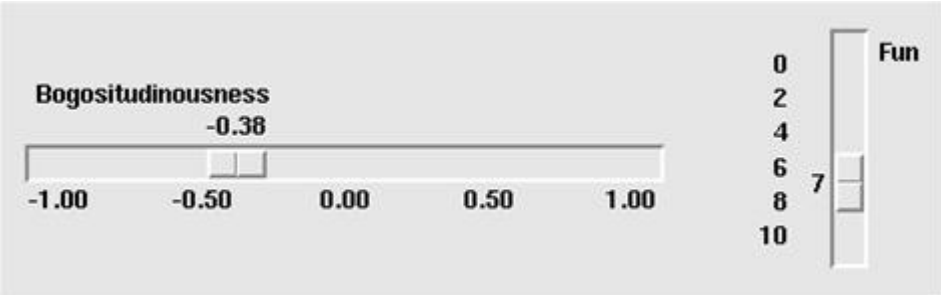
Vous pouvez appeler cette méthode obtenir la même action qui arriverait si l'utilisateur clicque sur le radiobutton pour changer son état.

.select()

Allume le radiobutton.

20 - Le widget Scale (Echelle)

Le but d'un widget échelle est de permettre à l'utilisateur de mettre une certaine valeur d'entier ou de réel dans une gamme indiquée. Voici deux widgets échelle, un horizontal et un vertical :



Chaque échelle affiche un slider (glisseur) que l'utilisateur peut traîner le long d'une cuvette pour changer la valeur. Dans la figure, le premier curseur est actuellement à -0.38 et le deuxième à 7.

 Vous pouvez traîner le slider à une nouvelle valeur avec le bouton 1 de souris.

 Si vous cliquez sur le bouton 1 dans la cuvette, le slider se déplacera de un incrément dans cette direction par clic.Le maintien du bouton 1 dans la cuvette, après un délai, commencera sa fonction auto-répéter.

 Si l'échelle a le contrôle de clavier, les touches flèche gauche et flèche haut déplaceront le slider vers le haut (pour l'échelle verticale) ou vers la gauche (pour l'échelle horizontale). La frappe flèche droite et flèche bas déplaceront le slider vers le bas et vers la droite.

Pour créer un widget échelle dans une fenêtre ou un cadre nommé parent :

```
w= Scale ( parent, option, ... )
```

Le constructeur renvoi le nouveau widget échelle. Les options disponibles:

activebackground	La couleur du slider quand la souris est dessus. Voir section (p.).
bg ou background	La couleur de fond des parties du widget qui sont à l'extérieur de la cuvette.
bd ou borderwidth	Largeur de la frontière 3D autour de la cuvette et du slider. Par défaut deux pixels. Voir section (p.).
command	Une procédure appelée chaque fois le slider est déplacé. Cette procédure passera en argument, la nouvelle valeur d'échelle. Si le slider est déplacé rapidement, vous ne pouvez pas obtenir un rappel de service pour chaque position possible, mais vous obtiendrez un rappel de service quand il s'immobilise.
cursor	Le curseur qui apparaît quand la souris est sur l'échelle. Voir section (p.).
digits	Votre programme lit la valeur courante affichée dans un widget échelle à travers une variable de contrôle; voir section (p.). La variable de contrôle pour une échelle

	peut être un IntVar, un DoubleVar (réel), ou un StringVar. Si c'est une variable chaîne, les options contrôlent le nombre de chiffres utilisés quand la valeur d'échelle numérique est convertie en chaîne.
font	La police de caractères utilisée pour l'étiquette et les annotations. Voir section (p.).
fg ou foreground	La couleur du texte utilisé pour l'étiquette et les annotations.
from_	Un réel ou un entier qui définit une fin de la gamme de l'échelle. Pour l'échelle verticale, c'est la fin supérieure; pour l'échelle horizontale, la fin gauche. L'underbar (_) n'est pas une coquille : parce que *from* est un mot réservé dans Python, cette option est orthographiée from_. Par défaut 0. Voir les options ci-dessous, pour l'autre extremité de la gamme.
highlightbackground	Couleur de focus quand le widget n'a pas de focus. Voir section (p.).
highlightcolor	Couleur de focus quand le widget a le focus.
highlightthickness	Intensité de la couleur de focus.Par défaut 1. Mettre highlightthickness = 0 pour supprimer l'affichage du focus.
label	Vous pouvez afficher une étiquette dans le widget échelle en mettant cette option au texte de l'étiquette. L'étiquette apparaît dans le coin en haut à gauche si l'échelle est horizontale, ou le coin droit supérieur si vertical. Par défaut il n'y a aucune étiquette.
length	La longueur du widget échelle. C'est la dimension x si l'échelle est horizontale, ou la dimension y si vertical. Par défaut 100 pixels. Voir section (p.).
orient	Mettre orient=HORIZONTAL si vous voulez que l'échelle fonctionne le long de la dimension x, ou orient=VERTICAL pour fonctionner parallèle à l'axe des ordonnées. Par défaut vertical.
relief	Avec le relief=FLAT par défaut, l'échelle n'est pas ressortie de son contexte. Vous pouvez aussi utiliser relief=SOLID pour obtenir un cadre noir solide autour de l'échelle, ou n'importe lequel des autres types de relief décrits dans la section (p.).
repeatdelay	Cette option contrôle combien de temps le bouton 1 doit être maintenu dans la cuvette avant que le slider ne commence à se déplacer dans cette direction à plusieurs reprises. Par défaut repeatdelay = 300 et les unités sont des millisecondes.
repeatinterval	Cette option contrôle combien de temps d'attente en millisecondes entre les sauts de slider une fois le bouton 1 maintenu dans la cuvette. Par exemple,

	repeatinterval=100 sauterait le slider toutes les 100 millisecondes.
resolution	Normalement, l'utilisateur sera seulement capable de changer l'échelle dans des unités entières. Mettre cette option à une certaine valeur pour changer le plus petit incrément de la valeur de l'échelle. Par exemple, si from_ =-1.0 et to=1.0 et vous mettez resolution=0.5, l'échelle aura 5 valeurs possibles :-1.0, -0.5, 0.0, +0.5 et +1.0. Tous les mouvements plus petits seront ignorés. Utilisez résolution=-1 pour mettre hors service n'importe quel arrondi de valeurs.
showvalue	Normalement, la valeur courante de l'échelle est affichée sous forme de texte par le curseur (au-dessus des échelles horizontales, vers la gauche pour les échelles verticales). Réglez cette option à 0 pour supprimer cette étiquette.
sliderlength	Normalement le slider est 30 pixels le long de la longueur de l'échelle. Vous pouvez changer cette longueur en mettant l'option sliderlength à la longueur désirée; voir section (p.).
sliderrelief	Par défaut, le slider est affiché avec un style de relief RAISED. Pour mettre d'autres styles, mettez cette option à n'importe laquelle des valeurs décrites dans la section (p.).
state	Normalement les widgets échelle répondent aux événements de la souris, et quand ils ont le focus, au contrôle de clavier.Positionner state=DISABLED pour rendre le widget insensible.
takefocus	Normalement, le focus contrôle les widgets échelle. Positionner cette option à 0 si vous ne voulez pas ce comportement. Voir section (p.).
tickinterval	Normalement, aucune "coche" ne sont affiché le long de l'échelle. Pour afficher des valeurs d'échelle périodiques, mettez cette option à un nombre et des coches seront affichés sur les multiples de cette valeur. Par exemple, si from_ = 0.0, to=1.0 et tickinterval=0.25, les étiquettes seront affichées le long de l'échelle aux valeurs 0.0, 0.25, 0.50, 0.75 et 1.00. Ces étiquettes apparaissent au-dessous de l'échelle si horizontal, à sa gauche si vertical. Par défaut 0, qui supprime l'affichage de coche.
to	Un réel ou un entier qui définit une fin de la gamme de l'échelle; l'autre fin est définie par l'option from_, expliqué susdite. La valeur to peut être plus grand ou moins grand que la valeur de from_. Pour les échelles verticales, le to définit le bas de l'échelle;

	pour les échelles horizontales, la fin à droite. La valeur par défaut est 100.
troughcolor	La couleur de la cuvette.
variable	La variable de contrôle pour cette échelle, si indiquée; voir section (p.). Les variables de contrôle peuvent être de la classe IntVar, DoubleVar, ou StringVar. Dans le dernier cas, la valeur numérique sera convertie en chaîne. Voir l'option digits, ci-dessus, pour plus d'informations sur cette conversion.
width	La largeur de la partie de cuvette du widget. C'est la dimension x pour l'échelle verticale et la dimension y si l'échelle a orient=HORIZONTAL. Par défaut 15 pixels.

Méthodes sur les widgets échelle :

.coords (value=None)

Rend les coordonnées relatives au coin gauche supérieur du widget, correspondant à une valeur donnée de l'échelle. Pour value=None, vous obtenez les coordonnées du centre du slider à sa position actuelle. Pour trouver où le slider serait si la valeur de l'échelle a été mise à une certaine valeur x, utilisez value=x.

.get()

Cette méthode rend la valeur actuelle de l'échelle.

.identify (*x* , *y*)

Etant donné une paire de coordonnées (x, y) relative au coin en haut à gauche du widget, cette méthode rend une identification de type chaîne de quelle partie fonctionnelle du widget est à cet emplacement. La valeur de retour peut être n'importe lequel d'entre ceux-ci:

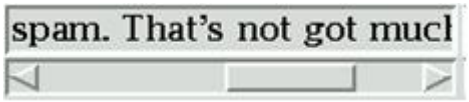
"slider"	Le slider
"trough1"	Pour les échelles horizontales, à gauche du slider; pour les échelles verticales, au-dessus du slider.
"trough2"	Pour les échelles horizontales, à droite du slider; pour les échelles verticales, au-dessous du slider.
""	La position (x, y) n'est sur aucune des susdites parties.

.set (*value*)

Met la valeur de l'échelle.

21 - Le widget Scrollbar

Un certain nombre de widget, comme des listbox et des canvas, peuvent agir comme des fenêtres glissantes dans un plus grand secteur virtuel. Vous pouvez leur connecter des widget scrollbar pour donner à l'utilisateur une possibilité de faire glisser le contenu par rapport au contour. Voici une copie d'écran d'un widget entrée avec un widget scrollbar associé :



 Le widget Scrollbar peut être horizontal, comme celui montré ci-dessus, ou vertical. Un widget qui a deux dimensions déroulantes, comme un canvas ou une listbox, peut avoir en même temps un scrollbar horizontal et un scrollbar vertical.

 Le *slider*, ou le *scroll thumb* (pouce de défilement), est le rectangle en relief qui montre la position actuelle du défilement.

 Les deux pointes de flèche triangulaires à chaque extrémité sont utilisées pour déplacer la position par petits pas. Celui à gauche ou au sommet est appelé arrow1 et celui à droite ou en bas est appelé arrow2.

 La cuvette est le secteur en creux visible derrière les pointes de flèche et slider. La cuvette est divisée en deux secteurs nommés trough1 (au-dessus ou à gauche du slider) et trough2 (au-dessous ou à droite du slider).

 La taille et la position du slider, relativement à la longueur du widget entier, montrent la taille et la position de la vue par rapport à sa taille totale. Par exemple, si un scrollbar vertical est associé à une listbox et son slider s'étend de 50 % à 75 % de la hauteur du scrollbar, cela signifie que la partie visible de la listbox montre la même proportion de la liste complète commençant à la marque à mi-chemin et finissant à la marque de 3/4.

 Dans scrollbar horizontal, en cliquant B1 (bouton 1) sur la pointe de flèche gauche on déplace la vue par une petite quantité à gauche. En cliquant B1 sur la pointe de flèche droite on déplace la vue par cette quantité à droite. Pour un scrollbar vertical, en cliquant les pointes de flèche en haut en bas indique un déplacement de la vue de petites quantités vers le haut ou vers le bas. Référez-vous à la discussion du widget associé pour découvrir la quantité exacte d'actions qui déplacent la vue.

 L'utilisateur peut traîner le slider avec B1 ou B2 (le bouton moyen) pour déplacer la vue.

 Pour un scrollbar horizontal, en cliquant B1 dans la cuvette à gauche du slider on déplace la vue vers la gauche par page et cliquant B1 dans la cuvette à droite du slider déplace la vue vers la droite par une page. Pour un scrollbar vertical, les actions correspondantes déplacent la vue une page vers le haut ou vers le bas.

 cliquer B2 n'importe où le long de la cuvette déplace le slider pour que son extremité gauche ou supérieure soit à la souris, ou aussi près que cela est possible.

La position normalisée du scrollbar se réfère à un nombre dans l'intervalle fermé [0.0, 1.0] qui définit la position du slider. Pour un scrollbar vertical, la position 0.0 est en haut et 1.0 en bas; pour un scrollbar horizontal, la position 0.0 est à l'extrimité gauche et 1.0 à droite.

Pour créer un widget scrollbar dans une fenêtre ou un cadre nommé parent :

```
w = Scrollbar ( parent, option, ... )
```

Le constructeur renvoi le nouveau widget scrollbar. Les options disponibles:

activebackground	La couleur du slider et des pointes de flèche quand la souris est sur eux. Voir section (p.).
activerelief	Par défaut, on montre le slider avec le style de soulagement RAISED. Pour afficher le

	slider avec un style de relief différent quand la souris est sur le slider.
bg ou background	La couleur du slider et des pointes de flèche quand la souris n'est pas sur eux.
bd ou borderwidth	La largeur de la frontière 3D autour du périmètre entier de la cuvette et aussi la largeur des effets 3D sur les pointes de flèche et le slider. Par défaut il ny a aucune frontière autour de la cuvette et une frontière de deux pixels autour des pointes de flèche et du slider. Pour connaître les différentes valeurs possibles, voir section (p.).
command	Une procédure appelée chaque fois que le scrollbar est déplacé. Pour toute explication, voir la section (p.).
cursor	Le curseur qui apparaît quand la souris est sur le scrollbar. Voir section (p.).
elementborderwidth	La largeur des frontières autour des pointes de flèche et du slider. Par défaut elementborderwidth =-1, qui signifie utiliser la valeur de l'option borderwidth.
highlightbackground	Couleur de focus quand le scrollbar n'a pas de focus.. Voir section (p.).
highlightcolor	Couleur de focus quand le scrollbar a le focus.
highlightthickness	Intensité de la couleur de focus. Par défaut 1. Mettre à 0 pour supprimer l'affichage de focus.
jump	Cette option contrôle ce qui arrive quand un utilisateur traîne le slider. Normalement (jump=0), chaque petite traînée du slider cause le rappel de commande callback. Si vous mettez cette option à 1, le rappel de commande n'est pas appelé tant que l'utilisateur ne relache le bouton de souris.
orient	Mettre orient=HORIZONTAL pour scrollbar horizontal, orient=VERTICAL pour un vertical (l'orientation par défaut).
relief	Contrôle le style de relief du widget; le style par défaut est SUNKEN. Cette option n'a aucun effet dans les Fenêtres.
repeatdelay	Cette option contrôle combien de temps le bouton 1 doit être maintenu dans la cuvette avant que le slider ne commence à se déplacer dans cette direction à plusieurs reprises. Par défaut repeatdelay = 300 et les unités sont des millisecondes.
repeatinterval	Cette option contrôle combien de temps d'attente en millisecondes entre les sauts de slider une fois le bouton 1 maintenu dans la cuvette. Par défaut, repeatinterval=100 et les unités sont des millisecondes.
takefocus	Normalement, le contrôle de clavier agit sur le widget scrollbar ; voir section (p.). Mettre takefocus=0 si vous ne voulez pas de ce comportement. Les touches par défaut pour scrollbar permettent à l'utilisateur

	d'utiliser ? et ? pour déplacer les scrollbars horizontaux et d'utiliser ? et ? pour déplacer les scrollbars verticaux.
troughcolor	La couleur de la cuvette.
width	Largeur du scrollbar (sa dimension y si horizontal et sa dimension x si vertical). Par défaut 16. Pour connaître les différentes valeurs possibles, voir section (p.).

Méthodes sur les widgets scrollbar :

.activate(element=None)

Si on ne fournit aucun argument, cette méthode rend une des chaînes "arrow1", "arrow2", "slider", ou "", selon où est la souris. Par exemple, la méthode rend "slider" si la souris est sur le slider. La chaîne est rendue vide si la souris n'est actuellement sur aucune de ces trois objets.

Pour mettre en évidence une des commandes (utilisant son style de relief activerelief et sa couleur d'activebackground), appelez cette méthode et passez une chaîne identifiant le contrôle que vous voulez mettre en évidence, parmis "arrow1", "arrow2", ou "slider".

.delta (*dx* , *dy*)

Etant donné un mouvement de souris de (dx, dy) en pixels, cette méthode rend le réel qui devrait être ajouté à la position courante du slider pour réaliser ce même mouvement. La valeur doit être comprise dans l'intervalle fermé [-1.0, 1.0].

.fraction (*x* , *y*)

Etant donné un emplacement en pixel (x, y), cette méthode retourne la position correspondante normalisée du slider dans l'intervalle [0.0, 1.0] qui est le plus proche de cet emplacement.

.get()

Rend deux nombres (a, b) description de la position actuelle du slider. La valeur *a* donne la position du bord gauche ou supérieur du slider, respectivement pour les scrollbars horizontaux et verticaux; la valeur *b* donne la position du bord bas ou droit. Chaque valeur est dans l'intervalle [0.0, 1.0] où 0.0 est la position extrême gauche ou supérieure et 1.0 est l'extrême droite ou inférieure. Par exemple, si le slider s'étend d'à mi-chemin aux trois-quarts de la voie le long de la cuvette, vous pourriez récupérer le tuple (0.5,0.75).

.identify (*x* , *y*)

Cette méthode rend une chaîne indiquant quel (si existe) des composants du scrollbar est sous la coordonnées donnée (x, y). La valeur de retour est parmis "arrow1", "trough1", "slider", "trough2", "arrow2", ou la chaîne vide "" si cet emplacement n'est sur aucun des composants du scrollbar.

.set (*first* , *last*)

Pour connecter un scrollbar à un autre widget *w*, mettre xscrollcommand ou yscrollcommand du widget *w* de la méthode .set du scrollbar. Les arguments ont la même signification que les valeurs rendues par la méthode .get (). Notez que le déplacement de slider du scrollbar ne déplace pas le widget correspondant.

21-1 - L'appel de la commande de Scrollbar

Quand l'utilisateur manipule un scrollbar, le scrollbar appelle sa commande de rappel de service. Les arguments à cet appel dépendent de ce que l'utilisateur fait :

 Quand l'utilisateur demande un mouvement d'une "unité" gauche ou haut, par exemple en cliquant du bouton B1 à gauche ou la pointe de flèche supérieure, les arguments au rappel de service ressemblent à:

```
command("scroll", -1, "units")
```

 Quand l'utilisateur demande un mouvement d'une unité droit ou bas, les arguments sont :

```
command("scroll", 1, "units")
```

 Quand l'utilisateur demande un mouvement d'une page gauche ou haute :

```
command("scroll", -1, "pages")
```

 Quand l'utilisateur demande un mouvement d'une page droite ou bas :

```
command("scroll", 1, "pages")
```

 Quand l'utilisateur traîne le slider à une valeur f dans la gamme [0,1], où 0 signifie entièrement à gauche ou en haut et 1 signifie entièrement à droite ou en bas, l'appel est :

```
command("moveto", f)
```

Ces ordres d'appel correspondent aux arguments attendus par les méthodes .xview () et .yview () des widgets canvas, listbox et text. Le widget Entrée n'a pas de méthode .xview (). Voir section (p.).

21-2 - Connecter un Scrollbar à un autre widget

Voici un fragment de code montrant la création d'un canvas avec scrollbar horizontal et vertical. Dans ce fragment, *self* est présumé être un widget Cadre.

```
self.canv = Canvas ( self, width=600, height=400, scrollregion=(0, 0, 1200, 800) )
self.canv.grid ( row=0, column=0 )
self.scrollY = Scrollbar ( self, orient=VERTICAL, command=self.canv.yview )
self.scrollY.grid ( row=0, column=1, sticky=N+S )
self.scrollX = Scrollbar ( self, orient=HORIZONTAL, command=self.canv.xview )
self.scrollX.grid ( row=1, column=0, sticky=E+W )
self.canv["xscrollcommand"] = self.scrollX.set
self.canv["yscrollcommand"] = self.scrollY.set
```

Notes:

 La connexion va dans les deux sens. L'option xscrollcommand de canvas doit être connectée à la méthode .set du scrollbar's horizontal et l'option de commande du scrollbar doit être connectée à la méthode .xview du canvas. Le scrollbar vertical et le canvas doivent avoir la même connexion mutuelle.

 Les options sticky (collantes) sur la méthode .grid () appellent au scrollbars de s'étirer juste assez pour s'adapter à la dimension correspondante du canvas.

22 - Le widget Spinbox

Le widget Spinbox permet à l'utilisateur de choisir des valeurs d'un jeu donné. Les valeurs peuvent être une gamme de nombres, ou un jeu fixe de chaînes.

Sur l'écran, un Spinbox a un secteur pour afficher la valeur actuelle et une paire de pointes de flèche.

 L'utilisateur peut cliquer sur la pointe de flèche vers le haut pour avancer à la valeur suivante (plus haut dans l'ordre). Si la valeur est déjà au maximum, vous pouvez paramétrer le widget pour que la nouvelle valeur revienne à la valeur la plus basse.

 L'utilisateur peut cliquer sur la pointe de flèche vers le bas pour avancer à la valeur inférieure suivante dans l'ordre. Cette flèche peut aussi être configurée pour que si la valeur actuelle est la plus basse, en cliquant sur la flèche vers le bas affichera la valeur la plus haute.

 L'utilisateur peut aussi entrer les valeurs directement, traitant le widget comme si c'était une Entrée. L'utilisateur peut déplacer le focus au widget (voir section (p.)), en cliquant dessus ou en utilisant *tab* ou *shift-tab* et éditant ensuite la valeur affichée.

Pour créer un widget spinbox dans une fenêtre ou un cadre nommé parent :

```
w = Spinbox ( parent, option, ... )
```

Le constructeur renvoi le nouveau widget spinbox. Les options disponibles:

activebackground	La couleur du widget quand la souris est dessus. Voir section (p.).
bg ou background	La couleur de fond du widget.
bd ou borderwidth	Largeur de la frontière autour du widget; voir section (p.). La valeur par défaut est un pixel.
buttonbackground	La couleur de fond affichée sur les pointes de flèche. Par défaut gris.
buttoncursor	Le curseur affiché quand la souris est sur les pointes de flèche; voir section (p.).
buttondownrelief	Le style de relief pour la pointe de flèche vers le bas; voir section (p.). Le style par défaut est RAISED.
buttonup	Le style de relief pour la pointe de flèche vers le haut; voir section (p.). Le style par défaut est RAISED.
command	Utilisez cette option pour spécifier une fonction ou une méthode à appeler chaque fois que l'utilisateur clique sur une des pointes de flèche. Notez que le rappel de service n'est pas appelé quand l'utilisateur

	édite la valeur directement comme si c'était une Entrée.
cursor	Le curseur qui est affiché quand la souris est sur la partie d'entrée du widget; voir section (p.).
disabledbackground disabledforeground	Ces options choisissent les couleurs de fond et de premier plan affichées quand l'état du widget est DISABLED (hors service).
exportselection	Normalement, le texte dans la partie d'entrée d'un Spinbox peut être coupé et collé. Pour interdire ce comportement, mettez l'option exportselection à True.
font	Utilisez cette option pour choisir une police de caractères différente pour le texte d'entrée; voir section (p.).
fg ou foreground	Cette option choisit la couleur d'affichage du texte dans la partie d'entrée du widget et la couleur des pointes de flèche.
format	Utilisez cette option pour contrôler le formatage des valeurs numériques en association avec les options *from_* et *to*. Par exemple, le format = '% 10.4f ' afficherait la valeur comme un champ à dix caractères, avec quatre chiffres après la décimale.
from_	Utilisez cette option en association avec l'option *to* (décrit ci-dessous) pour contraindre les valeurs à une gamme numérique. Par exemple, from_ = 1 et to=9 permettrait seulement des valeurs entre 1 et 9 compris. Voir aussi l'option d'incrément ci-dessous.
highlightbackground	La couleur du focus highlight quand le spinbox n'a pas de focus.Voir section (p.).
highlightcolor	La couleur du focus hightlight quand le spinbox a le focus.
highlightthickness	La densité du focus hightlight. Par défaut 1. Positionner à 0 pour supprimer l'affichage du focus hightlight.
increment	Quand vous renseignez les options *from_* et *to*, vous pouvez utiliser l'option increment pour spécifier de combien la valeur augmente ou diminue quand l'utilisateur clique sur une pointe de flèche. Par exemple, avec options from_=0.0, to=2.0 et increment=0.5, l' flèche en haut marchera par 0.0, 0.5, 1.0, 1.5 et 2.0.
insertbackground	Choisit la couleur du curseur d'insertion affiché dans la partie d'entrée du widget.
insertborderwidth	Cette option contrôle la largeur de la frontière autour du curseur d'insertion. Normalement, le curseur d'insertion n'aura aucune frontière. Si cette option est mise à une valeur différente de zéro, le curseur d'insertion sera affiché dans le style de relief RAISED.
insertofftime insertontime	Ces deux options contrôlent le cycle de clignotement du curseur d'insertion : le

	temps passe d'off à on, respectivement, en millisecondes. Par exemple, avec des options insertofftime=200 et insertontime=400, le curseur clignoterait off pendant 0.2 secondes et ensuite on pendant 0.4 secondes.
insertwidth	Utilisez cette option pour spécifier la largeur du curseur d'insertion; pour connaître les valeurs possibles, voir section (p.). La largeur par défaut est deux pixels.
justify	Cette option contrôle la position du texte dans la partie d'entrée du widget. Les valeurs peuvent être LEFT pour justifier le texte à gauche ; CENTER pour le centrer; ou RIGHT de justifier le texte à droite.
readonlybackground	Cette option spécifie la couleur de fond affichée quand l'état du widget est 'readonly'; voir section (p.).
relief	Utilisez cette option pour choisir un style de relief du widget; voir section (p.). Le style par défaut est SUNKEN.
repeatdelay repeatinterval	Ces options spécifient le comportement d'auto-répétition de clics de souris sur les pointes de flèche; Les valeurs sont en millisecondes. La valeur de repeatdelay spécifie combien de temps le bouton de souris doit être maintenu avant qu'il ne se répète et repeatinterval spécifie combien de fois la fonction se répète. Les valeurs par défaut sont 400 et 100 millisecondes, respectivement.
selectbackground	La couleur de fond utilisée pour l'affichage des articles sélectionnés.
selectborderwidth	La largeur de la frontière affichée autour des articles sélectionnés.
selectforeground	La couleur de premier plan utilisée pour l'affichage des articles sélectionnés.
state	Normalement, un widget Spinbox est créé dans l'état NORMAL. Mettre cette option à DISABLED pour rendre le widget insensible à la souris ou aux actions de clavier. Si vous le mettez à 'readonly', la valeur dans la partie d'entrée du widget ne peut pas être modifiée avec des frappes clavier, mais la valeur peut toujours être copiée au presse-papiers et le widget répond toujours aux clics sur les pointes de flèche.
takefocus	Normalement, le contrôle de clavier agit sur la partie d'entrée d'un widget Spinbox (voir section (p.)). Pour enlever ce contrôle au widget, mettre takefocus=False.
textvariable	Si vous voulez récupérer la valeur courante du widget, vous pouvez utiliser la méthode .get () ci-dessous, ou vous pouvez associer une variable de contrôle avec le widget en passant cette variable de contrôle

	comme valeur de cette option. Voir section (p.).
to	Cette option spécifie la limite supérieure d'une gamme estime. Voir l'option *from_*, ci-dessus et aussi l'option *increment*.
values	Il y a deux façons de spécifier les valeurs possibles du widget. Une est de fournir un tuple de chaînes comme valeur de l'option *values*. Par exemple, values=('red ',' blue ',' green ') permettraient seulement ces trois chaînes comme valeurs. Pour configurer le widget pour accepter une gamme de valeurs numériques, voir l'option *from_* ci-dessus.
width	Utilisez cette option pour spécifier le nombre de caractères permis dans la partie d'entrée du widget. La valeur par défaut est 20.
wrap	Normalement, quand le widget est à sa valeur la plus haute, la flèche qui pointe vers le haut ne fait rien et quand le widget est à sa valeur la plus basse, la flèche qui pointe vers le bas ne fait rien. Si vous choisissez wrap=True, la flèche qui pointe vers le haut s'avancera de la valeur la plus haute à la valeur la plus basse et la flèche qui pointe vers le bas s'avancera de la valeur la plus basse à la valeur la plus haute.
xscrollcommand	Utilisez cette option pour connecter un scrollbar à la partie d'entrée du widget. Pour plus de détails, voir section (p.).

Ces méthodes sont disponibles sur les widgets Spinbox:

.bbox (*index*)

Cette méthode rend la bounding box (boîte de limitation) du caractère positionné à *index* de dans la partie d'entrée du widget. Le résultat est un tuple (x, y, w, h), où les valeurs sont les coordonnées x et y à partir du coin supérieur gauche et la largeur et la hauteur du caractère en pixels, dans cet ordre.

.delete (first, last=None)

Cette méthode supprime les caractères de la partie d'entrée du Spinbox. Les valeurs de *first* et *last* sont interprétées de la façon standard pour des intervals Python.

.get()

Cette méthode rend la valeur du Spinbox. La valeur est toujours rendue en chaîne, même si le widget est paramétrer pour contenir un nombre.

.icursor (*index*)

Utilisez cette méthode de placer le curseur d'insertion à l'emplacement indiqué par *index*, utilisant la convention standard de Python concernant les positions.

.identify (*x* , *y*)

Etant donné la position (x, y) dans le widget, cette méthode rend une chaîne décrivant ce qui est à cet emplacement. Les valeurs peuvent être:

· 'entry' pour le secteur d'entrée.

· 'buttonup' pour la pointe de flèche indiquant vers le haut.

· 'buttondown' pour la pointe de flèche indiquant de haut en bas.

· ''(une chaîne vide) si ces coordonnées ne sont pas dans le widget.

.index (*i*)

Cette méthode rend la position numérique d'un index *i*. Les arguments peuvent être:

· END pour obtenir la position après le dernier caractère de l'entrée.

· INSERT pour obtenir la position du curseur d'insertion.

· ANCHOR pour obtenir la position de l'ancre de sélection.

· 'sel.first' pour obtenir la position du début de la sélection. Si la sélection n'est pas dans le widget, cette méthode lève une exception TclError.

· 'sel.last' pour obtenir la position juste devant la fin de la sélection. Si la sélection n'est pas dans le widget, cette méthode lève une exception TclError.

· une chaîne de la forme "@x" dénote une coordonnée *x* dans le widget. La valeur de retour est la position du caractère contenant cette coordonnée. Si la coordonnée est à l'extérieur du widget, la valeur de retour sera la position du caractère le plus proche à cette position.

.insert (*index* , *text*)

Cette méthode insère les caractères de la chaîne *text* à la position indiquée par *index*. Pour les valeurs d'index possibles, voir la méthode .index () ci-dessus.

.invoke (*element*)

Appelez cette méthode pour obtenir le même effet que l'utilisateur cliquant sur une pointe de flèche. L'argument d'élément est 'buttonup' pour la flèche qui pointe vers le haut et 'buttondown' pour la flèche qui pointe vers le bas.

.scan_dragto (*x*)

Cette méthode travaille comme la méthode .scan_dragto () décrite section (p.).

.scan_mark (*x*)

Cette méthode travaille comme la méthode .scan_mark () décrite section (p.).

.selection('from', *index*)

Met la sélection de l'ancre dans le widget à la position indiquée par *index*. Pour connaître les valeurs possibles d'*index*, voir la méthode .index () ci-dessus. La valeur initiale de la sélection de l'ancre est 0.

.selection('to', *index*)

Sélectionne le texte entre la sélection de l'ancre et l'*index* donné.

.selection('range', *start* , *end*)

Sélectionne le texte entre les indices *start* et *end*. Pour connaître les valeurs d'index permises, voir la méthode .index () ci-dessus.

.selection_clear()

Vide la sélection.

.selection_get()

Rend le texte sélectionné. S'il n'y a actuellement aucune sélection, cette méthode lèvera une exception TclError.

23 - Le widget Texte

Le widget texte est une méthode beaucoup plus généralisée pour manipuler les lignes multiples de texte que le widget Etiquette. Les widgets texte sont à peu près un éditeur de texte complet dans une fenêtre :

 Vous pouvez mélanger du texte avec des polices de caractères, des couleurs et des contextes différents.

 Vous pouvez incorporer diverses images et du texte. Une image est traitée comme un caractère seul. Voir section (p.).

 Un *index* est une façon de décrire une position spécifique entre deux caractères d'un widget texte. Voir section (p.).

 Un widget texte peut contenir des objets *mark* invisibles entre les positions de caractère. Voir section (p.).

 Les widgets texte vous permettent de définir des noms pour les régions du texte appelé des étiquettes. Vous pouvez changer l'apparence d'une région étiquetée, changeant sa police de caractères, ses couleurs de premier plan et de fond et d'autres attributs. Voir section (p.).

 Vous pouvez lier des événements à une région étiquetée. Voir section (p.).

 Vous pouvez même insérer un widget texte dans "une fenêtre" contenant n'importe quel widget, même un widget cadre contenant d'autres widgets. Une fenêtre est aussi traitée comme un caractère simple. Voir section (p.).

Pour créer un widget texte dans une fenêtre ou un cadre nommé parent :

```
w = Text ( parent, option, ... )
```

Le constructeur renvoi le nouveau widget texte. Les options disponibles:

autoseparators	Si l'option undo est mise, les commandes d'option d'autoséparateurs (si séparateurs) sont automatiquement ajoutés à la pile undo après chaque insertion ou effacement (si autoseparators=True) ou pas (si autoseparators=False). Pour une vue

	d'ensemble du mécanisme undo, voir section (p.).
bg ou background	La couleur par défaut de fond du gadget(machin) de texte. Voir section (p.).
bd ou borderwidth	La largeur de la frontière autour du widget texte; voir la section (p.). Par défaut deux pixels.
cursor	Le curseur qui apparaîtra quand la souris est sur le gadget(machin) de texte. Voir section (p.).
exportselection	Normalement, le texte choisi dans un widget texte est exporté pour être la sélection dans le directeur de fenêtre. Positionner exportselection=0 si vous ne voulez pas de ce comportement.
font	La police de caractères par défaut pour texte inséré dans le widget. Notez que vous pouvez avoir plusieurs polices de caractères dans les widgets en utilisant des étiquettes pour changer les propriétés de certain texte. Voir section (p.).
fg ou foreground	La couleur utilisée pour le texte (et bitmaps) dans le widget. Vous pouvez changer la couleur pour des régions étiquetées; cette option est juste la couleur par défaut.
height	La hauteur du widget en lignes (pas pixels!), mesuré selon la taille de police de caractères actuelle.
highlightbackground	Couleur de focus quand le widget texte n'a pas de focus. Voir section (p.).
highlightcolor	Couleur de focus quand le widget a le focus.
highlightthickness	Intensité de la couleur de focus.Par défaut 1. Mettre highlightthickness = 0 pour supprimer l'affichage du focus.
insertbackground	La couleur du curseur d'insertion. Par défaut noir.
insertborderwidth	Taille de la frontière 3-D autour du curseur d'insertion. Par défaut 0.
insertofftime	Le nombre de millisecondes que le curseur d'insertion est éteint pendant son cycle pour clignoter. Positionner cette option à zéro pour supprimer le clignotement. Par défaut 300.
insertontime	Le nombre de millisecondes que le curseur d'insertion est allumé pendant son cycle pour clignoter. Par défaut 600.
insertwidth	La largeur du curseur d'insertion (sa hauteur est fonction de l'article le plus grand de sa ligne). Par défaut 2 pixels.
maxundo	Cette option met le nombre maximum d'opérations conservées sur la pile undo. Pour une vue d'ensemble du mécanisme undo, voir section (p.). Mettre cette option

	à-1 pour spécifier un nombre illimité d'entrées dans la pile undo.
padx	Remplissage supplémentaire gauche et droite du secteur de texte. Par défaut 1 pixel. Voir section (p.) pour les valeurs possibles.
pady	Remplissage supplémentaire dessus et dessous le secteur de texte. Par défaut 1 pixel.
relief	L'apparence 3-D du widget texte. Par défaut relief=SUNKEN; pour les autres valeurs, voir section (p.).
selectbackground	La couleur de fond utilisée pour l'affichage le texte sélectionné.
selectborderwidth	La largeur de la frontière utilisée autour du texte sélectionné.
selectforeground	La couleur de premier plan utilisée pour l'affichage le texte sélectionné.
spacing1	Cette option spécifie combien d'espace vertical supplémentaire est mis à chaque ligne de texte. Si une ligne est enveloppée, cet espace est ajouté seulement avant la première ligne occupée sur l'affichage. Par défaut 0.
spacing2	Cette option spécifie combien d'espace vertical supplémentaire ajouter entre les lignes de texte affichées quand une ligne logique enveloppe. Par défaut 0.
spacing3	Cette option spécifie combien d'espace vertical supplémentaire est ajouté au-dessous de chaque ligne de texte. Si une ligne enveloppe, cet espace est ajouté seulement après la dernière ligne occupée sur l'affichage. Par défaut 0.
state	Normalement, les widgets texte répondent aux événements de souris et au clavier; mettre state=NORMAL pour obtenir ce comportement. Si vous mettez state=DISABLED, le widget texte ne répondra pas et vous ne serez pas capables de modifier son contenu par programmation non plus.
tabs	Cette option contrôle comment sont positionnés les caractères du texte d'étiquette. Voir section (p.).
takefocus	Normalement, le contrôle de clavier agit sur les widgets texte (voir section (p.)). Positionner takefocus=0 si vous voulez empêcher le contrôle de clavier du widget.
undo	Positionner cette option à *True* pour permettre le mécanisme undo, ou *False* pour le mettre hors de service. Voir section (p.).
width	La largeur du widget en caractères (pas pixels!), mesuré selon la taille de police de caractères courante.
wrap	Cette option contrôle l'affichage des lignes qui sont trop larges.

	· Avec le comportement par défaut, wrap=CHAR, n'importe quelle ligne qui devient trop longue sera pliée à n'importe quel caractère. · Paramétrer wrap=WORD et il pliera la ligne après le dernier mot entier. · Si vous voulez être capables de créer des lignes trop longues pour aller dans la fenêtre, mettre wrap=NONE et fournir un scrollbar horizontal.
xscrollcommand	Pour faire le widget texte horizontalement déroulant, mettez cette option à la méthode .set de scrollbar horizontal.
yscrollcommand	Pour faire le widget texte verticalement déroulant, mettez cette option à la méthode .set de scrollbar vertical.

23-1 - Index de widget texte

Un index est une méthode générale pour spécifier une position dans le contenu d'un widget texte. Un index est une chaîne avec une de ces formes :

" *line* . *column* "

La position juste avant la colonne donnée (comptant de zéro) sur la ligne donnée (comptant d'un). Exemples : "1.0" est la position du début du texte; "2.3" est la position avant le quatrième caractère de la deuxième ligne.

" *line* .end"

La position juste avant le retour à la ligne à la fin de la ligne donnée (comptant d'un). Ainsi, par exemple, l'index "10.end" est la position à la fin de la dixième ligne.

INSERT

La position du curseur d'insertion dans le widget texte. Cette constante est égal à la chaîne "insert".

CURRENT

La position du caractère le plus proche à l'indicateur de souris. Cette constante est égal à la chaîne "current".

END

La position après le dernier caractère du texte. Cette constante est égal à la chaîne "end"

SEL_FIRST

Si une partie de texte dans le widget est actuellement sélectionné (comme en y traînant la souris), c'est la position avant le début de la sélection. Si vous essayez d'utiliser cet index et que rien n'est sélectionné, une exception TclError sera levée. Cette constante est égal à la chaîne "sel.first".

SEL_LAST

La position après la fin de la sélection, si existe. Comme avec SEL_FIRST, vous obtiendrez une exception TclError si vous utilisez un tel index et qu'il n'y a aucune sélection. Cette constante est égal à la chaîne "sel.last".

" *markname* "

Vous pouvez utiliser une marque comme index; passez juste son nom où on s'attend à un index. Voir section (p.).

" *tag* .first"

La position avant le premier caractère de la région étiquetée avec le nom ***tag*** ; voir section (p.).

" *tag* .last"

La position après le dernier caractère d'une région étiquetée.

"@ *x* , *y* "

La position avant le caractère le plus proche à la coordonnée (x, y).

embedded-object

Si vous avez une image ou une fenêtre incorporée dans le widget texte, vous pouvez utiliser la Photoimage, BitmapImage, ou le widget incorporé comme un index. Voir section (p.) et section (p.).

En plus des options d'index de base ci-dessus, vous pouvez construire des expressions complexes arbitraires en ajoutant n'importe lequel de ces suffixes à un index de base ou indexer l'expression:

+ *n* chars

Depuis l'index donné, avancez de *n* caractères. Cette opération traversera les lignes frontières.

Par exemple, supposez que la première ligne ressemble à ceci :

```
abcdef
```

L'expression d'index "1.0 + 5 chars" se réfère à la position entre e et f. Vous pouvez omettre les blancs et abréger des mots-clés dans ces expressions si le résultat est sans équivoque. Cet exemple pourrait être abrégé "1.0+5c".

- *n* chars

Semblable à la forme précédente, mais la position déplace en arrière de *n* caractères.

+ *n* lines

Déplace de *n* lignes après l'index donné. Tkinter essaye de laisser la nouvelle position dans la même colonne qu'il était sur la ligne d'où il est parti, mais si la ligne à la nouvelle position est plus courte, la nouvelle position se fera à la fin de la ligne.

- *n* lines

Déplace de *n* lignes avant l'index donné.

linestart

Se déplace à la position avant le premier caractère de l'index donné. Par exemple, la position "current linestart" se réfère au début de la ligne la plus proche à l'indicateur de souris.

lineend

Se déplace à la position après le dernier caractère de l'index donné. Par exemple, la position "sel.last lineend" se réfère à la fin de la ligne contenant la fin de la sélection actuelle.

wordstart

La position avant le début du mot contenant l'index donné. Par exemple, l'index "11.44 wordstart" se réfère à la position avant le mot contenant la position 44 sur la ligne 11.

Pour cette opération, un mot est une chaîne de lettres, chiffres, ou underbar (_) consécutifs, ou un caractère seul qui n'est aucun de ces types.

23-2 - Mark de widget texte

Une *mark* représente une position flottante quelque part dans le contenu d'un widget texte.

 Vous manipulez chaque *mark* en lui donnant un nom. Ce nom peut être n'importe quelle chaîne qui inclut ni whitespace (caractère espace), ni période.

 Il y a deux *marks* spéciales. INSERT est la position actuelle du curseur d'insertion et le CURRENT est la position la plus proche du curseur de souris.

 Les *marks* flottent avec le contenu adjacent. Si vous modifiez le texte quelque part loin d'une *mark*, la *mark* reste à la même position par rapport à ses voisins immédiats.

 Les *marks* ont une propriété appelée *gravity* (gravité) qui contrôle ce qui arrive quand vous insérez le texte à une *mark*. Par défaut *gravity* est RIGHT, ce qui signifie que quand le nouveau texte est inséré à cette *mark*, la *mark* reste après la fin du nouveau texte. Si vous mettez *gravity* d'une *mark* à LEFT (utilisez la méthode du widget texte .mark_gravity ()), la *mark* restera à une position juste avant le texte nouvellement inséré à cette *mark*.

 Supprimer le texte tout autour d'une *mark* n'enlèvent pas la *mark*. Si vous voulez enlever une *mark*, utilisez la méthode .mark_unset () sur le widget texte.

Se référer à la section (p.), ci-dessous, pour voir comment utiliser les *marks*.

23-3 - Images de widget texte

Vous pouvez mettre une image ou un bitmap dans un widget texte. Il est traité comme un caractère seul dont la taille est la grandeur nature de l'objet. Voir section (p.) et section (p.).

Les images sont placées dans le widget texte en appelant la méthode .image_create() du widget. Voir ci-dessous pour l'ordre appelant et autres méthodes pour la manipulation d'image.

Les images sont manipulées en passant leur nom aux méthodes sur le widget texte. Vous pouvez donner un nom à Tkinter pour une image, ou vous pouvez juste laisser Tkinter produire un nom par défaut pour cette image.

Une image peut apparaître n'importe quel nombre de fois dans le même widget Texte. Chaque cas portera un nom unique. Ce nom peut être utilisé comme index.

23-4 - Fenêtres de widget texte

Vous pouvez mettre n'importe quel widget Tkinter même un cadre contenant d'autres widgets dans un widget texte. Par exemple, vous pouvez mettre un bouton entièrement fonctionnel ou un jeu de radiobuttons dans un widget texte.

Utilisez la méthode .window_create() sur le widget texte pour ajouter le widget incorporé. Pour l'ordre appelant et les méthodes liées, voir section (p.).

23-5 - Etiquettes de widget texte

Il y a tas de façons de changer tant l'apparence que la fonctionnalité des articles dans un widget texte. Pour le texte, vous pouvez changer la police de caractères, la taille et la couleur. Aussi, vous pouvez faire répondent des actions de souris ou du clavier le texte, les widgets, ou les images incorporées.

Pour contrôler cette apparence et ces caractéristiques fonctionnelles, vous associez chaque caractéristique avec une étiquette. Vous pouvez alors associer une étiquette avec n'importe quel nombre de pièces de texte dans le widget.

 Le nom d'une étiquette peut être n'importe quelle chaîne qui inclut ni whitespace (caractère espace), ni période.

 Il y a une étiquette spéciale prédéterminée appelée SEL. C'est la région actuellement sélectionnée, si existe.

 Puisque n'importe quel caractère peut faire partie de plus d'une étiquette, il y a une *tag stack* (pile d'étiquette) qui ordonne toutes les étiquettes. Les entrées sont ajoutées à la fin de la liste d'étiquette et les entrées postérieures ont la priorité sur les entrées précédentes.

Ainsi, par exemple, s'il y a un caractère *c* qui fait partie deux régions étiquetées *t1* et *t2*, et *t1* est plus profond dans la pile d'étiquette que *t2*, et *t1* veut que le texte soit vert et *t2* veut qu'il soit bleu, *c* sera rendu bleu parce que *t2* est précédant à *t1*.

 Les étiquettes sont créées en utilisant la méthode .tag_add() sur le widget texte. Voir section (p.), ci-dessous, pour informations et méthodes liées.

23-6 - Positionnement des étiquettes de widget texte

L'option d'étiquettes pour des widgets texte vous donne un certain nombre de possibilités de mettre des étiquette d'arrêt (tabulations) dans le widget.

 Par défaut place des étiquettes tous les huit caractères.

 Pour mettre des arrêts d'étiquette spécifiques, mettre cette option à un ordre d'une ou plusieurs distances. Par exemple, tabs=("3c", "5c", "12c") mettrait des étiquettes d'arrêt à 3, 5 et 12cm du côté gauche. Après la dernière étiquette vous obtenez des étiquettes avec la même largeur que la distance entre les deux derniers arrêts d'étiquette existants. Ainsi, continuant notre exemple, parce que 12c-5c fait 7 cm, si l'utilisateur continue à appuyer la Touche de tabulation, le curseur se déplacera à 19cm, 26cm, 33cm, et cetera.

 Normalement, le texte après un caractère d'étiquette est aligné sur son bord gauche sur l'arrêt d'étiquette, mais vous pouvez inclure n'importe lequel des mots-clés LEFT, RIGHT , CENTER, ou NUMERIC dans la liste après une distance et cela changera le positionnement du texte après chaque étiquette.

 Un arrêt d'étiquette LEFT (gauche) est le comportement par défaut.

 Un arrêt d'étiquette RIGHT placera le bord droit du texte sur l'arrêt.

 Une étiquette CENTER (centré) centrera le texte suivant sur l'arrêt d'étiquette.

 Un arrêt d'étiquette NUMERIC placera le texte suivant à gauche de l'arrêt jusqu'à la première période (".") dans le texte après quoi, centrera la période sur l'arrêt et le reste du texte sera placé à sa droite.

Par exemple, tabs=("0.5i", "0.8i", RIGHT, "1.2i", CENTER, "2i", NUMERIC) mettrait quatre étiquettes d'arrêt : un étiquette d'arrêt aligné-gauche à la moitié d'un pouce du côté gauche, une étiquette d'arrêt alignée-droite à 0.8" du côté gauche, une étiquette d'arrêt centrée à 1.2" à gauche et une étiquette d'arrêt numérique à 2" à gauche.

23-7 - Pile undo/redo de widget texte

Le widget texte a un mécanisme incorporé qui vous permet de mettre en oeuvre des opérations undo (défaire) et redo (refaire) qui peuvent annuler ou rétablir des changements au texte dans le widget.

Voici comment la pile undo/redo travaille :

 Chaque changement au contenu est enregistré en poussant des entrées sur la pile qui décrivent le changement, si insertion ou effacement. Ces entrées enregistrent l'ancien état du contenu aussi bien que le nouvel état : si effacement, le texte supprimé est enregistré; si insertion, le texte inséré est enregistré, avec une description de l'emplacement ainsi que si c'était une insertion ou un effacement.

 Votre programme peut aussi pousser un enregsitrement spécial appelé *separator* (séparateur) sur la pile.

 Une opération undo (défaire) change le contenu du widget à ce qu'ils étaient à un certain point précédent. Il le fait en changeant complètement tous les changements poussés sur la pile undo/redo jusqu'à ce qu'il atteigne un *separator* (séparateur) ou jusqu'à ce qu'il soit à la fin de la pile.

Cependant, notez que Tkinter se souvient aussi de la totalité de la pile qui a été complètement changé dans l'opération de undo, jusqu'à ce qu'une autre opération de rédaction change le contenu du widget.

 Une opération redo (refaire) travaille seulement si aucune opération de rédaction n'est arrivée depuis la dernière opération de undo. Il s'incrémente à nouveau de toutes les opérations de undo.

Pour les méthodes utilisées pour mettre en oeuvre la pile undo/redo, voir les méthodes .edit_redo, .edit_reset, .edit_separator et .edit_undo dans la section (p.). Le mécanisme undo/redo n'est pas positionné par défaut; vous devez mettre l'option undo dans le widget.

23-8 - Méthodes de widget texte

Ces méthodes sont disponibles sur tous les widgets texte :

.bbox (*index*)

Rend la boîte de limitation (bounding box) pour le caractère à l'index donné, un 4-tuple (*x, y, width, height*). Si le caractère n'est pas visible, rend None. Notez que cette méthode ne peut pas rendre une valeur précise à moins que vous n'appelliez la méthode .update_idletasks() (voir section (p.)).

.compare (*index1* , *op* , *index2*)

Compare les positions de deux indices dans le widget texte et retourne *true* si l'opérateur *op* se tient entre *index1* et *index2*. L'*op* spécifie quelle comparaison utiliser, un de : "<", "<=", "==", "!=", ">=", ou ">".

Par exemple, pour un widget texte *t*, t.compare("2.0", "< =", END) renvoi *true* si le début de la deuxième ligne est avant ou à la fin du texte dans *t*.

.delete (*index1* , *index2* =None)

Supprime le texte commençant juste après *index1*. Si le deuxième argument est omis, seulement un caractère est supprimé. Si on donne un deuxième index, l'effacement est fait jusqu'à, mais sans inclure, le caractère après *index2*. Rappellez-vous que les indices sont positionnés entre les caractères.

.dlineinfo (*index*)

Rend une boîte de limitation (bounding box) pour la ligne qui contient l'*index* donné. Pour la forme de la boîte de limitation, et un avertissement de la mise à jour de tâches inoccupées, voir la définition de la méthode .bbox ci-dessus.

.edit_modified (arg=None)

Questionne, positionne, ou nettoie le *modified flag* (drapeau modifié). Ce drapeau est utilisé pour suivre à la trace si le contenu du widget a été changé. Par exemple, si vous mettez en oeuvre un éditeur de texte dans un widget texte, vous pourriez utiliser le *modified flag* pour déterminer si le contenu a changé depuis votre dernière sauvegarde du contenu d'un fichier.

Quand appelé sans argument, cette méthode retourne *True* si le *modified flag* a été mis, False sinon. Vous pouvez aussi explicitement mettre le drapeau modifié en passant une valeur *True* à cette méthode, ou le nettoyer en passant une valeur False.

N'importe quelle opération qui insère ou supprime le texte, que ce soit par des actions de programme ou des actions d'utilisateur, ou une opération undo ou redo, mettra le *modified flag*.

.edit_redo()

Exécute une opération redo. Pour voir les détails section (p.).

.edit_reset()

Nettoie la pile undo.

.edit_separator()

Pousse un séparateur sur la pile undo. Ce séparateur limite la portée d'une future opération undo pour inclure seulement les changements poussés puisque le séparateur a été poussé. Pour voir les détails section (p.).

.edit_undo()

Change complètement le contenu du widget fait depuis le dernier séparateur qui a été poussé sur la pile undo, ou entièrement depuis le bas de la pile si la pile ne contient aucun séparateur. Pour voir les détails section (p.). Renvoi une erreur si la pile undo est vide.

.image_create (*index* [, *option* = *value* , ...])

Cette méthode insère une image dans le widget. L'image est traitée comme un autre caractère, dont la taille est la grandeur naturel de l'image.

Les options pour cette méthode sont montrés dans la table ci-dessous. Vous pouvez passer une série d'argument option=valeur, ou un dictionnaire de noms et de valeurs d'option.

align	Cette option spécifie comment l'image doit être alignée verticalement si sa hauteur

	est inférieure à la hauteur de sa ligne contenante. Les valeurs peuvent être *top* pour l'aligner au sommet de son espace; *center* pour le centrer; *bottom* pour le placer en bas; ou *baseline* pour aligner le bas de l'image avec la ligne de base de texte.
image	L'image utilisée. Voir section (p.).
name	Vous pouvez assigner un nom à cette image. Si vous omettez cette option, Tkinter produira un nom unique. Si vous créez plusieurs copies d'une image dans le même widget texte, Tkinter produira un nom unique en ajoutant un #suivi par un numéro.
padx	Si fourni, cette option correspond à un certain nombre de pixels d'espace supplémentaire à ajouter des deux côtés de l'image.
pady	Si fourni, cette option correspond à un certain nombre de pixels d'espace supplémentaire à ajouter au dessus et au dessous de l'image.

.get (*index1* , *index2* =None)

Utilisez cette méthode de récupérer le texte actuel du widget. La récupération commence à l'index *index1*. Si le deuxième argument est omis, vous obtenez le caractère après *index1*. Si vous fournissez un deuxième index, vous obtenez le texte entre ces deux indices. Les images et fenêtres (widgets) incorporées sont ignorés.

.image_cget (*index* , *option*)

Pour récupérer la valeur courante d'un jeu d'option sur une image incorporée, appelez cette méthode avec un index indiquant l'image et le nom de l'option.

.image_configure (*index* , *option* , ...)

Pour mettre une ou plusieurs options à une image incorporée, appelez cette méthode avec un index indiquant l'image comme premier argument et une ou plusieurs paires d'option=value.

Si vous ne spécifiez aucune option, vous récupérerez un dictionnaire définissant toutes les options sur l'image et les valeurs correspondantes.

.image_names()

Cette méthode rend un tuple des noms des images incorporées dans tout le widget texte.

.index (*i*)

Pour un index *i*, cette méthode rends la position équivalente sous forme "line.char".

.insert (*index* , *text* , *tags* =None)

Insère le texte donné à l'index donné.

Si vous omettez l'argument *tags* (étiquettes) , le texte nouvellement inséré sera étiqueté avec n'importe quelles étiquettes qui s'appliquent aux caractères autant avant qu'après le point d'insertion.

Si vous voulez appliquer une ou plusieurs étiquettes au texte que vous insérez, fournissez comme troisième argument un tuple de chaîne d'étiquette. N'importe quelles étiquettes qui s'appliquent aux caractères existants autour du point d'insertion sont ignorées.

Note : le troisième argument doit être un tuple. Si vous fournissez un argument liste, Tkinter échouera silencieusement pour appliquer les étiquettes. Si vous fournissez une chaîne, chaque caractère sera traité comme une étiquette.

.mark_gravity (*mark* , *gravity* =None)

Change ou fournie la gravité d'une mark existante; voir section (p.)., ci-dessus, pour une explication sur la gravité.

Pour positionner la gravité, passez au nom de la mark, suivie par LEFT ou RIGHT. Pour connaître la gravité d'une mark existante, omettez le deuxième argument et la méthode retourne LEFT ou RIGHT.

.mark_names()

Rend une séquence des noms de toutes les marks dans la fenêtre, y compris INSERT et CURRENT.

.mark_next (*index*)

Rend le nom de la mark après l'index donné; s'il n'y en a aucune, la méthode rend une chaîne vide.

Si l'index est de forme numérique, la méthode rend la première mark à cette position. Si l'index est une mark, la méthode rend la mark suivante après cette mark, qui peut être à la même position numérique.

.mark_previous (*index*)

Rend le nom de la mark précédant l'index donné. S'il n'y a aucune mark précédente, la méthode rend une chaîne vide.

Si l'index est de forme numérique, la méthode rend la dernière mark à cette position. Si l'index est une mark, la méthode rend la mark précédente, qui peut être à la même position numérique.

.mark_set (*mark* , *index*)

Si aucune mark avec pour nom *mark* n'existe, elle est créé avec la gravité RIGHT et placé où l'index indique. Si la mark existe déjà, elle est déplacée au nouvel emplacement.

Cette méthode peut changer la position des indices INSERT ou CURRENT.

.mark_unset (*mark*)

Enlève la mark nommée. Cette méthode ne peut pas être utilisée pour enlever les marks INSERT ou CURRENT.

.scan_dragto (*x* , *y*)

Voir .scan_mark, ci-dessous.

.scan_mark (*x* , *y*)

Cette méthode est utilisée pour mettre en oeuvre le scrolling rapide d'un widget Texte. Typiquement un utilisateur presse et tient un bouton de souris depuis une certaine position dans le widget et ensuite déplace la souris dans la direction voulue ; la souris se déplace dans le widget dans cette direction à un taux proportionnel à la distance des que le bouton a été relaché. Le mouvement peut être n'importe quelle combinaison de scrolling vertical ou horizontal.

Pour mettre en oeuvre cette caractéristique, liez l'événement en bas d'un bouton de souris à un entraîneur qui appelle .scan_mark (x, y), où x et y sont la position de souris actuelle. Liez alors l'événement <Motion> à un entraîneur qui appelle .scan_dragto (x, y), où x et y sont la nouvelle position de souris.

.search (*pattern* , *index* , *option* , ...)

Recherche du *pattern* (modèle) (qui peut être chaîne ou une expression régulière) dans le buffer commençant à l'index donné. S'il réussit, il rend un index de la forme "*line.char*"; s'il échoue, il rend une chaîne vide.

Les options permises pour cette méthode sont :

backwards	Mettre cette option à True pour chercher en arrière de l'index. Par défaut est en avant.
count	Si vous mettez cette option à une variable de contrôle IntVar, quand il y a un match vous pouvez récupérer la longueur du texte qui correspond en utilisant la méthode .get () sur cette variable après le renvoi de la méthode.
exact	Mettre cette option à True pour chercher le texte qui correspond exactement au *pattern*. C'est l'option par défaut. Comparez l'option regexp ci-dessous.
forwards	Mettre cette option à True pour chercher en avant de l'index. C'est l'option par défaut.
regexp	Mettre cette option à True pour interpréter le modèle comme une expression régulière Tcl-style. Par défaut doit chercher le model exact de *pattern*. Les expressions régulières Tcl sont des sous-ensemble d'expressions régulières de Python, supportant ces caractéristiques:. ^ [c1 □] (□) * + ? E1|e2
nocase	Mettre cette option à 1 pour ignorer la casse. Par défaut la recherche est sensible à la casse.
stopindex	Pour limiter la recherche, mettre cette option à l'index au-delà duquel la recherche ne devrait pas aller.

.see (*index*)

Si le texte contenant l'index donné n'est pas visible, défile le texte jusqu'à ce que ce texte devienne visible.

.tag_add (*tagName* , *index1* , *index2* =None)

Cette méthode associe l'étiquette nommée *tagName* avec une région du contenu commençant juste après l'index *index1* et se prolongeant jusqu'à l'index *index2*. Si vous omettez *index2*, seulement le caractère après index1 est étiqueté.

.tag_bind (*tagName* , *sequence* , *func* , *add* =None)

Cette méthode lie un événement à tout le texte étiqueté avec tagName. Voir section (p.), ci-dessous, pour plus d'informations sur les événements d'attache.

Pour créer une nouvelle attache pour le texte étiqueté, utilisez les trois premiers arguments : *sequence* identifie l'événement, et *func* est la fonction que vous voulez appeller quand cet événement arrive.

Pour ajouter une autre attache à une étiquette existante, passez les trois mêmes premiers arguments et "+" comme le quatrième argument.

Pour rechercher les attaches qui existent pour une *sequence* donnée sur une étiquette, passer seulement les deux premiers arguments; La méthode rend la fonction associée.

Pour trouver toutes les attaches pour une étiquette donnée, passer seulement le premier argument; la méthode rend une liste de tous les arguments de *sequence* de l'étiquette.

.tag_cget (*tagName* , *option*)

Utilisez cette méthode de récupérer la valeur de l'*option* donnée pour le *tagName* donné.

.tag_config (*tagName* , *option* , ...)

Pour changer la valeur d'options de l'étiquette nommée *tagName*, passez une ou plusieurs paires d'*option=value*.

Si vous passez seulement un argument, vous récupérerez un dictionnaire définissant toutes les options et leurs valeurs actuelles pour l'étiquette nommée.

Voici les options pour la configuration d'étiquette :

background	La couleur de fond pour texte avec cette étiquette. Notez que vous ne pouvez pas utiliser bg comme abréviation.
bgstipple	Pour rendre le contexte grisâtre, mettre cette option à un des noms de bitmap standard (voir section (p.)). Cela n'a aucun effet si vous ne spécifiiez pas aussi un *background*.
borderwidth	La largeur de la frontière autour du texte de cette étiquette. Par défaut 0. Notez que vous ne pouvez pas utiliser bd comme abréviation.
fgstipple	Pour rendre le texte grisâtre, mettre cette option à un nom de bitmap.
font	La police de caractères utilisée pour afficher le texte de cette étiquette. Voir section (p.).
foreground	La couleur utilisée pour le texte de cette étiquette. Notez que vous ne pouvez pas utiliser l'abréviation fg.
justify	L'option de justification positionné sur le premier caractère de chaque ligne détermine comment cette ligne est justifiée : LEFT (par défaut), CENTER, ou RIGHT.
lmargin1	De combien indenter la première ligne d'un gros morceau de texte qui a cette étiquette. Par défaut 0. Voir section (p.) pour les valeurs permises.
lmargin2	De combien indenter les lignes successives d'un gros morceau de texte qui a cette étiquette. Par défaut 0.
offset	De combien augmenter (valeurs positives) ou diminuer (valeurs négatives) le texte de cette étiquette par rapport à la ligne des bases.

	Utilisez-le par exemple pour obtenir des superscripts ou des subscripts. Pour les valeurs permises, voir section (p.).
overstrike	Mettre overstrike=1 pour dessiner une ligne horizontale passant par le centre du texte de cette étiquette.
relief	Effet 3-D utilisé pour le texte de cette étiquette. Par défaut relief=FLAT; pour d'autres valeurs possibles voir section (p.).
rmargin	Taille de la marge droite pour les gros morceaux de texte de cette étiquette. Par défaut 0.
spacing1	Cette option spécifie combien d'espace vertical supplémentaire est mis au dessus de chaque ligne de texte de cette étiquette. Si une ligne enveloppe, cet espace est ajouté seulement avant la première ligne occupée sur l'affichage. Par défaut 0.
spacing2	Cette option spécifie combien d'espace vertical supplémentaire ajouter entre les lignes de texte affichées pour cette étiquette quand une ligne logique enveloppe. Par défaut 0.
spacing3	Cette option spécifie combien d'espace vertical supplémentaire est ajouté au-dessous de chaque ligne de texte de cette étiquette. Si une ligne enveloppe, cet espace est ajouté seulement après la dernière ligne occupée sur l'affichage. Par défaut 0.
tabs	Comment les étiquettes sont étendues sur les lignes de cette étiquette. Voir section (p.).
underline	Mettre underline=1 pour souligner le texte de cette étiquette.
wrap	Longueur des lignes enveloppées dans le texte de cette étiquette. Voir la description de l'option *wrap* pour des widgets texte, ci-dessus.

.tag_delete (*tagName* , ...)

Pour supprimer une ou plusieurs étiquettes, passez leurs noms à cette méthode. Leurs options et attaches partent et les étiquettes sont enlevées de toutes les régions de texte.

.tag_lower (*tagName* , *belowThis* =None)

Utilisez cette méthode pour changer l'ordre d'étiquettes dans la pile d'étiquette (voir section (p.), ci-dessus, pour une explication de la pile d'étiquette). Si vous passez deux arguments, l'étiquette avec le nom *tagName* est déplacée à une position juste au-dessous de l'étiquette avec le nom *belowThis*. Si vous passez seulement un argument, cette étiquette est déplacée au bas de la pile d'étiquette.

.tag_names (*index* =None)

Si vous passez un argument d'index, cette méthode rend un ordre de tous les noms d'étiquette qui sont associés au caractère après cet index. Si vous ne passez aucun argument, vous obtenez un ordre de tous les noms d'étiquette définis dans le widget texte.

.tag_nextrange (*tagName* , *index1* , *index2* =None)

Cette méthode recherche dans une région donnée les endroits où une étiquette nommée *tagName* commence. La région à fouiller commence à l'index *index1* et fini à l'index *index2*. Si l'argument *index2* est omis, la recherche va entièrement à la fin du texte.

S'il y a un endroit dans la région donnée où ces étiquette commencent, la méthode rend une *sequence* [i0, i1], où i0 est l'index du premier caractère étiqueté et i1 est l'index de la position juste après le dernier caractère étiqueté.

Si aucun démarrage d'étiquette n'est trouvé dans la région, la méthode rend une chaîne vide.

.tag_prevrange (*tagName* , *index1* , *index2* =None)

Cette méthode recherche dans une région donnée les endroits où une étiquette nommée *tagName* commence. La région à fouiller commence avant l'index *index1* et fini à l'index *index2*. Si l'argument index2 est omis, la recherche va entièrement à la fin du texte.

Les valeurs de retour sont comme dans .tag_nextrange().

.tag_raise (tagName, *aboveThis* =None)

Utilisez cette méthode pour changer l'ordre d'étiquettes dans la pile d'étiquette (voir section (p.), ci-dessus, pour une explication de la pile d'étiquette). Si vous passez deux arguments, l'étiquette avec le nom *tagName* est déplacée à une position juste au-dessus de l'étiquette avec le nom *aboveThis*. Si vous passez seulement un argument, cette étiquette est déplacée au sommet de la pile d'étiquette.

.tag_ranges (*tagName*)

Cette méthode trouve toutes les gammes de texte dans le widget qui sont étiquetées avec le nom *tagName* et rend un ordre [s0, e0, s1, e1, □], où chaque *si* est l'index juste avant le premier caractère de la gamme et *ei* est l'index juste après le dernier caractère de la gamme.

.tag_remove (*tagName* , *index1* , *index2* =None)

Enlève l'étiquette nommée *tagName* de tous les caractères entre *index1* et *index2*. Si *index2* est omis, l'étiquette est enlevée du caractère seul après *index1*

.tag_unbind (*tagName* , *sequence* , *funcid* =None)

Enlevez l'attache d'événement pour la *sequence* donnée de l'étiquette nommée *tagName*. S'il y a de multiples attaches pour cette *sequence* et étiquette, vous pouvez enlever seulement une attache en le passant comme le troisième argument.

.window_cget (*index* , *option*)

Rend la valeur de l'option donnée pour le widget incorporé à l'index donné.

.window_configure (*index* , *option*)

Pour changer la valeur des options pour le widget incorporé à l'index donné, passez une ou plusieurs paires de valeur *option=value*.

Si vous passez seulement un argument, vous récupérerez un dictionnaire définissant toutes les options et leurs valeurs actuelles pour le widget donné.

.window_create (*index* , *option* , ...)

Cette méthode crée une fenêtre où un widget peut être incorporé dans un widget texte. Il y a deux façons de fournir le widget incorporé :

a Vous pouvez passer le widget à l'option de fenêtre dans cette méthode, ou
b Vous pouvez définir une procédure qui créera le widget et passer la procédure comme un rappel de service à l'option se créant.

Les options pour .window_create () sont :

align	Spécifie comment placer le widget incorporé verticalement à sa ligne, s'il n'est pas aussi grand que le texte sur la ligne. Les valeurs incluent : align=CENTER (par défaut), qui centre le widget verticalement dans la ligne; align=TOP, qui place le sommet de l'image au sommet de la ligne; align=BOTTOM, qui place le bas de l'image au bas de la ligne; et align=BASELINE, qui aligne le bas de l'image avec la ligne des bases de texte.
create	Une procédure qui créera le widget incorporé sur demande. Cette procédure ne prend aucun argument et doit créer le widget comme un enfant du widget texte et rendre le widget comme son résultat.
padx	Espace supplémentaire ajouté a gauche et droite du widget dans la ligne de texte. Par défaut 0.
pady	Espace supplémentaire au dessus et au dessous du widget dans la ligne de texte. Par défaut 0.
stretch	Cette option contrôle ce qui arrive quand la ligne est plus haute que le widget incorporé. Normalement cette option est à 0, signifiant que le widget incorporé est laissé à sa grandeur naturelle. Si vous mettez stretch=1, le widget est tendu verticalement pour remplir la hauteur de la ligne et l'option *align* est ignorée.
window	Le widget à être incorporé. Ce widget doit être un enfant du widget texte.

.window_names()

Rend une séquence contenant les noms de tous les widgets incorporés.

.xview (MOVETO, *fraction*)

Cette méthode fait défiler le widget texte horizontalement et est destinée à l'option de commande de scrollbar horizontal lié.

Cette méthode peut être appelée de deux façons différentes. La première place le texte à une position donnée par *fraction*, où 0.0 déplace le texte à sa position extrême gauche et 1.0 à sa position extrême droite.

.xview (SCROLL, *n* , *what*)

La deuxième déplace le texte à gauche ou à droite : l'argument *what* spécifie de combien déplacer et peut être UNITS ou PAGES et *n* dit de combien de caractères ou de pages déplacer le texte à droite par rapport à son image (ou gauche, si négatif).

.xview_moveto (*fraction*)

Cette méthode fait défiler le texte de la même façon que .xview (MOVETO, fraction).

.xview_scroll (*n* , *what*)

Identique à .xview (SCROLL, n, what).

.yview(MOVETO, *fraction*)

Le scrolling vertical équivalent à .xview (MOVETO, □).

.yview(SCROLL, *n* , *what*)

Le scrolling vertical équivalent à .xview (SCROLL, □). Quand le scrolling vertical se fait par UNITS, les unités sont des lignes.

.yview_moveto(*fraction*)

Le scrolling vertical équivalent à .xview_moveto ().

.yview_scroll(*n* , *what*)

Le scrolling vertical équivalent à .xview_scroll ().

24 - Au plus haut niveau : méthodes de fenêtre top-level

Une fenêtre top-level est une fenêtre qui a une existence indépendante sous le directeur de fenêtre. Elle a les attributs du directeur de fenêtre et peut être déplacée et redimensionnée indépendamment. Votre application peut utiliser n'importe quel nombre de fenêtres top-level.

Pour n'importe quel widget w, vous pouvez arriver à son widget au plus haut niveau en utilisant w.winfo_toplevel ().

Pour créer une nouvelle fenêtre top-level :

```
w = Toplevel ( option, ... )
```

Les options sont :

bg ou background	La couleur de fond de la fenêtre. Voir section (p.).
bd ou borderwidth	Largeur de frontière en pixels; par défaut 0. Pour les valeurs possibles, voir section (p.). Voir aussi l'option *relief*, ci-dessous.
class_	Vous pouvez donner un nom de "classe" à une fenêtre top-level. De tels noms ont une correspondance avec les options de base de données, donc votre application peut prendre les préférences de configuration

	de l'utilisateur (tel que couleur) par le nom de classe. Par exemple, vous pourriez concevoir une série de menus contextuel appelés "screamers" et les positionner tous ensemble avec class _ = "Screamer". Alors vous pouvez mettre une ligne dans votre base de données d'option comme cela : `*Screamer*background: red` Et ensuite, si vous utilisez la méthode .option_readfile () pour lire votre base de données d'option, tous les widgets avec ce nom de classe auront une couleur de fond rouge. Cette option est nommée *class_* parce que *class* est un mot réservé dans le Python.
cursor	Le curseur utilisé quand la souris est dans la fenêtre; voir section (p.).
height	Hauteur de la fenêtre, voir section (p.).
highlightbackground	Couleur de focus quand la fenêtre n'a pas de focus. Voir section (p.).
highlightcolor	Couleur de focus quand la fenêtre a le focus.
highlightthickness	Intensité de la couleur de focus. Par défaut 1. Mettre highlightthickness=0 pour supprimer l'affichage du focus hightlight.
menu	Pour ajouter à cette fenêtre une barre de menu top-level, mettre un widget Menu comme la valeur de cette option. Sous MacOS, ce menu apparaîtra au sommet de l'écran quand la fenêtre est active. Sous WINDOWS ou UNIX, il apparaîtra au sommet de la l'application.
padx	Utilisez cette option pour fournir l'espace supplémentaire sur les côtés gauche et droit de la fenêtre. La valeur est en pixels.
pady	Utilisez cette option pour fournir l'espace supplémentaire en haut et en bas de la fenêtre. La valeur est en pixels.
relief	Normalement, une fenêtre top-level n'a pas de frontière 3D. Pour obtenir une frontière hachurée, mettre l'option *bd* à une valeur différente de zéro (par défaut) et mettre l'option de relief à une des constantes de la section (p.).
takefocus	Normalement, le focus ne fait pas de contrôle pour une fenêtre top-level; Mettre takefocus=True (vrai) Si vous voulez avoir un contrôle de focus ; voir section (p.).
width	La largeur voulue de la fenêtre; voir la section (p.).

Ces méthodes sont disponibles pour des fenêtres top-level :

.aspect (*nmin* , *dmin* , *nmax* , *dmax*)

Force la proportion largeur:hauteur de la fenêtre racine dans la gamme [nmin / dmin, nmax / dmax].

.deiconify()

Si cette fenêtre est iconifiée, l'étendre.

.geometry (*newGeometry* =None)

Positionner la géométrie de fenêtre. Pour la forme de l'argument, voir Geometry strings.

Si l'argument est omis, la chaîne de géométrie courante est rendue.

.iconify()

Iconifie la fenêtre.

.lift (*aboveThis* =None)

Pour élever cette fenêtre au sommet de l'ordre de pile dans le directeur de fenêtre, appelez cette méthode sans arguments. Vous pouvez aussi élever à une position dans l'ordre de pile juste au-dessus d'une autre fenêtre top-level en passant cette fenêtre un argument.

.lower (*belowThis* =None)

Si l'argument est omis, déplace la fenêtre au bas de l'ordre de pile dans le directeur de fenêtre. Vous pouvez aussi déplacer la fenêtre à une position juste sous une autre fenêtre top-level en passant le widget top-level comme un argument.

.maxsize (*width* =None, *height* =None)

Paramètre la taille de fenêtre maximale. Si les arguments sont omis, rend le courant (largeur, hauteur).

.minsize (*width* =None, *height* =None)

Paramètre la taille de fenêtre minimale. Si les arguments sont omis, rend les minimums actuels comme un 2-tuple.

.overrideredirect (flag=None)

Si appelé avec un argument *True*, cette méthode positionne l'ignoré du suivi de drapeau, qui enlève le directeur des décorations de la fenêtre, pour qu'elle ne puisse pas être déplacée, redimensionnée, iconifiée, ou fermée. Si appelé avec un argument *False*, le directeur des décorations de la fenêtre est rétabli et l'ignoré du suivi de drapeau est purifié. Si appelé sans argument, il retourne l'état actuel de l'ignoré du suivi de drapeau.

Assurez-vous d'appeler la méthode .update_idletasks () (voir section (p.)) avant de positionner ce drapeau. Si vous l'appelez avant l'entrée à la boucle principale, votre fenêtre sera mise hors de service avant qu'elle n'apparaisse.

Cette méthode peut ne pas marcher sur un certain UNIX et des plates-formes MacOS.

.resizable (*width* =None, *height* =None)

Si *width* est *true*, permet de redimensionner en horizontal. Si *height* est *true*, permet de redimensionner en vertical. Si les arguments sont omis, rend la taille actuelle comme un 2-tuple.

.state(newstate=None)

Rend l'état actuel de la fenêtre :

· "normal" : Affichée normalement.

· "iconic" : Iconifiée avec la méthode .iconify().

· "withdrawn" : Caché; voir la méthode .withdraw() ci-dessous.

Pour changer l'état de la fenêtre, passez une des chaînes ci-dessus comme argument à la méthode. Par exemple, pour iconifier une instance top-level *T*, utilisez "T.state("iconify")".

.title (*text* =None)

Met le titre de la fenêtre. Si l'argument est omis, rend le titre actuel.

.transient (*parent* =None)

Fait de cette fenêtre une fenêtre passagère pour une fenêtre parent; la fenêtre parent par défaut est le parent de cette fenêtre.

Cette méthode est utile pour des fenêtres de dialogue contextuelles de conservation limitée . Une fenêtre passagère apparaît toujours devant son parent. Si la fenêtre parent est iconifiée, la passagère est iconifiée aussi.

.withdraw()

Cache la fenêtre. Rétablissez-la avec .deiconify() ou .iconify().

25 - Méthodes universelles de widget

Les méthodes qui sont définies ci-dessous concernent tous les widgets. Dans les descriptions, *w* peut être n'importe quel widget de n'importe quel type.

***w* .after (*delay_ms* , callback=None, * *args*)**

Demande à Tkinter d'appeler le rappel de service de fonction avec les arguments *args* après un retard d'au moins *delay_ms* millisecondes. Il n'y a aucune limite supérieure à combien de temps cela prendra en réalité, mais votre rappel de service ne sera pas appelé plus tôt que vous l'avez demandé et il sera appelé seulement une fois.

Cette méthode rend un entier "after identifier" que l'on peut passer a la méthode .after_cancel() si vous voulez annuler le rappel de service.

Si vous ne passez pas d'argument de rappel de service, cette méthode attend delay_ms millisecondes, comme dans la fonction .sleep() du module de temps de Python standard (5) .

***w* .after_cancel (*id*)**

Abandonne une demande du rappel de service .after() fait plus tôt. L'argument *id* est le résultat rendu par l'appel original .after().

***w* .after_idle (*func* , * *args*)**

Demande a ce que Tkinter appel la fonction *func* avec les arguments *args* la prochaine fois que le système est inoccupé, c'est-à-dire la prochaine fois qu'il ni a aucun événement à être traité. Le rappel de service sera appelé

seulement une fois. Si vous voulez que votre rappel de service soit appelé de nouveau, vous devez appeler de nouveau la méthode .after_idle.

_w_ .bell()

Fait du bruit, habituellement un beep.

_w_ .bind (_sequence_ =None, _func_ =None, _add_ =None)

Cette méthode est utilisée pour lier une attache d'événement à un widget. Voir section (p.) pour une vue d'ensemble des attaches d'événement.

L'argument *sequence* décrit quel événement nous attendons et l'argument *func* est une fonction qui est appelée quand cet événement arrive au widget. S'il y avait déjà une attache pour cet événement pour ce widget, normalement le vieux rappel de service est remplacé par *func*, mais vous pouvez préserver les deux rappels de service par le passage du paramètre add = "+".

_w_ .bind_all (_sequence_ =None, _func_ =None, _add_ =None)

Comme .bind (), mais s'applique à tous les widget dans l'application entière.

_w_ .bind_class (_className_ , _sequence_ =None, _func_ =None, add=None)

Comme .bind (), mais s'applique à tous les widgets nommés *className* (par exemple, "Button").

_w_ .bindtags (_tagList_ =None)

Si vous appelez cette méthode, elle rendra les □binding tags□ (étiquettes obligatoires) pour le widget dans une séquence chaîne. Une étiquette obligatoire est le nom d'une fenêtre (commençant par ".") ou le nom d'une classe (par exemple, "Listbox").

Vous pouvez changer l'ordre dans lequel des niveaux obligatoires sont appelés en passant comme argument la séquence d'étiquettes obligatoires que vous voulez que le widget utilise.

Voir section (p.) pour une explication sur les niveaux de liens et leur relation aux étiquettes.

_w_ .cget (_option_)

Rend la valeur actuelle d'option dans une chaîne. Vous pouvez aussi obtenir la valeur d'une option pour le widget *w* de cette façon : w[option].

_w_ .clipboard_append (_text_)

Ajoute la chaîne de caractères donnée au presse-papiers de l'affichage, où les chaînes coupées et collées sont stockées pour toutes les applications de l'affichage.

_w_ .clipboard_clear()

Vide le presse-papiers de l'affichage (voir .clipboard_append () ci-dessus).

_w_ .column_configure()

Voir section (p.).

***w* .config(*option* = *value* , ...)**

Identique à .configure().

***w* .configure (*option* = *value* , ...)**

Met les valeurs d'une ou plusieurs options. Pour les options dont les noms sont des mots réservés par Python (*class*, *from*, *in*), utilisent un underbar : "class_", "from_", "in_".

Vous pouvez aussi mettre la valeur d'une option pour le widget *w* avec la déclaration

```
w[option]=value
```

Si vous appelez la méthode .config() sur un widget sans arguments, vous obtiendrez un dictionnaire des options actuelles de tout le widget. Les clés sont les noms d'option (incluant des pseudonymes comme *bd* pour *borderwidth*).

La valeur pour chaque clé est :

· pour la plupart des entrées, un cinq-tuple : (le nom d'option, la clé d'option de base de données, la classe d'option de base de données, la valeur par défaut, la valeur actuelle); ou,

· pour des noms de pseudonyme (comme "fg"), un deux-tuple : (nom de pseudonyme, nom standard équivalent).

***w* .destroy()**

L'appel w.destroy() sur un widget *w* détruit *w* et tous ses enfants.

***w* .event_add (*virtual* , * *sequences*)**

Cette méthode créé un événement virtuel dont le nom est donné par l'argument de chaîne *virtual*. Chaque argument supplémentaire décrit un ordre, c'est-à-dire la description d'un événement physique. Quand cet événement arrive, le nouvel événement virtuel est déclenché.

Voir section (p.) pour une description générale des événements virtuels.

***w* .event_delete (*virtual* , * *sequences*)**

Supprime des événements physiques de l'événement virtuel dont le nom est donné par la chaîne *virtual*. Si tous les événements physiques sont enlevés d'un événement virtuel donné, l'événement virtuel ne se produira plus désormais.

***w* .event_generate (*sequence* , ** *kw*)**

Cette méthode produit un événement qui déclenche sans aucun stimulus externe. Le traitement de l'événement est le même que s'il avait été déclenché par un stimulus externe. L'argument *sequence* décrit l'événement qui doit être déclenché. Vous pouvez mettre des valeurs pour des champs choisis dans l'objet événement en fournissant des arguments keyword=value, où *keyword* spécifie le nom d'un champ de l'objet événement.

Voir voir section (p.) pour une description complète sur les événements.

***w* .event_info (*virtual* =None)**

Si vous appelez cette méthode sans un argument, vous récupérerez un ordre de tout des noms d'événement virtuels actuellement définis.

Pour récupérer les événements physiques associés à un événement virtuel, passez à cette méthode le nom de l'événement virtuel et vous récupérerez une séquence des noms d'ordre physiques, ou *None* si l'événement virtuel donné n'a jamais été défini.

w .focus_displayof()

Rend le nom de la fenêtre qui actuellement a le focus sur le même affichage que le widget.

Si aucune telle fenêtre n'a le focus, rend *None*.

Voir section (p.) pour une description générale des entrée de focus.

w .focus_force()

Force le focus au widget. Ce n'est pas recommandé. Il est meilleur d'attendre le directeur de fenêtre pour vous donner le focus. Voir aussi .grab_set_global() ci-dessous.

w .focus_get()

Obtenir le nom du widget qui a le focus dans cette application, si aucun retourne *None*.

w .focus_lastfor()

Cette méthode récupère le nom du dernier widget qui avait le focus de saisie dans la fenêtre top-level qui contient *w*. Si aucun des widget de ce niveau top-level n'a jamais eu le focus de saisie, il rend le nom du widget top-level. Si cette application n'a pas le focus de saisie .focus_lastfor() rendra le nom du widget qui aura le focus la prochaine fois qu'il revient dans cette application.

w .focus_set()

Si l'application du *w* a le focus de saisie, le focus sautera à *w*. Si l'application du *w* n'a pas de focus, Tk se rappellera de le donner à *w* dès que l'application obtiendra le focus.

w .grab_current()

S'il y a un grab (usurpation) en force pour l'affichage du *w*, rend son identificateur, sinon rend *None*. Référez-vous à la section (p.) pour une explication sur les grabs (usurpations).

w .grab_release()

Si *w* a une usurpation en force, l'enlève.

w .grab_set()

Le widget *w* saisit tous les événements pour l'application du *w*. S'il y avait une autre usurpation en force, il part. Voir section (p.) pour une explication sur les usurpations.

w .grab_set_global()

Le widget w saisit tous les événements de l'écran entier. On le considère impoli et devrait être utilisé seulement en cas de grand besoin. Toute autre usurpation en force est supprimée. Essayez de n'utiliser ce pouvoir stupéfiant seulement pour les forces du bien et jamais pour les forces du mal, OK ?

w .grab_status()

S'il y a une usurpation en force locale (mis par .grab_set ()), cette méthode rend la chaîne "local". S'il y a une usurpation en force globale (de .grab_set_global ()), il retourne "global". S'il ni a aucune usurpation en force, il rend *None*.

_w_ .grid_forget()

Voir section (p.).

_w_ .grid_propagate()

Voir section (p.).

_w_ .grid_remove()

Voir section (p.).

_w_ .image_names()

Rend les noms de toutes les images dans l'application du *w* dans une séquence de chaîne.

_w_ .keys()

Rend les noms des options du *w* dans une séquence de chaîne.

_w_ .lift (aboveThis=None)

Si l'argument est None, la fenêtre contenant *w* est déplacée au sommet de l'ordre d'empilement des fenêtres.Pour déplacer la fenêtre juste au-dessus d'une certaine fenêtre top-level *w*, passez *w* comme argument.

_w_ .lower (belowThis=None)

Si l'argument est None, la fenêtre contenant *w* est déplacée au fond de l'ordre d'empilement des fenêtres.Pour déplacer la fenêtre juste au-dessous d'une certaine fenêtre top-level *w*, passez *w* comme argument.

_w_ .mainloop()

Cette méthode doit être appelée, généralement après que tout les widget statiques sont créés, pour commencer à traiter les événements. Vous pouvez quitter la boucle principale avec la méthode .quit() (ci-dessous). Vous pouvez aussi appeler cette méthode à l'intérieur d'un entraîneur d'événement pour reprendre la boucle principale.

_w_ .nametowidget (_name_)

Cette méthode rend le widget réel dont le nom de chemin est *name*. Voir section (p.). Si le nom *name* est inconnu, cette méthode lèvera KeyError.

_w_ .option_add (_pattern_ , _value_ , _priority_ =None)

Cette méthode ajoute des valeurs d'option par défaut à la base de données d'option Tkinter. *Pattern* (le modèle) est une chaîne qui spécifie une valeur par défaut pour les options d'un ou plusieurs widget. Les valeurs prioritaires sont un de :

20	Pour les propriétés globales par défaut de widget.
40	Pour les propriétés par défaut d'applications spécifiques.
60	Pour les options qui viennent de fichiers d'utilisateur comme leur fichier de .Xdefaults.
80	Pour les options qui sont mises après les démarrages d'application. C'est le niveau prioritaire par défaut.

Les priorités de niveau plus haut ont la priorité sur des de niveau inférieur. Voir section (p.) pour une vue d'ensemble de la base de données d'option. La syntaxe de l'argument *pattern* de .option_add() est la même que dans la partie de modèle d'option de la ligne de spécification de ressource.

Par exemple, pour obtenir l'effet de cette ligne de spécification de ressource :

```
*Button*font: times 24 bold
```

Votre application (self dans cet exemple) pourrait inclure ces lignes :

```
self.bigFont = tkFont.Font ( family="times", size=24, weight="bold" )
self.option_add ( "*Button*font", self.bigFont )
```

Tous les widgets Button créés après l'exécution de ces lignes seraient par défaut en police de caractères gras Times 24 (à moins que forcé par une option de police de caractères au constructeur de Bouton).

_w_ .option_clear()

Cette méthode enlève toutes les options de la base de données d'option Tkinter. Cela a l'effet de retour à toutes les valeurs par défaut.

_w_ .option_get (_name_ , _classname_)

Utilisez cette méthode de récupérer la valeur courante d'une option de la base de données d'option Tkinter. Le premier argument est la clé de l'option et le deuxième argument est la clé de la classe. S'il y en a, il rend la valeur de l'option la mieux ciblée. S'il n'y en a aucune, il retourne "".

Voir section (p.) pour comprendre comment les clés correspondent avec les options.

_w_ .option_readfile (_fileName_ , _priority_ =None)

Comme une commodité pour la configuration utilisateur, vous pouvez désigner un fichier nommé où les utilisateurs peuvent mettre leurs options préférées, utilisant le même format que le fichier de .Xdefaults. Alors, quand votre application initialise, vous pouvez passer le nom de ce fichier à cette méthode et les options de ce fichier seront ajoutées à la base de données. Si le fichier n'existe pas, ou son format est invalide, cette méthode lèvera TclError.

Voir section (p.) pour une introduction à la base de données d'options et le format de fichiers d'option.

_w_ .quit()

Cette méthode quitte la boucle principale. Voir .mainloop (), ci-dessus, pour une information sur les boucles principales.

_w_ .rowconfigure()

Voir section (p.).

_w_ .selection_clear()

Si *w* a actuellement une sélection (comme un segment de texte mis en évidence dans un widget d'entrée), supprime cette sélection.

_w_ .selection_get()

Si *w* a actuellement une sélection, cette méthode rend le texte choisi. S'il n'y a aucune sélection, il lève TclError.

_w_ .selection_own()

Fait *w* le propriétaire de la sélection dans l'affichage de *w*, le volant du propriétaire précédent, s'il y en a un.

_w_ .selection_own_get()

Rend le widget qui possède actuellement la sélection dans l'affichage de *w*. Lève TclError s'il n'y a aucune telle sélection.

_w_ .tk_focusFollowsMouse()

Normalement, les cycles du focus passe par un ordre de widget décidés par leur hiérarchie et l'ordre de création; voir section (p.)). Vous pouvez, au lieu de cela, dire à Tkinter de forcer le focus partout où la souris pointe en appelant simplement cette méthode. Cependant, il n'y a aucune méthode facile pour le défaire.

_w_ .tk_focusNext()

Rend le widget qui suit *w* dans l'ordre de traversée du focus. Voir section (p.) pour une explication sur la traversée du focus.

_w_ .tk_focusPrev()

Rend le widget qui précède *w* dans l'ordre de traversée du focus.

_w_ .unbind (_sequence_ , _funcid_ =None)

Cette méthode supprime des engageant de *w* pour l'événement décrit par *sequence*. Si le deuxième argument est un rappel de service attaché à cette *sequence*, ce rappel de service est enlevé et le reste, s'il en a, est laissé en place. Si le deuxième argument est omis, toutes les attaches sont supprimées.

Voir section (p.), ci-dessous, pour une explication générale des attaches d'événement.

_w_ .unbind_all (_sequence_)

Supprime toutes les attaches d'événement partout dans l'application pour l'événement décrit par la *sequence* donnée.

_w_ .unbind_class (_className_ , _sequence_)

Comme .unbind(), mais s'applique à tous les widgets nommés *className* (par exemple, "Entry" ou "Listbox").

_w_ .update()

Cette méthode force la mise à jour de l'affichage. Il devrait être utilisé seulement si vous connaissez ce que vous faites, puisqu'il peut mener au comportement imprévisible ou la formation de boucle. Il ne devrait jamais être appelé d'un rappel de service d'événement ou d'une fonction qui est appelée d'un rappel de service d'événement.

_w_ .update_idletasks()

Quelques tâches dans la mise à jour de l'affichage, comme des widgets redimensionnants et refaisants, sont appelées des tâches inoccupées parce qu'ils sont d'habitude reportés jusqu'à ce que l'application a fini de traiter des événements et est retournée à la boucle principale pour attendre de nouveaux événements.

Si vous voulez forcer l'affichage a être mis à jour avant la prochaine demande de l'application, appelez la méthode w.update_idletasks() sur n'importe quel widget.

_w_ .wait_variable (_v_)

Attend jusqu'à ce que la valeur de variable *v* soit mise, même si la valeur ne change pas. Cette méthode entre une boucle locale d'attente, donc il ne bloque pas le reste de l'application.

_w_ .wait_visibility (_w_)

Attend jusqu'à ce que le widget *w* (typiquement un Top-level) soit visible.

_w_ .wait_window (_w_)

Attend jusqu'à ce que la fenêtre *w* soit détruite.

_w_ .winfo_children()

Rend une liste de tous les enfants de *w*, dans leur ordre d'entassement du plus bas au plus haut.

_w_ .winfo_class()

Rend le nom de classe de *w* (par exemple, "Button").

_w_ .winfo_containing (_rootX_ , _rootY_ , _displayof_ =0)

Cette méthode est utilisée pour trouver la fenêtre qui contient le point (*rootX*, *rootY*). Si l'option displayof est *false*, les coordonnées sont par rapport à la fenêtre racine de l'application; si *true*, les coordonnées sont traitées par rapport à la fenêtre top-level qui contient *w*. Si le point indiqué est dans une des fenêtres top-level de l'application, cette méthode retourne cette fenêtre; autrement il rend *None*.

_w_ .winfo_depth()

Rend le nombre de bits par pixel dans l'affichage de *w*.

_w_ .winfo_fpixels (_number_)

Pour n'importe quelle dimension *number* (voir section (p.)), cette méthode retourne la distance en pixels sur l'affichage de *w*, comme un nombre à virgule flottante.

_w_ .winfo_geometry()

Rend la Geometry string décrivant la taille et l'emplacement sur l'écran de *w*. Voir section (p.).

La géométrie n'est pas correcte tant que l'application n'ait mis à jour ses tâches inoccupées. Particulièrement toute les géométries sont initialement "1x1+0+0" jusqu'à ce que les widgets et le directeur de géométrie aient négocié leurs tailles et positions. Voir la méthode .update_idletasks(), ci-dessus, dans cette section pour voir comment assurer que la géométrie du widget est à jour.

_w_ .winfo_height()

Rend la hauteur actuelle de *w* en pixels. Voir les remarques sur la mise à jour de géométrie sous .winfo_geometry(), ci-dessus. Vous pouvez préférer utiliser .winfo_reqheight(), décrit ci-dessous, qui est toujours à jour.

_w_ .winfo_id()

Rend un entier qui identifie uniquement *w* dans sa fenêtre top-level. Vous en aurez besoin pour la méthode .winfo_pathname(), ci-dessous.

_w_ .winfo_ismapped()

Cette méthode retourne *true* si w est mapped, false autrement. Un widget est mapped s'il a été gridded (ou placé ou empaqueté, si vous utilisez un des autres directeurs de géométrie) dans son parent et si son parent est mapped, et cetera jusqu'à la fenêtre top-level.

_w_ .winfo_manager()

Si *w* n'a pas été gridded (ou placé via un des autres directeurs de géométrie), cette méthode rend une chaîne vide. Si *w* a été gridded ou placé, il rend une chaîne nommant le directeur de géométrie pour *w* : cette valeur sera un de "grid", "pack", "place", "canvas", ou "text".

_w_ .winfo_name()

Cette méthode rend le nom de *w* relatif à son parent. Voir section (p.). Aussi voir .winfo_pathname(), ci-dessous, pour découvrir comment obtenir le nom du chemin d'un widget.

_w_ .winfo_parent()

Rend le nom du chemin du parent de *w*, ou une chaîne vide si *w* est une fenêtre top-level. Voir section (p.) ci-dessus, pour plus d'informations sur les noms de chemin de widget.

_w_ .winfo_pathname (_id_ , _displayof_ =0)

Si l'argument displayof est *false*, rend le nom de chemin de fenêtre du widget avec l'identificateur unique id dans la fenêtre principale de la l'application. Si displayof est *true*, le numéro d'identification spécifie un widget dans la même fenêtre top-level que *w*. Voir section (p.) ci-dessus, pour plus d'informations sur les noms de chemin de widget.

_w_ .winfo_pixels (number)

Pour n'importe quel *number* de dimension (voir Dimensions, ci-dessus), cette méthode retourne la distance en pixels sur l'affichage de *w*, dans un entier.

_w_ .winfo_pointerx()

Rend la même valeur que la coordonnée de x rendue par .winfo_pointerxy().

_w_ .winfo_pointerxy()

Rend un tuple (x, y) contenant les coordonnées de l'indicateur de souris par rapport à la fenêtre racine de *w*. Si l'indicateur de souris n'est pas sur le même écran, retourne (-1,-1).

_w_ .winfo_pointery()

Rend la même valeur de la coordonnée d'y rendue par .winfo_pointerxy().

_w_ .winfo_reqheight()

Ces méthodes rendent la hauteur demandée du widget *w*. C'est la hauteur minimale nécessaire pour que tout le contenu de *w* ait la pièce dont il a besoin. La hauteur réelle peut être différente en raison des négociations avec le directeur de géométrie.

_w_ .winfo_reqwidth()

Rend la largeur demandée du widget *w*, la largeur minimale nécessaire pour contenir *w*. Comme avec .winfo_reqheight(), la largeur réelle peut être différente en raison des négociations avec le directeur de géométrie.

_w_ .winfo_rgb (color)

Pour n'importe quelle couleur donnée, cette méthode rend la spécification colorée bleue verte rouge équivalente comme 3-tuple (r, g, b), où chaque nombre est un entier dans la gamme [0, 65536). Par exemple, si la couleur est "green" (verte), cette méthode retourne le 3-tuple (0, 65535, 0). Pour plus sur les spécifications de couleurs, voir section (p.).

_w_ .winfo_rootx()

Rend la coordonnée x du gauche côté de la fenêtre racine de *w* par rapport au parent de *w*.

Si *w* a une frontière, c'est le bord extérieur de la frontière.

_w_ .winfo_rooty()

Rend la coordonnée y du côté supérieur de la fenêtre racine de *w* par rapport au parent de *w*.

Si *w* a une frontière, c'est le bord supérieur de la frontière.

_w_ .winfo_screenheight()

Rend la hauteur de l'écran en pixels.

_w_ .winfo_screenmmheight()

Rend la hauteur de l'écran en millimètres.

_w_ .winfo_screenmmwidth()

Rend la largeur de l'écran en millimètres.

_w_ .winfo_screenvisual()

Rend une chaîne qui décrit la mode d'affichage des couleurs. C'est d'habitude "truecolor" pour des affichages 16 ou 24 bits, "pseudocolor" pour des affichages 256 couleurs.

***w* .winfo_screenwidth()**

Rend la largeur de l'écran en pixels.

***w* .winfo_toplevel()**

Rend la fenêtre top-level contenant *w*. Cette fenêtre supporte toutes les méthodes sur des widgets top-level; voir section (p.).

***w* .winfo_viewable()**

Un attribut qui rend une valeur *true* si *w* est viewable, c'est-à-dire si lui et tous ses ancêtres dans le même top-level sont mapped.

***w* .winfo_width()**

Rend la largeur actuelle de *w* en pixels. Voir les remarques sur la mise à jour de géométrie sous .winfo_geometry(), ci-dessus. Vous pouvez préférer utiliser la méthode .winfo_reqwidth(), décrite ci-dessus; c'est toujours à jour.

***w* .winfo_x()**

Rend la coordonnée x du côté gauche de *w* par rapport à son parent. Si *w* a une frontière, c'est le bord extérieur de la frontière.

***w* .winfo_y()**

Rend la coordonnée y du côté supérieur de *w* par rapport à son parent. Si *w* a une frontière, c'est le bord extérieur de la frontière.

26 - Standardisation d'apparence

Il est facile d'appliquer des couleurs, des polices de caractères et d'autres options aux widgets quand vous les créez. Cependant,

 Si vous voulez beaucoup de widgets avec la même couleur de fond ou la même police de caractères, il est ennuyeux de spécifier chaque option chaque fois et

 Il est agréable de laisser l'utilisateur ignorer vos choix avec leurs combinaisons de couleurs préférées, polices de caractères et autres choix

En conséquence, nous utilisons l'idée d'une base de données d'options pour paramétrer des valeurs d'option par défaut.

 Votre application peut spécifier un fichier (comme le fichier standard .Xdefaults utilisé par le Système X Window) qui contient les préférences de l'utilisateur. Vous pouvez paramétrer votre application pour lire le fichier et dire à Tkinter de l'utiliser par défauts. Voir la section sur la méthode .option_readfile(), ci-dessus, dans la section (p.), pour connaître la structure de ce fichier.

 Votre application peut directement spécifier les options par défauts pour un ou plusieurs types de widget en utilisant la méthode .option_ ajoutent(); voir cette méthode sous la section (p.).

Avant de commencer l'explication comment les options sont paramétrees, on considère le problème de personnaliser l'apparence des GUIs en général. Nous pourrions donner à chaque widget dans l'application un nom et demander ensuite à l'utilisateur de spécifier chaque propriété de chaque nom. Mais c'est encombrant et ferait aussi que

l'application serait compliquée à reconfigurer- si le designer ajoute de nouveaux widgets, l'utilisateur devrait décrire chaque propriété de chaque nouveau widget.

Ainsi, la base de données d'option permet au programmeur et à l'utilisateur de spécifier des modèles généraux décrivant quels widgets configurer.

Ces modèles fonctionnent sur les noms des widgets, mais les widgets sont nommés utilisant deux arrangements de désignation parallèles :

a. Chaque widget a un nom de classe. Par défaut, le nom de classe est le même que celui du constructeur de classe : "Button" pour boutons, "Frame" pour un cadre, et cetera. Cependant, vous pouvez créer les nouvelles classes de widgets, usuellement héritant de la classe de Cadre et leur donner les nouveaux noms de votre propre création. Voir section (p.) pour plus de détails.

b. Vous pouvez aussi donner un *instance name* (nom d'instance) à n'importe quel widget. Le nom par défaut d'un widget est habituellement un numéro vide de sens (voir section (p.)). Cependant, comme avec les classes de widget, vous pouvez assigner un nom à n'importe quel widget. Voir section (p.) pour plus de détails.

Chaque widget dans chaque application a donc deux hiérarchies de noms - la hiérarchie de nom de classe et la hiérarchie de nom d'instance. Par exemple, un bouton incorporé dans un widget de texte qui est incorporé dans un cadre aurait la hiérarchie de classe Frame.Text.Button. Il pourrait aussi avoir une hiérarchie d'instance quelque chose comme .mainFrame.messageText.panicButton si vous avez ainsi nommé toutes les instances. Le point initial signifie la fenêtre racine; voir section (p.) pour plus d'informations sur noms de chemin de fenêtre.

Le mécanisme de base de données d'option peut se servir des noms de classe ou des noms d'instance dans la définition d'options, donc vous pouvez faire des options a appliquer aux classes entières (par exemple, tous les Button ont un contexte bleu) ou aux instances spécifiques (par exemple, le Panic Button a des lettres rouges). Après avoir regardé comment nommer des classes et des instances, dans section (p.), nous discuterons comment la base de données d'options travaille vraiment .

26-1 - Comment nommer une classe de widgets

Par exemple, supposez que le Juke-box est une nouvelle classe de widget que vous avez créée. Il est probablement mieux d'avoir des nouvelles classes de widget qui héritent de la classe de Cadre, aussi dans Tkinter il agit comme un cadre et vous pouvez arranger d'autres widgets tel que des étiquettes, des entrées et des boutons à l'intérieur.

Vous mettez le nouveau nom de classe de widget en passant le nom à l'attribut de class_ au constructeur parental dans le constructeur de votre nouvelle classe. Voici un fragment de code qui définit la nouvelle classe :

```
class Jukebox(Frame):
def __init__(self, master):
"Constructor for the Jukebox class"
Frame.__init__ ( self, master, class_="Jukebox" )
self.__createWidgets()
...
```

26-2 - Comment nommer une instance de widgets

Pour donner un nom d'instance à un widget spécifique dans votre application, mettre à l'option de nom du widget une chaîne contenant le nom.

Voici un exemple d'un nom d'instance. Supposons que vous créez plusieurs boutons dans une application et que vous vouliez qu'un des boutons ait un nom d'instance de panicButton. Votre appel au constructeur pourrait ressembler à cela :

```
self.panic = Button ( self, name="panicButton", text="Panic", ...)
```

26-3 - Lignes de spécification de Ressource

Chaque ligne dans un fichier d'option spécifie la valeur d'une ou plusieurs options dans une ou plusieurs applications et a un de ces formats :

```
app option-pattern: value
option-pattern: value
```

La première forme met des options seulement quand le nom de la l'application correspond à *app*; la deuxième forme met des options pour toutes les applications.

Par exemple, si votre application est appelée *xparrot*, une ligne de la forme

```
 xparrot*background: LimeGreen
```

met toutes les options de fond dans l'application *xparrot* au vert citron. (Utilisez l'option -nom sur la ligne de commande en lançant votre application pour mettre le nom à "*xparrot*".)

La partie de modèle d'option a cette syntaxe :

```
{{*|.}name}...option
```

C'est-à-dire chaque modèle d'option est une liste de zéro ou plus de noms, dont chacun est précédé par un astérisque ou une période. Le dernier nom dans la série est le nom de l'option que vous mettez. Chacun du reste des noms peut être :

- Le nom d'une classe de widget (capitalized), ou
- Le nom d'une instance (lowercased).

La façon dont travail le modèles d'option est un peu compliqué. Commençons par un exemple simple :

```
*font: times 24
```

Cette ligne dit que toutes les options de police de caractères devraient par défaut être en Times 24 points. Le * est appelé le symbole obligatoire libre (*loose binding* symbol) et signifie que ce modèle d'option s'applique à n'importe quelle option de police de caractères n'importe où dans n'importe quelle application.

Comparez cet exemple :

```
*Listbox.font: lucidatypewriter 14
```

La période (.) entre *Listbox* et *font* est appelée le symbole obligatoire serré (*tight binding* symbol) et cela signifie que cette règle s'applique seulement aux options de police de caractères pour des widget de la classe Listbox.

Comme autre exemple, supposez que votre application *xparrot* a des instances de widgets de classe *Jukebox*. Pour mettre une couleur par défaut de fond pour tous les widgets de cette classe *Jukebox*, vous pourriez mettre une ligne dans votre fichier d'options comme cela :

```
xparrot*Jukebox*background: PapayaWhip
```

Le symbole obligatoire libre (*) entre *Jukebox* et *background* fait que cette règle s'applique à n'importe quel attribut de fond de n'importe quel widget n'importe où à l'intérieur d'un *Jukebox*. Comparez cette ligne d'option :

```
xparrot*Jukebox.background: NavajoWhite
```

Cette règle s'appliquera au cadre constituant le widget *Jukebox* lui-même, mais à cause du symbole obligatoire serré il ne s'appliquera pas aux widgets qui sont à l'intérieur du widget *Jukebox*.

Dans la section suivante nous parlerons comment Tkinter comprend exactement quelle valeur de l'option utiliser s'il y a les lignes de spécification de ressource multiples qui s'appliquent.

26-4 - Règles des correspondances de Ressource

Quand vous créez un widget et que vous ne spécifiez pas de valeur pour une certaine option et que deux ou plus spécifications de ressource s'appliquent à cette option, le plus spécifique s'applique.

Par exemple, supposez que votre fichier d'options a ces deux lignes :

```
*background: LimeGreen
*Listbox*background: FloralWhite
```

Les deux spécifications s'appliquent à l'option de fond dans un widget Listbox, mais le deuxième est plus spécifique, donc il gagnera.

En général, les noms dans une spécification de ressource sont une séquence *n1, n2, n3..., o* où chaque *ni* est un nom d'instance ou une classe. On ordonne les noms de classe du plus haut niveau au plus bas et *o* est le nom d'une option.

Cependant, quand Tkinter crée un widget, tout ce qu'il a est le nom de classe et le nom d'instance de ce widget.

Voici les règles de préséance aux spécifications de ressource :

1. Le nom de l'option doit correspondre à la partie *o* du modèle d'option. Par exemple, si la règle est

```
xparrot*indicatoron: 0
```

Cela correspondra seulement aux options nommées indicatoron.

2. Le symbole obligatoire serré (.) est plus spécifique que le symbole obligatoire libre (*). Par exemple, une ligne pour *Button.font est plus spécifique qu'une ligne pour *Button*font.

3. Les références aux instances sont plus spécifiques que des références aux classes. Par exemple, si vous avez un bouton dont le nom d'instance est panicButton, une règle pour *panicButton*font est plus spécifique qu'une règle pour *Button*font.

4. Une règle avec plus de niveaux est plus spécifique. Par exemple, une règle pour *Button*font est plus spécifique qu'une règle pour *font.

5. Si deux règles ont le même numéro de niveaux, celle nommée plus tôt dans la liste est plus spécifique que celle postérieure. Par exemple, une règle pour xparrot*font est plus spécifique qu'une règle pour *Button*font.

27 - Connexion de votre logique d'application aux widgets

Les sections précédentes ont parlé de la façon d'arranger et configurer les widgets la façade de l'application.

Ensuite, nous parlerons de la façon de connecter les widgets à la logique qui effectue les actions que l'utilisateur demande.

 Pour faire que votre application réponde aux événements comme des clics de souris ou des entrées au clavier, il y a deux méthodes :

 Quelques commandes comme des boutons ont un attribut de commande qui vous laisse spécifier une procédure, appeler un entraîneur, qui sera appelé à chaque clics d'utilisateur. La chronologie pour utiliser un widget Bouton est très spécifique, cependant l'utilisateur doit déplacer l'indicateur de souris sur le widget avec le bouton 1 de souris en haut, appuyer ensuite le bouton 1 de souris et relacher ensuite le bouton 1 de souris tandis qu'il est toujours sur le widget. Aucune autre chronologie "n'appuiera" un widget Bouton.

 Il y a un mécanisme beaucoup plus général qui peut laisser votre application réagir à beaucoup plus de sortes d'entrées: l'appui ou la relache de n'importe quel touche du clavier ou des boutons de souris; le mouvement de la souris dans, autour, ou or d'un widget; et beaucoup d'autres événements. Comme avec des entraîneurs de commande, dans ce mécanisme vous écrivez les procédures d'entraîneur qui seront appelées chaque fois que les certains types d'événements arrivent. Ce mécanisme est expliqué sous la section (p.).

 Beaucoup de widgets exigent que vous utilisiez des variables de contrôle, les objets spéciaux qui connectent des widgets ensemble et à votre programme, pour que vous puissiez lire et mettre les propriétés des widgets. Les variables de contrôle seront expliqués dans la section suivante.

28 - Variables de Contrôle : Valeurs des widgets

Une variable de contrôle de Tkinter est un objet spécial qui agit comme une variable régulière de Python dans laquelle est un conteneur pour une valeur, comme un nombre ou une chaîne.

Une qualité spéciale d'une variable de contrôle est qu'il peut être partagé par un certain nombre de widgets différents et la variable de contrôle peut se rappeler tous les widgets qui le partagent actuellement. Cela signifie, particulièrement que si votre programme stocke une valeur v dans une variable de contrôle c avec sa méthode c.set(v), n'importe quel widget qui est lié avec cette variable de contrôle est automatiquement mis à jour sur l'écran.

Tkinter utilise des variables de contrôle pour un certain nombre de fonctions importantes, par exemple :

 Les checkbuttons utilisent une variable de contrôle pour tenir l'état actuel du checkbutton (on ou off).

 Une simple variable de contrôle est partagée par un groupe de radiobuttons et peut être utilisée pour dire lequel d'entre eux est actuellement positionné. Quand l'utilisateur clique sur un radiobutton dans un groupe, le partage de cette variable de contrôle est le mécanisme des groupes de radiobuttons Tkinter par lequel quand vous couchez un, tout autre radiobutton dans le groupe est relevé.

 Les variables de contrôle sont des chaînes de caractères dans plusieurs applications. Normalement le texte affiché dans un widget Entrée est lié à une variable de contrôle. Dans plusieurs autres commandes, il est possible d'utiliser une variable de contrôle de chaîne pour mettre le texte comme les étiquettes de checkbuttons et radiobuttons et le contenu de widgets Etiquettes.
Par exemple, vous pourriez lier un widget Entrée avec un widget Etiquette pour que quand l'utilisateur change le texte dans l'entrée et presse la touche Entrée, l'étiquette est automatiquement mise à jour pour afficher le même texte.

Pour obtenir une variable de contrôle, utilisez un de ces constructeurs de classe, selon le type de valeurs que vous devez y stocker :

```
v = DoubleVar() # Holds a float; default value 0.0
v = IntVar() # Holds an integer; default value 0
v = StringVar() # Holds a string; default value ""
```

.get()

Rend la valeur actuelle de la variable.

.set (*value*)

Change la valeur actuelle de la variable. Si plusieurs options de widget sont enchaînées à cette variable, ces widgets seront mis à jour à la prochaine boucle principale; voir .update_idletasks(),section (p.) pour plus d'informations sur le contrôle de ce cycle de mise à jour.

Voici quelques commentaires sur le comment les variables de contrôle sont utilisées avec des widgets spécifiques :

Button

Vous pouvez mettre son textvariable à un StringVar. A n'importe quel momment cette variable est changée, le texte sur le bouton sera mis à jour pour afficher la nouvelle valeur. Cette option n'est pas nécessaire à moins que le texte du bouton ne doive en réalité changer: utilisez l'attribut text si l'étiquette du bouton est statique.

Checkbutton

Normalement, vous mettrez l'option variable du widget à un IntVar et cette variable sera mise à 1 quand le checkbutton est allumé et à 0 quand il est éteint. Cependant, vous pouvez choisir des valeurs différentes pour ces deux états avec les options onvalue et offvalue, respectivement.

Vous pouvez même utiliser un StringVar comme variable du checkbutton et fournir des valeurs de chaîne pour offvalue et onvalue. Par exemple :

```
self.spamVar = StringVar()
self.spamCB = Checkbutton ( self, text="Spam?",
variable=self.spamVar, onvalue="yes", offvalue="no" )
```

Si ce checkbutton est branché, self.spamVar.get() rendra la chaîne "yes"; si le checkbutton éteint, ce même appel rendra la chaîne "no". En outre, votre programme peut allumer le checkbutton en appelant .set ("yes").

Vous pouvez aussi mettre l'option textvariable d'un checkbutton à un StringVar. Alors vous pouvez changer l'étiquette de texte du checkbutton à l'utilisation la méthode .set() sur cette variable.

Entry

Mettre son option textvariable à un StringVar. Utilisez la méthode .get() de cette variable pour récupérer le texte actuellement affiché dans le widget. Vous pouvez aussi utiliser la méthode .set() de la variable pour changer le texte affiché dans le widget.

Label

Vous pouvez mettre son option textvariable à un StringVar. Alors n'importe quel appel à la méthode .set() de la variable changera le texte affiché sur l'étiquette. Ce n'est pas nécessaire si le texte de l'étiquette est statique; utilisez l'attribut de texte pour les étiquettes qui ne changent pas tandis que la l'application fonctionne.

Menubutton

Si vous voulez être capables de changer le texte affiché sur le menubutton, mettre son option textvariable à un StringVar et utiliser la méthode .set() de cette variable pour changer le texte affiché.

Radiobutton

L'option variable doit être mise à une variable de contrôle, IntVar ou un StringVar. L'ensemble du radiobutton dans un groupe fonctionnel doit partager la même variable de contrôle.

Mettre l'option de valeur de chaque radiobutton dans le groupe à une valeur différente. Chaque fois que l'utilisateur actionne un radiobutton, la variable sera mise à l'option de valeur du radiobutton et tout le reste du radiobutton qui partage le groupe sera éteint.

Vous pourriez vous demander, quel est l'état dans un groupe de radiobuttons quand la variable de contrôle n'a jamais été mise et que l'utilisateur n'a jamais cliqué sur eux ? Chaque variable de contrôle a une valeur par défaut : 0 pour un IntVar, 0.0 pour un DoubleVar et "" pour un StringVar. Si un des radiobuttons a cette valeur, ce radiobutton sera positionné initialement. Si l'option de valeur d'aucun des radiobutton ne correspond à la valeur de la variable, le radiobuttons entier semblera être éteint.

Si vous voulez changer l'étiquette de texte sur un radiobutton pendant l'exécution de votre application, mettez son option textvariable à un StringVar. Alors votre programme peut changer l'étiquette de texte en passant le nouveau texte d'étiquette avec la méthode .set() de la variable.

Scale

Pour un widget échelle, mettez son option variable à une variable de contrôle de n'importe quelle classe et mettez ses option *from_* et *to* aux valeurs de limitation pour les fins opposées de l'échelle.

Par exemple, vous pourriez utiliser un IntVar et mettre l'échelle from_= 0 et to=100. Alors chaque changement utilisateur au widget changerait la valeur de la variable à une certaine valeur entre 0 et 100 compris.

Votre programme peut aussi déplacer le slider en utilisant la méthode .set() sur la variable de contrôle. Pour continuer le susdit exemple, .set(75) déplacerait le slider à une position 3/4 de la voie le long de sa cuvette.

Pour paramétrer un widget Echelle pour des valeurs réelles, utilisez un DoubleVar.

Vous pouvez utiliser un StringVar comme la variable de contrôle d'un widget Echelle. Vous devrez toujours fournir des valeurs numérique à *from_* et *to*, mais la valeur numérique du widget sera convertie en chaîne pour le stockage dans le StringVar. Utilisez l'option de chiffres de l'échelle pour contrôler la précision de cette conversion.

29 - Focus : routines d'entrée au clavier

Dire qu'un widget a le focus signifie que l'entrée au clavier est actuellement adressée à ce widget.

 Par le focus *traversal*, nous voulons dire l'ordre des widgets qui seront visités comme les mouvements d'utilisateur d'un widget a un autre widget avec la touche de tabulation. Voir ci-dessous pour les règles à cet ordre.

 Vous pouvez traverser en arrière par l'utilisation de shift-tab.

 Les widgets Entrée et Texte sont destinés pour accepter l'entrée au clavier et si un widget entrée ou texte a actuellement le focus, n'importe quels caractères tapé sera ajouté à son texte. Les caractères de rédaction habituels comme ? et ? auront leurs effets habituels.

 Parce que les widgets Texte peuvent contenir des caractères tab, vous devez utiliser la séquence spéciale *control-tab* pour déplacer le focus devant un widget texte.

 La plupart des autres types de widgets seront normalement visités par le focus traversal, et quand ils ont le focus :

 Les widgets bouton peuvent être "appuyés" en appuyant la barre d'espace.

 Les widgets Checkbutton peuvent être basculé entre les états allumé et éteint utilisant la barre d'espace.

 Dans les widgets Listbox, les touches ? et ? indexent le rouleau vers le haut ou vers le bas d'une ligne; les touches *PageUp* et *PageDown* indexent le rouleau par pages; et la barre d'espace sélectionne la ligne actuelle, ou la déselectionne si elle a été déjà choisi.

 Vous pouvez sélectionner un widget Radiobutton en appuyant la barre d'espace

 Les widgets Echelle Horizontaux répondent au touches ? et ? et les verticaux répondent à ? et ?.

 Dans un widget Scrollbar, les touches *PageUp* et *PageDown* déplacent le scrollbar par pageloads. Les touches ? et ? déplaceront les scrollbars verticales par unités et les touches ? et ? déplaceront les scrollbars horizontales par unités.

 Beaucoup de widgets sont pourvus d'un contour appelé *focus highlight* qui montre à l'utilisateur que le widget a le point culminant. C'est normalement un cadre noir mince situé juste à l'extérieur de la frontière du widget (s'il en a). Pour les widgets qui n'ont pas normalement de *focus highlight* (spécifiquement, cadres, étiquettes et menus), vous pouvez mettre l'option highlightthickness à une valeur nonzéro pour rendre le *focus highlight* visible.

 Vous pouvez aussi changer la couleur du *focus highlight* en utilisant l'option highlightcolor.

 Les widgets de classe Cadre, Etiquette et Menu ne sont pas normalement visités par le focus. Cependant, vous pouvez paramétrer leurs options takefocus à 1 les inclure dans la traversée de focus. Vous pouvez aussi sortir n'importe quel widget de la traversée de focus en mettant son option takefocus à 0.

L'ordre dans lequel la touche de tabulation traverse les widgets est :

 Pour les widgets qui sont les enfants du même parent, le focus entrent dans le même ordre que les widgets ont été créé.

 Pour les widgets parents qui contiennent d'autres widgets (comme des cadres), le focus visite le widget parent d'abord (à moins que son option takefocus ne soit 0), puis visite les widgets enfants, récursivement, dans l'ordre ou ils ont été créés.

Pour résumer : pour paramétrer l'ordre de traversée de focus de vos widgets, créez-les dans cet ordre. Extraire les widgets de l'ordre de traversée en mettant leurs options takefocus à 0 et pour ceux qui ont par défaut l'option takefocus à 0, la mettre à 1 si vous voulez les ajouter à l'ordre de traversée.

Ce qui est au dessus décrit le fonctionnement par défaut du focus de saisie dans Tkinter. Il y a une autre façon complètement différente de le traiter - laisser le focus aller partout où la souris va. Sous la section (p.), voir la méthode .tk_focusFollowsMouse().

Vous pouvez aussi ajouter, changer ou supprimer la façon que chacun met en marche les fonctions de clavier à l'intérieur de n'importe quel widget en utilisant des attaches d'événement. Voir section (p.) pour plus de détails.

30 - Evênements : répondre à des stimuli

Un événement est quelque chose qui arrive à votre application - par exemple, l'utilisateur presse une touche ou clique ou traîne la souris - auquel l'application doit réagir.

Les widgets ont normalement beaucoup de comportements incorporés. Par exemple, un bouton réagira à un clic de souris en appelant son rappel de service de commande. Autre exemple, si vous déplacez le focus à un widget d'entrée et tapez une lettre, cette lettre est ajouté au contenu du widget.

Cependant, la capacité d'événement obligatoire de Tkinter vous permet d'ajouter, changer, ou supprimer des comportements.

D'abord, quelques définitions :

 Un *event* (événement) est quelque chose qui se produit et que votre application doit connaître.

 Un *event handler* (entraîneur d'événement) est une fonction dans votre application qui est appelée quand un événement arrive.

 Nous l'appelons *binding* (attache) quand votre application lie un entraîneur d'événement qui est appelé quand un événement arrive à un widget.

30-1 - Les niveaux d'attache

Vous pouvez lier un entraîneur à un événement à un de ces trois niveaux :

1 Attache de cas : Vous pouvez lier un événement à un widget spécifique. Par exemple, vous pourriez lier la touche PageUp dans un widget Canvas à un entraîneur qui fait monter le Canvas d'une page. Pour lier un événement d'un widget, appelez la méthode .bind() sur ce widget (voir section (p.)).
Par exemple, supposez que vous avez un widget Canvas nommé self.canv et que vous voulez dessiner une goutte orange sur le canvas chaque fois que l'utilisateur clique sur le bouton 2 de la souris (le bouton du milieu). Pour mettre en oeuvre ce comportement :

```
self.canv.bind ( "&lt;Button-2&gt;", self.__drawOrangeBlob )
```

Le premier argument est un descripteur d'ordre qui dit à Tkinter que chaque fois que le bouton du milieu de la souris descend, il doit appeler l'entraîneur d'événement nommé self.__drawOrangeBlob. (Voir section (p.), ci-dessous, pour une vue d'ensemble comment écrire des entraîneurs comme .__drawOrangeBlob()). Notez que vous omettez les parenthèses après le nom d'entraîneur, pour que le Python passe en référence l'entraîneur au lieu d'essayer de l'appeler tout de suite.

1 Attache de classe : Vous pouvez lier un événement à tous les widgets d'une classe. Par exemple, vous pourriez lier tous les widgets Bouton pour répondre aux clics du bouton du milieu de la souris en changeant dans les deux sens entre des étiquettes anglaises et japonaises. Pour lier un événement à tous les widgets d'une classe, appelez la méthode .bind_class() sur n'importe quel widget (voir la section (p.), ci-dessus).
Par exemple, supposez que vous avez plusieurs canvas et que vous voulez lier le bouton 2 de souris pour dessiner une goutte orange dans n'importe lequel d'entre eux. Plutôt que devoir pour appeler .bind() pour chacun d'entre eux, vous pouvez les tous les paramétrer avec quelque chose comme cela :

```
self.bind_class ( "Canvas", "&lt;Button-2&gt;",
self.__drawOrangeBlob )
```

3.Attache d'application : Vous pouvez lier une attache pour qu'un certain événement appelle un entraîneur peu importe quel widget a le focus ou est sous la souris. Par exemple, vous pourriez lier la touche *PrintScrn* à tous les widgets d'une application, pour qu'il imprime l'écran peu importe quel widget obtient cette touche. Pour lier un événement au niveau d'application, appelez la méthode .bind_all() sur n'importe quel widget (voir section (p.)).

Voici comment vous pourriez lier la touche *PrintScrn*, dont "le nom de touche" est "**Print**" :

```
self.bind_all ( "&lt;Key-Print&gt;", self.__printScreen )
```

30-2 - Ordres d'événement

Tkinter a une méthode puissante et générale pour vous permettre de définir exactement quels événements, tant spécifique que général, vous voulez lier aux entraîneurs.

En général, un ordre d'événement est une chaîne contenant un ou plusieurs modèles d'événement. Chaque modèle d'événement décrit une chose qui peut arriver. S'il y a plus qu'un modèle d'événement dans un ordre, l'entraîneur sera appelé seulement quand tous les modèles arrivent dans ce même ordre.

La forme générale d'un modèle d'événement est :

```
&lt;[<i>modifier</i>-]...<i>type</i>[-<i>detail</i>]&gt;
```

 Le modèle entier est inclus à l'intérieur <>.

 Le *type* d'événement décrit la sorte générale d'événement, comme un appui sur une touche ou un clic de souris. Voir section (p.).

 Vous pouvez ajouter des articles facultatifs de modificateur avant le type pour spécifier des combinaisons comme les touches *shift* ou *control* qui devront être appuyée pendant l'appui des autres touches ou clics de souris. Section (p.).

 Vous pouvez ajouter des articles facultatifs de détail pour décrire quelle touche ou bouton de souris vous cherchez. Pour les boutons de souris, c'est 1 pour le bouton 1, 2 pour le bouton 2, ou 3 pour le bouton 3.

 L'installation habituelle a le bouton 1 à gauche et le bouton 3 à droite, mais les gauchers peuvent intervertir ces positions.

 Pour des touches du clavier, c'est le caractère de la touche (pour des touches de caractère seul comme A un *) ou le nom de la touche; voir Section (p.) pour une liste de tous les noms des touches.

Voici quelques exemples pour vous donner une idée de modèles d'événement :

<Button-1>	L'utilisateur a appuyé le premier bouton de souris.
<KeyPress-H>	L'utilisateur a appuyé la touche H.
<Control-Shift-KeyPress-H>	L'utilisateur a appuyé control-shift-H.

30-3 - Type d'événement

Le jeu complet de types d'événement est plutôt grand, mais beaucoup d'entre eux ne sont pas généralement utilisé. Voici la plupart d'entre ceux que vous aurez besoin :

Type	Nom	Description
36	Activate	Un widget change d'être inactif à être actif. Cela se réfère aux changements dans l'option d'état d'un

		widget comme un bouton changeant d'inactif à actif.
4	Button	L'utilisateur a appuyé un des boutons de souris. La partie de détail spécifie quel bouton. Pour le support de roue de souris sous Linux, utilisez Button-4 (le rouleau vers le haut) et Button-5 (le rouleau vers le bas). Sous Linux, votre entraîneur pour des attaches de roue de souris distinguera entre le rouleau en haut et rouleau enbas en examinant le champ .num du cas d'Evénement; voir section (p.).
5	ButtonRelease	L'utilisateur maintient l'appui sur un bouton de souris. C'est probablement un meilleur choix dans la plupart des cas que l'événement de Bouton, parce que si l'utilisateur appuie accidentellement le bouton, il peut déplacer la souris hors du widget pour faire ressortir de l'événement.
22	Configure	L'utilisateur a changé la taille d'un widget, par exemple en traînant un coin ou un côté de la fenêtre.
37	Deactivate	Un widget change d'être actif à être inactif. Cela se réfère aux changements dans l'option d'état d'un widget comme un radiobutton changeant d'actif à inactif.
17	Destroy	Un widget est détruit.
7	Enter	L'utilisateur a déplacé l'indicateur de souris dans une partie visible d'un widget. (C'est différent de la touche *enter*, qui est un événement *KeyPress* pour une touche dont le nom est en réalité "*return*".)
12	Expose	Cet événement arrive chaque fois qu'au moins une certaine partie de votre application ou widget devient visible après avoir été dissimulé par une autre fenêtre.
9	FocusIn	Un widget a obtenu le focus de saisie (voir section (p.) pour une introduction

		générale du focus de saisie.) Cela peut arriver en réponse à un événement d'utilisateur (comme l'utilisation de la touche de tabulation pour déplacer le focus entre des widgets) ou par programme (par exemple, votre programme appelle le .focus_set() sur un widget).
10	FocusOut	Le focus de saisie a été déplacé d'un widget. Comme avec FocusIn, l'utilisateur ou votre programme peut causer cet événement.
2	KeyPress	L'utilisateur a appuyé une touche du clavier. La partie de détail spécifie quelle touche. Cette *keyword* peut être abrégé *key*.
3	KeyRelease	L'utilisateur relache une touche.
8	Leave	L'utilisateur a déplacé l'indicateur de souris hors d'un widget.
19	Map	Un widget est mapped, c'est-à-dire visible dans l'application. Cela arrivera, par exemple, quand vous appelez la méthode .grid() du widget.
6	Motion	L'utilisateur a déplacé l'indicateur de souris entièrement dans un widget.
38	MouseWheel	L'utilisateur a déplacé la roue de souris vers le haut ou vers le bas. A présent, fonctionne sur Windows et MacOS, mais pas sous Linux. Pour Windows et MacOS, voir l'explication du champ .delta du cas d'Evénement dans section (p.). Pour Linux, voir la note ci-dessus sous Button.
18	Unmap	Un widget est unmapped et n'est plus visible. Cela arrive, par exemple, quand vous utilisez la méthode .grid_remove() du widget.
15	Visibility	Arrive quand au moins une partie de la fenêtre d'application devient visible sur l'écran.

30-4 - Modificateurs d'événement

Les noms de modificateur que vous pouvez utiliser dans des ordres d'événement incluent :

Alt	Vrai quand l'utilisateur maintient la touche alt.
Any	Ce modificateur généralise un type d'événement. Par exemple, le modèle d'événement "<Any-KeyPress>" s'applique à la pression de n'importe quelle touche.
Control	Vrai quand l'utilisateur maintient la touche control.
Double	Spécifie deux événements arrivant à suivre. Par exemple, <Double-Button-1> décrit deux appuis du bouton 1 dans un enchaînement rapide.
Lock	Vrai quand l'utilisateur a appuyé *shift lock*.
Shift	Vrai quand l'utilisateur maintien la touche des majuscules (*shift*).
Triple	Comme Double, mais spécifie trois événements dans un enchaînement rapide.

Vous pouvez utiliser les formes plus courtes d'événements. Voici quelques exemples :

- "<1>" est identique à "<Button-1>".

- "x" est identique à "< KeyPress-x >".

Notez que vous pouvez quitter l'action d'inclure "<>" pour la plupart des appuis de touches de caractères seuls, mais vous ne pouvez pas le faire pour le caractère espace (dont le nom est "<space>") ou le caractère inférieur (<) (dont le nom est "<less>").

30-5 - Nom des touches

La partie de détail d'un modèle d'événement pour un **KeyPress** ou un événement **KeyRelease** spécifie quelle touche est liée. (Voir le modificateur *Any*, ci-dessus, si vous voulez obtenir tout les touches pressées ou touches relachées).

La table ci-dessous montre plusieurs façons différentes de nommer les touches. Voir section (p.), ci-dessous, pour plus d'informations sur des objets d'Evénement, dont les attributs décriront des touches de ces mêmes façons.

- La colonne .keysym montre le "symbole touche", un nom de chaîne pour la touche. Cela correspond à l'attribut .keysym de l'objet d'Evénement.

- La colonne .keycode est le "code touche". Cela identifie quelle touche a été appuyée, mais le code ne reflète pas l'état de modificateurs divers comme les touches *shift* et *control* et la touche *NumLock*. Ainsi, par exemple, a et A ont le même code touche.

- La colonne **.keysym_num** montre un équivalent de code numérique du symbole clé. A la différence de **.keycode**, ces codes sont différents pour des modificateurs différents. Par exemple, le chiffre 2 sur le clavier numérique (symbole de touche **KP_2**) et la flèche vers le bas sur le clavier numérique (symbole touche **KP_Down**) a le même key code (88), mais des valeurs de **.keysym_num** différentes (65433 et 65458, respectivement).

- La colonne "Key" montre le texte que vous trouverez habituellement sur la touche physique, comme *tab*.

Il y a beaucoup plus de noms de key pour des jeux de caractères internationaux. Cette table montre seulement le jeu " Latin-1" pour le clavier habituel à 101 touches type USA. Pour le jeu actuellement supporté, voir la page manuelle pour Tk keysym values (6) .

.keysym	.keycode	.keysym_num	Key
Alt_L	64	65513	La touche *alt* gauche
Alt_R	113	65514	La touche *alt* droite
BackSpace	22	65288	*Backspace*
Cancel	110	65387	*Break*
Caps_Lock	66	65549	*CapsLock*
Control_L	37	65507	La touche *control* gauche
Control_R	109	65508	La touche *control* droite
Delete	107	65535	*Delete*
Down	104	65364	?
End	103	65367	*end*
Escape	9	65307	*esc*
Execute	111	65378	*SysReq*
F1	67	65470	Touche de fonction F1
F2	68	65471	Touche de fonction F2
Fi	66+i	65469+i	Touche de fonction Fi
F12	96	65481	Touche de fonction F12
Home	97	65360	*home*
Insert	106	65379	*insert*
Left	100	65361	?
Linefeed	54	106	Linefeed (*control-J*)
KP_0	90	65438	0 sur le clavier
KP_1	87	65436	1 sur le clavier
KP_2	88	65433	2 sur le clavier
KP_3	89	65435	3 sur le clavier
KP_4	83	65430	4 sur le clavier
KP_5	84	65437	5 sur le clavier
KP_6	85	65432	6 sur le clavier
KP_7	79	65429	7 sur le clavier
KP_8	80	65431	8 sur le clavier
KP_9	81	65434	9 sur le clavier
KP_Add	86	65451	+ sur le clavier
KP_Begin	84	65437	La touche centre (même touche que 5) sur le clavier
KP_Decimal	91	65439	Decimal (.) sur le clavier
KP_Delete	91	65439	*delete* sur le clavier
KP_Divide	112	65455	/ sur le clavier
KP_Down	88	65433	? sur le clavier
KP_End	87	65436	*end* sur le clavier
KP_Enter	108	65421	*enter* sur le clavier
KP_Home	79	65429	*home* sur le clavier
KP_Insert	90	65438	*insert* sur le clavier
KP_Left	83	65430	? sur le clavier
KP_Multiply	63	65450	x sur le clavier
KP_Next	89	65435	*PageDown* sur le clavier
KP_Prior	81	65434	*PageUp* sur le clavier
KP_Right	85	65432	? sur le clavier
KP_Subtract	82	65453	- sur le clavier
KP_Up	80	65431	? sur le clavier
Next	105	65366	*PageDown*
Num_Lock	77	65407	*NumLock*
Pause	110	65299	*pause*

			Enter se réfère à un événement concernant la souris, pas un appui de touche; voir section (p.).
Right	102	65363	?
Scroll_Lock	78	65300	*ScrollLock*
Shift_L	50	65505	La touche *shift* gauche
Shift_R	62	65506	La touche *shift* droite
Tab	23	65289	La touche *tab*
Up	98	65362	?

30-6 - Ecriture de votre entraîneur : la classe Evénement

Les sections ci-dessus vous expliquent comment décrire quels événements vous voulez manipuler et comment les lier. Laissez-nous maintenant vous former à l'écriture de l'entraîneur qui sera appelé quand l'événement arrive en réalité .

On passera l'entraîneur un objet d'Evénement qui décrit ce qui est arrivé. L'entraîneur peut être une fonction ou une méthode. Voici l'ordre appelant pour une fonction régulière :

```
def handlerName ( event ):
```

Et pour une méthode :

```
def handlerName ( self, event ):
```

Les attributs de l'objet *event* passés à l'entraîneur sont décrit ci-dessous. Certains de ces attributs sont toujours mis, mais certains sont mis seulement pour les certains types d'événements.

.char	Si l'événement a été rapproché d'un **KeyPress** ou **KeyRelease** pour une touche qui produit un caractère ASCII régulier, cette chaîne sera renseignée avec ce caractère. (Pour des touches spéciales comme *delete*, voir l'attribut de **.keysym**, ci-dessous.)
.delta	Pour des événements **MouseWheel** (roue de souris), cet attribut contient un entier dont le signe est positif pour faire défiler en haut, et négatif pour faire défiler en bas. Sous Windows, cette valeur sera un multiple de 120; par exemple, 120 signifie faire défiler en haut d'un pas et -240 faire défiler en bas de deux pas. Sous MacOS, il sera un multiple de 1, ainsi 1 signifie faire défiler en haut d'un pas et -2 faire défiler en bas de deux pas. Pour le support de roue de souris Linux, voir

	la note sur l'attache d'événement de Bouton dans la section (p.).
.height	Si l'événement est un **Configure**, cet attribut est mis à la nouvelle hauteur du widget en pixels.
.keycode	Pour des événements **KeyPress** ou **KeyRelease**, cet attribut est mis à un code numérique qui identifie la touche. Cependant, il n'identifie pas lequel des caractères sur cette touche a été produit, aussi "**x**" et "**X**" ont la même valeur de **.keyCode**. Pour les valeurs possibles de ce champ, voir section (p.).
.keysym	Pour des événements **KeyPress** ou **KeyRelease** impliquant une touche spéciale, cet attribut est mis au nom de chaîne de la touche, par exemple, "**Prior**" pour la touche ***PageUp*** . Voir section (p.) pour une liste complète de noms de **.keysym**.
.keysym_num	Pour des événements **KeyPress** ou **KeyRelease**, c'est mis à une version numérique du champ **.keysym**. Pour les touches régulières qui produisent un caractère simple, ce champ est mis à la valeur d'entier du code ASCII de la touche. Pour des touches spéciales, référez-vous à la section (p.).
.num	Si l'événement a été rapproché d'un bouton de souris, cet attribut est mis au numéro de bouton (1, 2, ou 3). Pour le support de roue de souris sous Linux, lie les Evénements au Button-4 et au Button-5; quand la roue de souris est tourné en haut, ce champ sera 4, ou 5 quand tourné en bas.
.serial	Un numéro de série entier qui est incrémenté chaque fois que le serveur traite une demande du client. Vous pouvez utiliser des valeurs de **.serial** pour trouver la chronologie exacte : ceux avec des valeurs inférieures se sont produit plus tôt
.state	Un entier décrivant l'état de toutes les touches modifiée. Voir la table de masques de modification ci-dessous pour l'interprétation de cette valeur.
.time	Cet attribut est mis à un entier qui n'a aucune signification absolue, mais est incrémenté chaque milliseconde. Cela permet à votre application de déterminer, par exemple, la durée entre deux clics de souris.
.type	Un code numérique décrivant le type d'événement. Pour l'interprétation de ce code, voir section (p.).
.widget	Toujours positionné au widget qui a causé l'événement. Par exemple, si l'événement

	était un clic de souris qui est arrivé sur un **Canvas**, cet attribut sera ce widget **Canvas**.
.width	Si l'événement est un **Configure**, cet attribut est mis à la nouvelle largeur du widget en pixels.
.x	La coordonnée x de la souris au moment de l'événement, par rapport au coin gauche supérieur du widget.
.y	La coordonnée y de la souris au moment de l'événement, par rapport au coin gauche supérieur du widget.
.x_root	La coordonnée x de la souris au moment de l'événement, par rapport au coin gauche supérieur de l'écran.
.y_root	La coordonnée y de la souris au moment de l'événement, par rapport au coin gauche supérieur de l'écran.

Utilisez ces masques pour tester la valeur des bits **.state** pour voir quelles touches de modificateur et boutons ont été appuyés pendant l'événement :

Mask	**Modifier**
0x0001	*Shift*.
0x0002	*Caps Lock*.
0x0004	*Control*.
0x0008	*Alt* gauche.
0x0010	*Num Lock*.
0x0080	*Alt* droit.
0x0100	Bouton 1 de la souris
0x0200	Bouton 2 de la souris
0x0400	Bouton 3 de la souris

Voici un exemple d'un entraîneur d'événement. Sous la Section (p.), ci-dessus, il y a un exemple montrant comment lier des clics du bouton 2 de souris sur un canvas nommé **self.canv** à un entraîneur appelé **self. __ drawOrangeBlob()**. Voici cet entraîneur :

```
def __drawOrangeBlob ( self, event ):
"Draws an orange blob in self.canv where the mouse is."
r = 5 # Blob radius
self.canv.create_oval ( event.x-r, event.y-r,
event.x+r, event.y+r, fill="orange" )
```

Quand cet entraîneur est appelé, la position de souris actuelle est (**event.x, event.y**). La méthode **.create_ ovale()** dessine un cercle dont la limitation de la boîte est un carré et centrée sur cette position et a les côtés de longueur 2*r.

30-7 - Le tour des arguments supplémentaires

Parfois vous voudriez passer d'autres arguments à un entraîneur en plus de l'événement.

Voici un exemple. Supposons que votre application a un tableau de dix checkbuttons dont les widgets sont stockés dans une liste **self.cbList**, indexés par le nombre de checkbutton dans la gamme(10).

Supposons plus loin que vous voulez écrire un entraîneur nommé. **__ cbHandler** pour les événements de **<Button-1>** dans tous ces dix checkbuttons. L'entraîneur peut obtenir le widget **Checkbutton** réel qui l'a déclenché en se référant

à l'attribut de **.widget** de l'objet d'Evénement que l'on passe, mais comment découvre-t-il que l'indice du checkbutton dans **self.cbList** ?

Il serait agréable d'écrire notre entraîneur avec un argument supplémentaire pour le nombre de checkbutton, quelque chose comme cela :

```
def __cbHandler ( self, event, cbNumber )
```

:
Mais on passe aux entraîneurs d'événement seulement un argument, l'événement. Donc nous ne pouvons pas utiliser la fonction ci-dessus à cause d'une disparité dans le nombre d'arguments.

Heureusement, la capacité du Python à fournir des valeurs par défaut pour des arguments de fonction nous donne une sortie. Regardez ce code :

```
def __createWidgets ( self ):

□
self.cbList = [] # Create the checkbutton list
for i in range(10):
cb = Checkbutton ( self, □ )
self.cbList.append ( cb )
cb.grid( row=1, column=i )
def handler ( event, self=self, i=i ): ***1***
return self.__cbHandler ( event, i )
cb.bind ( "&lt;Button-1&gt;", handler )
□
def __cbHandler ( self, event, cbNumber ):
□
```

1 Ces lignes définissent un nouvel entraîneur de fonction qui s'attend à trois arguments. Le premier argument est l'objet d'Evénement a passé à tous les entraîneurs d'événement et les deuxièmes et troisièmes arguments seront mis à leurs valeurs par défaut - les arguments supplémentaires nous devons le passer.

Cette technique peut être étendue pour fournir n'importe quel nombre d'arguments supplémentaires aux entraîneurs.

30-8 - Evénements virtuels

Vous pouvez créer vos propres nouvelles sortes d'événements appelés *virtuel events* (événements virtuels). Vous pouvez leur donner n'importe quel nom que vous voulez tant qu'il est inclus dans les paires doubles de << □ >>.

Par exemple, supposez que vous voulez créer un nouvel événement appelé <<panic>>, qui est déclenché par le bouton 3 de souris ou par la touche *pause*. Pour créer cet événement, appelez cette méthode sur n'importe quel widget *w* :

```
w.event_add ( "&lt;&lt;panic&gt;&gt;", "&lt;Button-3&gt;",
"&lt;KeyPress-Pause&gt;" )
```

Vous pouvez alors utiliser "<<panic>>" dans toute séquence d' événement. Par exemple, si vous utilisez cet appel :

```
w.bind ( "&lt;&lt;panic&gt;&gt;", h )
```

N'importe quel appui sur le bouton 3 de la souris ou sur la touche *pause* dans le widget *w* déclencheront l'entraîneur h.

Voir **.event_add()**, **.event_delete()** et **.event_info()** sous la section (p.) pour plus d'informations sur la création et la gestion d'événements virtuels.

31 - Dialogues contextuels

Tkinter fournit trois modules qui peuvent créer des fenêtres de dialogue contextuelles pour vous :

- La Section (p.), fournit un assortiment de menus contextuel communs pour des tâches simples.
- La Section (p.), permet à l'utilisateur de naviguer pour des fichiers.
- La Section (p.), permet à l'utilisateur de choisir une couleur.

31-1 - Le module de dialogues tkMessageBox

Une fois que vous importez le module tkMessageBox, vous pouvez créer n'importe lequel de ces sept types communs de menu contextuel en appelant des fonctions de cette table.

	.askokcancel(*title*, *message*, *options*)
	.askquestion(*title*, *message*, *options*)
	.askretrycancel(*title*, *message*, *options*)
	.askyesno(*title*, *message*, *options*)
	.showerror(*title*, *message*, *options*)
	.showinfo(*title*, *message*, *options*)
	.showwarning(*title*, *message*, *options*)

Dans chaque cas, le titre (*title*) est une chaîne à afficher dans le sommet de la décoration de fenêtre. L'argument de message (*message*) est une chaîne qui apparaît dans le corps de la fenêtre contextuelle; dans cette chaîne, les lignes sont cassées au des caractères retour à la ligne ("\n").

Les arguments d'option peuvent être n'importe lequel de ces choix.

default

Quel bouton devrait être le choix par défaut ? Si vous ne spécifiez pas cette option, le premier bouton ("OK", "Yes", ou "Retry") sera le choix par défaut.

Pour spécifier quel bouton est le choix par défaut, utiliser **default=** ***C*** , où ***C*** est une de ces constantes définis dans **tkMessageBox** : **CANCEL**, **IGNORE**, **OK**, **NO**, **RETRY**, ou **YES**.

icon

Sélectionne quelle icône apparaît dans le menu contextuel. Utilisez un argument de la forme icon=*I* où *I* est une de ces constantes définis dans tkMessageBox : **ERROR**, **INFO**, **QUESTION**, ou **WARNING**.

parent

Si vous ne spécifiez pas cette option, le menu contextuel apparaît au-dessus de votre fenêtre racine. Pour faire apparaître le menu contextuel susdit sur une certaine fenêtre enfant *W*, utilise l'argument parent=*W*.

Chaque "**ask...**" des fonctions contextuelles rend une valeur qui dépend du bouton sur lequel l'utilisateur a poussé pour enlever le menu contextuel.

. askokcancel, **askretrycancel** et **askyesno** retourne tous une valeur **boolenne** : **True** pour les choix "OK" ou "Yes", **False** pour les choix "No" ou "Cancel".

. askquestion rend u "yes" pour "Yes", ou u "no" pour "No".

31-2 - Le module tkFileDialog

Le module **tkFileDialog** fournit deux fenêtres contextuelles différentes que vous pouvez utiliser pour donner à l'utilisateur la capacité de trouver des fichiers existants ou créer de nouveaux fichiers.

.askopenfilename(*option* = *value* , ...)

Destiné aux cas où l'utilisateur veut choisir un fichier existant. Si l'utilisateur choisit un fichier inexistant, un menu contextuel apparaîtra l'informant que le fichier choisi n'existe pas.

.asksaveasfilename(*option* = *value* , ...)

Destiné aux cas où l'utilisateur veut créer un nouveau fichier ou remplacer un fichier existant. Si l'utilisateur choisit un fichier existant, un menu contextuel apparaîtra informant que le fichier existe déjà, et demandant s'il veut vraiment le remplacer.

Les arguments aux deux fonctions sont les mêmes :

defaultextension= *s*

L'extension de fichier par défaut, une chaîne commençant avec une période ("."). Si la réponse de l'utilisateur contient une période, cet argument n'a aucun effet. Il est ajouté à la réponse de l'utilisateur dans le cas où il n'y a aucune période.

Par exemple, si vous fournissez un argument defaultextension = ".jpg" et que l'utilisateur entre "gojiro", le nom de fichier rendu sera "gojiro.jpg".

filetypes=[(*label1* , *pattern1*), (*label2* , *pattern2*), ...]

Une liste de tuples à deux éléments contenant les types de nom de fichier et les modèles qui seront choisi pour apparaît dans la liste de fichier. Dans l'image d'écran ci-dessous, notez le menu déroulant étiqueté "Files of type:". L'argument filetypes que vous fournissez peuplera cette liste déroulante. Chaque modèle est un nom de type de fichier ("PNG" dans l'exemple) et un modèle qui choisit les fichiers d'un type donné ("(*.png)" dans l'exemple).

initialdir= *D*

Le nom de chemin du répertoire à être affiché initialement. Le répertoire par défaut est le répertoire courant de travail.

initialfile= *F*

Le nom de fichier à être affiché initialement dans le champ "File name:", s'il existe.

parent= *W*

Pour faire le menu contextuel apparaitre sur une certaine fenêtre ***W*** , fournir cet argument. Le comportement par défaut consiste en ce que le menu contextuel apparaisse sur la fenêtre racine de votre application.

title= *T*

Si spécifié, T est une chaîne affichée comme titre de la fenêtre contextuelle.

Si l'utilisateur choisit un fichier, la valeur rendue est le nom de chemin complet du fichier choisi. Si l'utilisateur utilise le bouton *Cancel*, la fonction rend une chaîne vide.

Voici un exemple :

31-3 - Le module tkColorChooser

Pour donner un menu contextuel à l'utilisateur de votre application qu'il peut utiliser pour choisir une couleur, importer le module tkColorChooser et appeler cette fonction :

```
result = tkColorChooser.askColor ( color, option=value, ...)
```

Les arguments sont :

color

La couleur initiale à être affichée. La couleur initiale par défaut est gris clair.

title= *text*

Le texte indiqué apparaît dans le secteur de titre de la fenêtre contextuelle. Le titre par défaut est "Color".

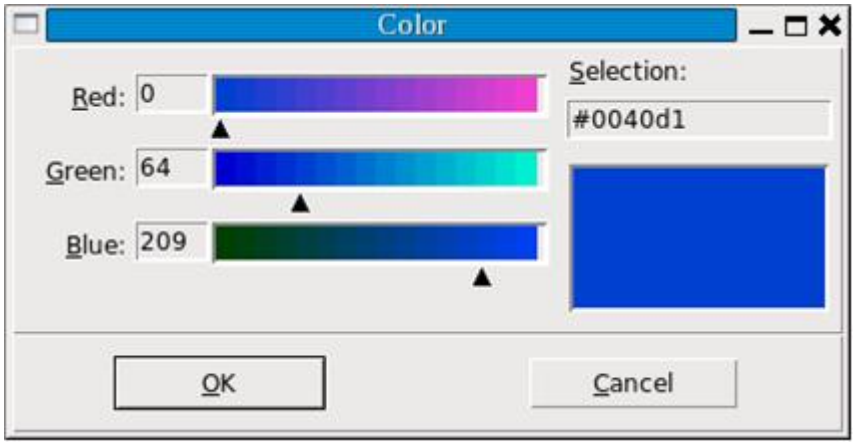
parent= *W*

Fait le menu contextuel apparaître sur la fenêtre ***W*** . Le comportement par défaut consiste en ce qu'il apparaisse sur votre fenêtre racine.

Si l'utilisateur clique bien le bouton sur le menu contextuel, la valeur rendue sera un tuple (*triple*, color), où *triple* est un tuple (R, G, B) le contenant respectivement les valeurs de rouge, vert et bleu dans la gamme [0,255] et la *color* est la couleur choisie comme l'objet régulier couleur de Tkinter.

Si l'utilisateur clique *Cancel*, cette fonction retournera (None, None).

Voici à quoi le menu contextuel ressemble sur le système de l'auteur :



1 : http://www.pythonware.com/library/tkinter/introduction/index.htm
2 : http://www.nmt.edu/tcc/help/pubs/python/web/
3 : http://flourish.org/upsidedownmap/
4 : http://www.nmt.edu/tcc/help/pubs/pil/
5 : http://docs.python.org/library/time.html
6 : http://www.tcl.tk/man/tcl8.4/TkCmd/keysyms.htm