

Table des matières

tk00 : manuel des références de tkinter.....	8
1. notion d'application graphique.....	8
1.1. les origines.....	8
1.2. les principes.....	8
1.3. programmation.....	9
1.4. le cas de Python.....	9
1.5. l'interfaçage de tkinter.....	9
2. la librairie dans l'arborescence Python.....	10
3. la documentation.....	12
4. une documentation illustrée, en français.....	13
4.1. notre objectif.....	13
4.2. en français.....	13
5. première approche.....	14
5.1. étape 1.....	14
5.2. étape 2.....	15
5.3. étape 3.....	16
6. les widgets de tkinter.....	18
6.1. liste des widgets.....	18
6.2. les fonctions des widgets.....	18
6.3. les constructeurs.....	19
6.4. le problème du nom d'une instance.....	19
tk01 : valeurs d'attributs standards.....	20
1. couleurs.....	20
1.1. couleurs nommées.....	20
1.2. couleur codées hex.....	20
1.3. méthodes de widget.....	20
2. bordures.....	21
2.1. six reliefs.....	21
2.2. codage.....	21
3. unités.....	21
4. l'attribut bitmap.....	22
5. l'attribut cursor.....	22
6. justification des textes.....	23
7. complément sur les types de données.....	23
tk02 : BitmapImage et PhotoImage.....	24
1. questions de formats.....	24
1.1. l'attribut image.....	24
1.2. modes d'une image bitmap.....	24
1.3. question de fichier.....	24
1.4. un exemple.....	25
2. la classe BitmapImage.....	26
3.1. syntaxe.....	26

3.2. les options.....	26
3.3. les méthodes.....	26
3. la classe PhotoImage.....	26
3.1. syntaxe.....	26
3.2. les options.....	26
3.3. les méthodes.....	27
4. Un problème avec l'attribut «image»	27
4.1. on considère le script suivant :.....	27
4.2. traiter le bug.....	28
tk03 : les fontes.....	30
1. descripteur de fonte.....	30
1.1. caractéristiques d'une fonte.....	30
1.2. valeur de fonte : les descripteurs de fonte.....	30
1.3. familles prédéfinies.....	31
2. Le module Font.....	31
2.1. le constructeur Font.....	31
2.2. les options de Font	31
2.3. les méthodes d'instance de Font.....	31
2.4. fonctions du module font.....	32
tk04 : géométries.....	34
1. géométrie des fenêtres.....	34
1.1. geometry.....	34
1.2. fixer la géométrie d'une fenêtre.....	34
1.3. retrouver la géométrie d'une fenêtre.....	34
2. Les gestionnaires de placement.....	35
2.1. les trois gestionnaires.....	35
2.2. règles d'usage.....	35
3. Le gestionnaire Pack.....	35
3.1. fonctionnement du gestionnaire Pack.....	35
3.2. les méthodes de widget.....	37
3.3. les valeurs d'attribut :.....	38
3.4. les options.....	38
4. le gestionnaire Grid.....	38
4.1. principe du gestionnaire Grid.....	38
4.2. les méthodes de Grid.....	39
4.3. les options de configuration de grille.....	39
4.4. les options de cellule.....	39
5. Le gestionnaire Place.....	40
5.1. principe du gestionnaire Place.....	40
5.2. les méthode du gestionnaire Place.....	40
5.3. le valeurs d'options.....	40
5.4. les options.....	40
6. la troisième dimension.....	41
6.1. empilement des «calques».....	41
6.2. Les méthodes.....	41
tk05 : attributs partagés.....	42

1. méthodes d'attributs.....	42
1.1. attributs donnés à un widget.....	42
1.2. modification d'un ou plusieurs attributs.....	42
1.3. accès aux attributs.....	42
2. attributs système.....	43
3. attributs de bordure et de marge.....	44
4. attributs de couleur.....	44
5. attributs pour les gestionnaires de géométrie (rappel).....	44
6. les attributs default et state.....	45
7. l'attribut command.....	45
8. attributs de désaffectation (disable).....	45
tk06: les méthodes partagées.....	46
1. méthodes portant sur les attributs.....	46
1.1. méthodes de configuration.....	46
1.2. méthodes d'options de classes.....	46
2. méthodes concernant les événements.....	47
3. méthodes de presse-papier.....	47
4. méthodes relatives aux placements.....	47
4.1. gestionnaires de placement.....	47
4.2. empilement des widgets.....	47
5. mesures relatives au widget.....	48
6. méthodes portant sur la console.....	48
7. méthodes de souris.....	49
8. les boucles.....	49
9. méthodes de gestion du déroulement de programme.....	51
10. méthodes d'identification.....	52
11. méthodes de focus.....	53
12. méthodes d'accaparement.....	54
tk07 : Événements.....	56
1. les événements reconnus.....	56
1.1. événements clavier.....	56
1.2. événements souris.....	56
1.3. événements fenêtre.....	56
2. la notion de séquence.....	57
2.1. écrire une séquence.....	57
2.2. exemples :.....	57
2.3. événements virtuels.....	58
2.4. le cas de la molette.....	58
3. gestionnaire d'événement.....	58
3.1. fonction gestionnaire.....	58
3.2. l'objet Event.....	59
3.3. cas mixtes.....	60
4. la liaison.....	60
4.1. les bindtags.....	60

4.2. les niveaux de liaison.....	60
4.3. le retour de bind().....	63
4.4. méthodes de suppression de liaisons.....	63
4.5. génération d'événement : event_generate().....	63
5. Les protocoles.....	64
5.1. les trois modes de liaison.....	64
5.2. la syntaxe.....	64
pour s'y retrouver dans la gestion des événements.....	64
annexe : les keysym.....	65
clavier de base.....	65
pavé numérique	66
tk08 : Tk et Toplevel.....	68
1. les constructeurs.....	68
2. les attributs.....	68
2.1. liste des attributs	68
2.2. les attributs spécifiques.....	68
3. les méthodes spécifiques.....	69
3.1. relations avec le gestionnaire de fenêtres.....	69
3.2. Icone et visibilité.....	69
3.3. Style.....	70
3.4. géométrie et position.....	70
4. Un exemple de fenêtre Toplevel.....	70
tk09 : Menu.....	73
1. le constructeur.....	73
2. les attributs.....	73
2.1. liste des attributs	73
2.2. les attributs spécifiques.....	73
3. un exemple de barres de menu.....	74
5. exemple de menu popup.....	75
6. les méthodes spécifiques.....	76
6.1. les options d'items.....	76
6.2. les méthodes d'ajouts d'items à un menu.....	77
6.3. méthodes de configuration.....	78
6.4. méthodes de popup.....	78
6.5. Divers.....	78
tk10 : Frame et LabelFrame.....	79
1. le constructeur.....	79
2. options.....	79
2.1. liste des options de Frame.....	79
2.2. liste des options de LabelFrame.....	79
2.3. attributs spécifiques.....	80
3. méthodes	80
tk11 : PanedWindow.....	81
1. le constructeur.....	81
2. options.....	82

2.1. liste des options de PanedWindows.....	82
2.2. options spécifiques :.....	82
3. configuration de panneaux	83
4. méthodes du widget PanedWindow.....	84
tk12 : Scrollbar.....	86
1. le constructeur.....	86
2. options.....	86
2.1. liste des options.....	86
2.2. attributs spécifiques.....	86
3. la fonction command.....	87
3.1. cas où le curseur de la barre de scroll est déplacée.....	87
3.2. cas où un mouvement unitaire est requis	87
4. méthodes.....	87
5. un script pour montrer le fonctionnement des connexions.....	88
5.1. la double connexion.....	88
5.2. le script commenté.....	89
tk13 : Label.....	92
1. le constructeur.....	92
2. les attributs.....	92
2.1. liste des attributs	92
2.2. les attributs spécifiques.....	92
3. les méthodes spécifiques.....	93
tk14 : Button.....	94
1. le constructeur.....	94
2. les attributs.....	94
2.1. liste des attributs.....	94
2.2. attributs spécifiques.....	94
2. méthodes	95
tk15 : Checkbutton.....	96
1. le constructeur.....	96
2. les attributs.....	96
2.1. liste des attributs.....	96
2.2. attributs spécifiques.....	96
3. méthodes	97
tk16 : Radiobutton.....	98
1. le constructeur.....	98
2. les attributs.....	98
2.1. liste des attributs.....	98
2.2. attributs spécifiques.....	98
3. méthodes	100
tk17 : Listbox.....	101
1. le constructeur.....	101
2. les attributs.....	101

2.1. liste des attributs	101
2.2. les attributs spécifiques.....	101
3. les méthodes spécifiques.....	102
4. un exemple de widget Listbox avec ascenseur.....	103
tk18 : OptionMenu.....	106
1. le constructeur.....	106
2. les attributs.....	106
2.1. liste des attributs.....	106
2.2. les attributs spécifiques.....	106
3. un exemple de widget OptionMenu.....	107
4. les méthodes.....	109
tk19 : Entry.....	110
1. le constructeur.....	110
2. les attributs.....	110
2.1. liste des attributs.....	110
2.2. les attributs spécifiques.....	110
3. les méthodes.....	112
4. scroller le widget Entry.....	114
5. problème de validation : un exemple.....	115
tk20 : Text.....	117
1. le constructeur.....	117
2. les attributs.....	117
2.1. liste des attributs.....	117
2.2. les attributs spécifiques.....	117
3. se repérer dans un texte.....	119
3.1. les index de position.....	119
3.2. les marqueurs.....	120
4. les balises (tags).....	120
4.1. fonctionnalités et aspect de blocs.....	120
4.2. la pile des tags.....	121
4.3. les option de tag.....	121
5. les méthodes.....	121
5.1. utilitaires.....	121
5.2. méthodes générales d'édition.....	122
5.3. méthodes pour les marqueurs.....	122
5.4. méthodes pour les éléments fenêtrés inclus.....	123
5.5. méthodes pour les images incluses.....	123
5.6. méthodes pour les balises.....	124
5.7. méthode de rendu.....	124
5.8. méthodes de recherche.....	125
5.9. méthodes pour les ascenseurs.....	125
6. quelques exemples.....	126
6.1. méthodes de sélection.....	126
6.2. un exemple simple.....	126

tk21 : Canvas	130
1. le constructeur.....	130
2. les attributs.....	130
2.1. liste des attributs.....	130
2.2. les attributs spécifiques.....	130
3. fonctionnement du widget.....	131
3.1. dimensions.....	131
3.2. les coordonnées.....	132
3.3. identification et taguage des items.....	134
4. méthodes générales du widget Canvas.....	135
4.1. méthodes générales.....	135
4.2. méthodes de manipulation de tags.....	135
4.3. méthodes relatives aux dimensions et aux transformations.....	136
4.4. méthodes relatives aux événements.....	136
4.5. méthodes de texte.....	136
4.5. méthodes de recherche d'item.....	137
4.6. méthodes de scroll.....	137
5. les méthodes create.....	138
5.1. création à partir d'un Bitmap ou d'une Image.....	138
5.2. création de ligne et polygone.....	139
5.3. création de rectangle, d'ellipse.....	141
5.4. création d'un arc.....	141
5.5. création de texte.....	141
5.6. création de fenêtre.....	142
tk22 : Scale	143
1. le constructeur.....	143
2. les attributs.....	144
2.1. liste des attributs.....	144
2.2. les attributs spécifiques.....	144
3. les méthodes du widget Scale.....	146
tk23 : Spinbox	147
1. le constructeur.....	147
2. les options.....	148
2.1. la liste des options.....	148
2.2. les options spécifiques.....	148
3. les méthodes du widget Spinbox.....	150

tk00 : manuel des références de tkinter

À l'origine, il y a le langage Tcl, langage de programmation simple, compilé à la volée, implanté sur toutes les plate-formes (Apple, Windows, Linux). Tcl dispose d'une librairie graphique Tk, assez riche, extensible et bien maintenue. L'appellation usuelle est Tcl/Tk. Cette librairie est facile à interfacer, et c'est ainsi que Python l'a adopté comme librairie graphique par défaut.

(voir <http://www.ensta-paristech.fr/~diam/tcl/>)

version : tout le travail a été réalisé avec **Python3.2**. sous Linux (et vérifié sous Windows) et la version (8.5) de **tkinter** qui l'accompagne.

1. notion d'application graphique

1.1. les origines

Une application graphique est une application qui est exécutée dans une unité d'interface graphique (GUI, pour **Graphic User Interface**). Les GUI les plus connus sur le PC sont Windows (système Microsoft) et X11 (système Linux). Le premier matériel qui a disposé d'un GUI sur ordinateur personnel a été Lisa (1982), d'Apple, ancêtre du Macintosh, et qui a été repris sur les plates formes Next (qui ont engendré l'actuel Mac OS X d'Apple en 2001). Les principes de l'élaboration d'interface graphique et des logiciels graphiques ont été définis entre 1970 et 1981 au **Palo Alto Research Center**.

L'interface graphique utilise la métaphore du bureau : des pages empilées ou se chevauchant ; des piles de documents (pages) distribuées sur un bureau ; le dessus d'une des piles est la page active, page sur laquelle on travaille ; les pages peuvent être réunies en dossier ou en systèmes à onglets (*notebook*) ; des systèmes de classement par listes, répertoires etc. Ce type de métaphore qui paraît si naturelle aujourd'hui n'est cependant pas si évident qu'il y paraît. Les tablettes filent une autre métaphore, plus proche de celle de livres issus d'une bibliothèque : on tourne les pages, on glisse d'une partie du document à une autre, on choisit son application dans une bibliothèque d'applications indépendantes...

1.2. les principes

* Un logiciel graphique se présente sous forme de **composants graphiques fenêtrés**, et il dispose d'un **outil de pointage** (la souris) qui permet assez souvent de ne pas utiliser la commande par le clavier.

* Par ailleurs, la gestion du logiciel se fait par **événements**, et non selon un déroulement séquentiel impératif comme dans les logiciels classiques.

les composants fenêtrés

* Les **composants fenêtrés** peuvent être de divers aspects et fonctions : **cadres, menus, boutons, images, étiquettes, tableaux** etc. Chaque élément fenêtré, est un objet au sens de la POO, avec ses paramètres, ses attributs et ses méthodes.

* Les éléments fenêtrés affichés sont **hiérarchisés** et forment un arbre ; à la racine, on trouve la **fenêtre principale (root window)**. Les divers éléments entretiennent un rapport de dépendance du type maître (**master**) à enfant (**child**). La dépendance se traduit graphiquement par l'inclusion de l'enfant dans son maître, sauf pour les fenêtres de haut niveau. Attention, ce rapport de dépendance ne doit pas être confondu, malgré la proximité de vocabulaire avec la hiérarchie des objets (rapport hiérarchique qu'il vaut mieux exprimer en français par les mots **ascendant** et **descendant**). Ce rapport de dépendance est identifiable à un rapport de contenant à contenu sauf pour les éléments de haut niveau (les fenêtres **TopLevel**) : tout élément fenêtré a un **conteneur** unique, Si le maître est modifié ou s'il disparaît, toute sa progéniture est modifiée (ou disparaît). Supprimer la fenêtre principale revient à supprimer la présence à l'écran du logiciel, et en pratique sa fin.

* La gestion est **événementielle** : cela implique qu'un dispositif de surveillance scrute en permanence toutes les sources d'événements possibles : frappe d'une touche au clavier, action d'un timer, actions liées à la souris, au programme, à des éléments d'interface (comme la modification d'une dimension de fenêtre)... L'élément racine dispose d'une boucle qui surveille tout ce qui relève de l'application.

Sortir de cette boucle s'accompagne immédiatement de la suppression de la réactivité aux événements. Dans les logiciels de «console», il existe au mieux un élément qui est en attente d'un événement (en général, la frappe dite de «validation» venant du clavier). Dans les logiciels graphique, ce sont tous les éléments qui peuvent être en «en attente» d'un événement.

* Liée à cette notion de gestion événementielle on trouve celle de **focus** : une seule fenêtre de haut niveau à la fois peut être en fonctionnement ; on dit qu'elle est active. Parmi ses contenus , un élément (ou lui-même) peut attendre un événement venant du clavier : on dit qu'il a le focus. Le système d'exploitation et l'interface graphique gèrent les fenêtres de haut niveau et n'accordent l'activité qu'à l'une d'entre elles : c'est la fenêtre active, qui se repère par une teinte différente donnée à la barre de titre de la fenêtre.

1.3. programmation

La notion d'interface graphique fenêtrée et celle de programmation objet naissent en même temps. Comme ce mode de modélisation se prête bien à la programmation de l'interface graphique, l'essentiel des langages de la programmation graphique relève de la POO. Pour mémoire, Smalltalk, C++, Delphi (Pascal), Visual Basic, Java. Les systèmes qui implémentent l'interface graphique fenêtrée comportent deux parties, le système d'exploitation proprement dit, et une surcouche graphique capable de commander la carte graphique. C'est cette surcouche, Windows (Microsoft), X11 (système Linux) que le programme commande. Depuis les années 1990, on voit apparaître des modules complémentaires (*frameworks*) que l'on peut interfacer aussi bien avec les langages compilés que les langages interprétés et indépendants des langages destination

1.4. le cas de Python.

Python comporte plusieurs version : CPython, le Python classique, Jython, écrit en Java, IronPython etc. Jython qui fonctionne sous Java peut utiliser les capacités graphiques de Java (awt, swing).

L'activité foisonnante des programmeurs Python a permis de proposer au moins quatre bibliothèques graphiques d'origine externe, dont voici une évocation rapide :

librairie	wrapper	commentaire
tk	tkinter	Écrit en Tcl. Comporte tous les essentiels. Inclus le système Python. Utilisé aussi par Perl, Ruby. Existe en version Python 3. Exploite peu le système hôte.
wxWidget	wxPython	Bibliothèque plus riche que tkinter . Exploite au maximum le système hôte (Windows, Linux...). Les applications on le look du système hôte.
Qt	PyQt, Pyside	Développé en C++. À la base de KDE sous Linux. C'est la cour de grands ! Programmation assez spécifique.
Gtk	PyGtk	Comparable à tk ou wxWidget . Peut être programmé avec le logiciel Glade (dont il existe une version pour wxPython).

tkinter n'est pas le plus puissant ni me plus beau. Mais il a l'avantage d'être présent dans le système Python, ce qui aide à la diffusion des programmes. Il est aussi le plus simple, et son intérêt pédagogique est évident.

1.5. l'interfaçage de tkinter.

Pour aider à comprendre comment se situe la librairie tkinter dans le complexe logiciel, on propose le schéma qui suit.

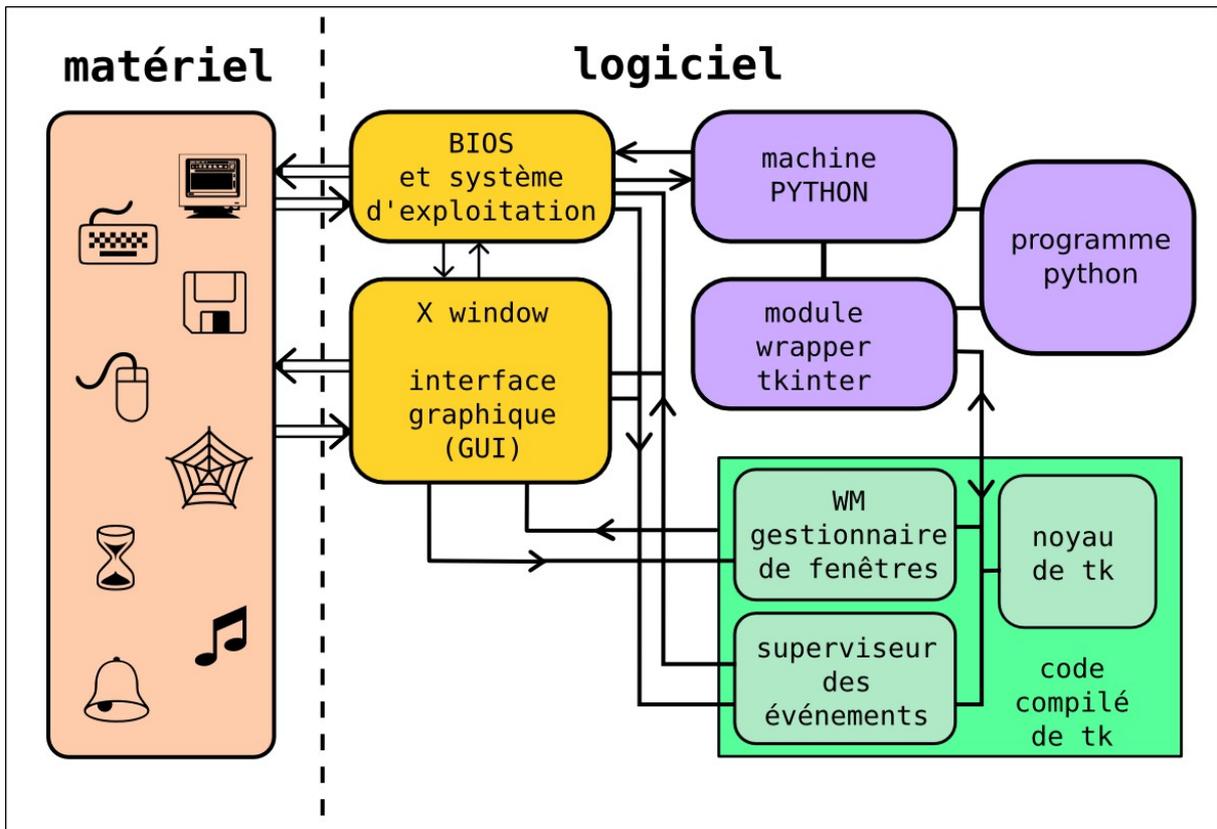
Parmi les choses importantes, il faut signaler la structure tripartite de tkinter :

- le cœur de **tkinter** comprend l'essentiel des routines ; ces routines ont un fonctionnement très classique puisque une fois appelées, elles effectuent leur tâche jusqu'au bout avant de rendre la main.
- le superviseur des événements : cette partie comporte une surveillance des événements, et

les empile. Le traitement des événements est donc différé et certains événements sont simplement oubliés. Il faut penser aussi que dans les système à un seul thread, si une boucle logicielle est engagée, le système de surveillance est interrompu. Au cours d'un dessin d'animation dans un canevas, pas question d'activer un bouton ; on y parvient cependant en multithread pour lequel Python est bien équipé.

le gestionnaire de fenêtre est chargé du calcul des éléments d'affichage et de la commande de l'interface graphique du système. Ce gestionnaire fonctionne sur le mode paresseux (*idle*). **Il n'effectue les calculs demandés que lorsqu'il n'y a rien de plus urgent à faire.**

Le cas d'espèce consiste à demander à ce gestionnaire d'effectuer une tâche complexe, utilisant d'autres éléments graphiques déjà programmé. Pour que cela fonctionne, il faut être sûr qu'il n'y a pas de tâches en attente dont on désire exploiter le résultat : par exemple, si on demande à une fenêtre d'être modale sur une fenêtre hôte, il faut être sûr que celle-ci soit calculée complètement -peu importe qu'elle soit effectivement affichée- On peut sinon avoir un comportement partiellement modal ! bonne réaction à l'iconisation, mauvais affichage-.



2. la librairie dans l'arborescence Python

tkinter est un wrapper qui se situe dans la librairie de Python. C'est un module répertoire.

Sur notre machine, le chemin est : `/usr/lib64/python3.2/tkinter`

La librairie `python` peut se situer dans une autre arborescence ; cela dépend de l'implémentation de Python dans le système d'exploitation.

Le répertoire se présente ainsi :

```
jean@mse:/usr/lib64/python3.2/tkinter$ ls -ls
total 380
-rw-r--r-- 1 root root 155054 2011-09-05 23:30 __init__.py
```

```

-rw-r--r-- 1 root root 77690 2011-09-05 23:30 tix.py
-rw-r--r-- 1 root root 56250 2011-09-05 23:30 ttk.py
-rw-r--r-- 1 root root 14552 2011-09-05 23:30 filedialog.py
-rw-r--r-- 1 root root 11488 2011-09-05 23:30 dnd.py
-rw-r--r-- 1 root root 11395 2011-09-05 23:30 simplifiedialog.py
-rw-r--r-- 1 root root 6135 2011-09-05 23:30 font.py
drwxr-xr-x 2 root root 4096 2012-12-26 17:25 __pycache__
-rw-r--r-- 1 root root 3701 2011-09-05 23:30 messagebox.py
-rw-r--r-- 1 root root 2887 2011-09-05 23:30 _fix.py
-rw-r--r-- 1 root root 1814 2011-09-05 23:30 scrolledtext.py
-rw-r--r-- 1 root root 1793 2011-09-05 23:30 colorchooser.py
-rw-r--r-- 1 root root 1568 2011-09-05 23:30 dialog.py
-rw-r--r-- 1 root root 1493 2011-09-05 23:30 constants.py
-rw-r--r-- 1 root root 1412 2011-09-05 23:30 commondialog.py
-rw-r--r-- 1 root root 148 2011-09-05 23:30 __main__.py

```

* `__init__.py`

Le répertoire est un module car il possède le présent fichier. Lors de l'importation du module, le fichier `__init__.py` est exécuté, et les classes, fonctions et constantes qu'il définit sont importées.

On rappelle qu'il y a deux grandes façons pour importer un module :

```

- from tkinter import <données à importer>
- import tkinter

```

Dans le premier cas, les données à importer sont citées sous forme d'une liste de noms séparés par des virgules :

```

from tkinter import Tk, Toplevel, Label, Button

```

ou si l'on veut tout importer :

```

from tkinter import *

```

Dans ce cas, les données à importer sont utilisées sans préfixe et seules les données importées sont connues du script d'importation :

```

monBouton = Button(monParent, text="QUITTER")
fenetrePrincipale = Tk()

```

Dans la seconde approche, `import tkinter`, seule la référence du module est connue ; pour disposer des éléments qu'il connaît, c'est-à-dire ceux définis dans la fonction `__init__()`, il suffit de les utiliser comme qualificatifs :

```

monBouton = tkinter.Button(monParent, text="QUITTER")
fenetrePrincipale = tkinter.Tk()

```

Attention : si on fait `from tkinter import *`, on ne connaît que les éléments de `__init__`, pas ceux des fichiers du répertoire.

On peut cependant importer un module inclus par les deux méthodes :

```

from tkinter import messagebox
from tkinter.messagebox import askyesno
import tkinter.messagebox as popMsg
import tkinter.messagebox

```

* `tix.py`

Tix Tk Interface eXtension est une extension de **tkinter** qui offre des composants de plus haut niveau (plus de 40) comme `ComboBox`, `NoteBook` (onglets), `DirTree`, `FileSelectBox`.

Il est en principe inclus dans toutes les éditions de Python.

* **ttk.py**

Redéfinit des composants de **tkinter** : **Button**, **Checkbutton**, **Entry**, **Frame**, **Label**, **LabelFrame**, **Menubutton**, **PanedWindow**, **Radiobutton**, **Scale** et **Scrollbar** en améliorant leur aspects Il en ajoute 6 qui sont : **Combobox**, **Notebook**, **Progressbar**, **Separator**, **Sizegrip** et **Treeview**.

Cette librairie n'existe que sur les versions récentes.

* **filedialog.py**

Librairie intéressante puisqu'elle permet de disposer d'un composant graphique pour choisir des fichiers. On rappelle que si l'on veut utiliser les éléments de cette librairie (comme les autres du répertoire), il faut les importer explicitement :

```
from tkinter.filedialog import *
```

* **simpledialog.py**

Librairie qui fournit des fonctions qui permet la saisie clavier à travers une interface graphique : **askinteger**, **askfloat**, **askstring**

* **dnd.py**

Librairie de drag & drop

* **font.py**

Une librairie importante si l'on veut travailler finement sur les fontes (rechercher les fontes disponibles par exemple). Elle complète des dispositions existant dans le cœur de **tkinter**, et n'est pas utile si on se contente de définir des attributs de fonte pour les composants **Label**, **Button** etc.

* **messagebox.py**

La librairie la plus communément utilisée pour les petits travaux d'interrogation par popup à travers les fonctions comme **askyesno**, **askokcancel**, **askquestion...**

* **_fix.py**

Actuellement non utilisable

* **scrolledtext.py**

Affichage et saisie de texte avec un ascenseur latéral

* **colorchooser.py**

Permet de choisir une couleur sur une palette.

* **constants.py**

Cette librairie est à usage interne : comme elle est importée dans **__init__** ses données sont disponibles depuis le module **tkinter**.

* **commondialog.py** , **dialog.py**

À usage interne.

3. la documentation.

La documentation en français est restreinte. Les tutoriels traitent tous plus ou moins des mêmes sujets et avec un bonheur à relativiser. La documentation de référence n'existe en pratique qu'en anglais. On peut pour des cas d'espèce consulter la documentation **Tcl/Tk**, partiellement traduite en français (<http://wfr.tcl.tk/>)

Le passage obligé pour **Python** est "**An Introduction to Tkinter**" de **Frederik Lundth**. Cette documentation date de 1999, et nécessite quelques adaptations ; une mise à jour est annoncée depuis 2006 ! On trouve en <http://effbot.org/tkinterbook/> la version la plus récente, qui reste incomplète. Cette documentation reste la plus sérieuse malgré de nombreuses inexactitudes et obsolescences. Elle est disponible sur le web aux formats **html (version récente)** et **pdf (vieille version)**.

Il existe une documentation de John W. Shipman pour les étudiants de l'Université du Nouveau Mexique. Elle répond bien à son objet -une référence pour des étudiants-, elle est fiable mais reste

lacunaire. S'en tenir à la version originale, en anglais.

Bien entendu, il reste la possibilité d'utiliser la documentation interne, engendrée par **pydoc3**.

Le plus simple est de lancer dans une console :

```
pydoc3 -w tkinter
pydoc3 -w tkinter.tix
```

Pydoc crée alors la doc au format HTML dans le répertoire courant. C'est lourd, assez indigeste, mais précieux pour disposer de références de première main. On peut toujours faire du copier/coller de certaine parties dans Open Office. En veillant toutefois à éviter de trop grosses transpositions qui sont fort longues et hasardeuses.

On peut évidemment faire la même opération avec les modules additionnels évoqués dans la section qui précède. C'est même en pratique la seule documentation qui existe pour plusieurs modules. Pour le paramétrage de pydoc3, ouvrir une console (sous Linux) et faire :

```
man pydoc3
```

Une doc **Python/tkinter** est elle aussi disponible dans la doc Python, mais elle s'avère assez peu pratique.

4. une documentation illustrée, en français.

4.1. notre objectif

L'objectif est de réaliser un document des références de **tkinter**, aussi exhaustif que possible, mais pratique pour le programmeur non spécialisé (à l'origine, rédigée pour un atelier de club informatique).

Si on compare à une approche telle que celle de Lundh, nous inclurons davantage d'explication, d'exemples, nous évoquerons les pièges ou problèmes rencontrés, quitte à être par ailleurs moins systématique. L'étude d'un composant par exemple pourra comporter plusieurs renvois, alors que la documentation traditionnelle essaie souvent, au prix de nombreuses redites, de constituer un dossier complet pour chaque composant.

4.2. en français...

L'adaptation française d'une référence pose le problèmes du vocabulaire à employer.

Certaines appellations française sont sans problèmes, comme **module**, **importer**, **variable**, **fonction**, **méthode**, **classe**, **canevas**, **option**. Ils sont un équivalent du mot anglais, avec lequel ils partagent l'origine linguistique. Mais assez souvent, plutôt qu'un mot aussi général que **option**, on préfère attribut, qui n'a pas la même connotation de *donnée facultative*; les options sont en effet des attributs qui peuvent avoir une affectation par défaut, et non des attributs qui peuvent rester indéterminés.

Pour le mot **font** on utilise **fonte** plutôt que **police** de caractères. Le mot **texte** traduisant **text** désigne une séquence de caractères.

Certaines traductions ne devraient pas faire de difficulté : **manager** devient **gestionnaire**, **event** devient **événement**. Mais un mot comme **handler** se traduit mal. Ainsi **event handler** devient plus clairement **un gestionnaire d'événement**. Même chose pour **slider**, que nous conservons en l'état.

Les méthodes **geometry()**, **pak()**, **grid()**, **place()** constituent **les gestionnaires de géométrie**.

Le problème est plus délicat pour **main window**, **oplevel window**, **widget**. Ces composants des logiciels graphiques constituent les **composants graphiques fenêtrés** évoqués ci-dessus. Pour les deux premiers la transposition française en **fenêtre principale (racine)** et **fenêtre de haut niveau** ne devraient pas poser de difficultés ; mais **fenêtre Toplevel** est bien pratique. Quant à **widget**, mot dérivé du mot d'argot américain **gadget**, il est suffisamment pratique et évocateur qu'il vaut mieux le garder.

Les mots **command** et **control** sont plus délicats à apparier ; **command** désignera toujours une fonction.

Le mot **mark** se traduit usuellement par marque. Mais étant donné la polysémie de *marque* en français, dans le domaine scientifique on préfère **marqueur** : un marqueur radioactif, la fluorescéine en tant que marqueur. Non utiliserons donc **marqueur**.

Le mot **tag** est souvent adapté en informatique comme **balise** ; par exemple en html. C'est une bonne

approche, d'autant que le mot étiquette parfois employé est préférable pour la traduction de **label**. Cependant on peut également garder **tag**, en précisant.

5. première approche.

À travers cet exemple, on va montrer quelle est la structure basique de la programmation d'un logiciel graphique avec **tkinter**. Cela permettra de **fixer le vocabulaire** pour la suite du documents.

5.1. étape 1

La première étape consiste à :

- importer la librairie **tkinter**
- créer une fenêtre principale
- lancer la méthode **mainloop**.

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *

# fenêtre principale
fenPrincipale = Tk()

# boucle de surveillance des événements
fenPrincipale.mainloop()

#
# fichier tk00ex00.py
```

discussion

* la première ligne est un **shebang**. Un **shebang** indique dans un environnement UNIX que ce qui suit est un texte interprétable appelé aussi un **script**. Il précise le mode d'accès à l'interpréteur.

Même s'il ne sert pas, il rappelle toujours quel interpréteur doit être utilisé (ici python version 3). On rappelle que l'on peut rendre un script directement exécutable sous UNIX grâce au shebang, en mettant les droits du fichier à «exécutable». Sous Windows, le shebang ne sert pas ; mais on rend exécutable en mettant l'extension du fichier à **pyw** et en réglant le gestionnaire de fichiers.

* la deuxième ligne ne sert à rien en Python3, puisque par défaut, le logiciel utilise l'UNICODE et que les sauvegardes sont en mode **UTF-8**. Elle est cependant présente «pour mémoire», et rappeler de bien surveiller les paramètres de l'éditeur de travail.

* L'**importation du module** permet de disposer de tous les éléments disponibles dans **tkinter** et de les utiliser sans qualificatif. On peut discuter de ce mode d'importation. Il convient de formuler les remarques suivantes :

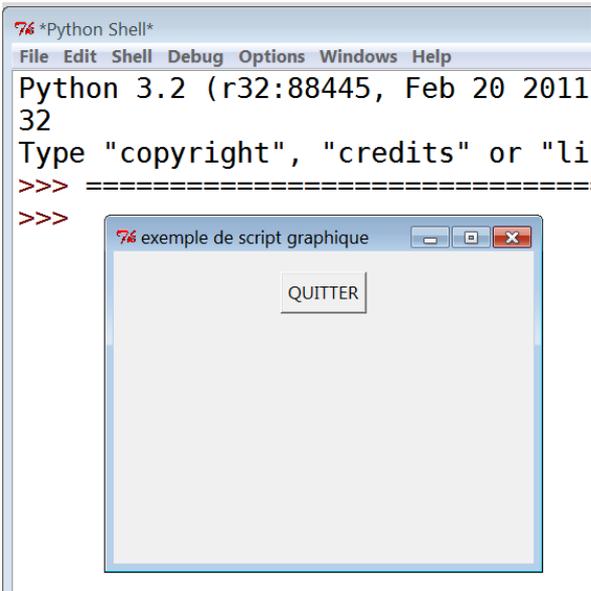
- si on importe une librairie quelconque après cette première importation, les éléments nouvellement importés occultent les anciens de même nom. Il n'y a donc pas de problème pour importer **ttk** sur le même mode, puisque c'est l'effet recherché.
- il peut cependant y avoir un hiatus avec une librairie comme **pil** (traitement d'images), qui contient par exemple un composant **Image**, homonyme d'un composant de **tkinter**, et avec des fonctionnalités qui sont différentes. Une règle pratique de conduite est d'importer les librairies externes en utilisant l'importation nominale ou la qualification, et éventuellement un alias :

```
from pil import Image as Img # alias
import pil # qualification
```

exécution du script

L'exécution du script sous Linux se fait à partir de la console.

Note : Dans tout le document, on ne fait pas apparaître la ligne de commande d'exécution ; le nom du fichier exemple apparaît en dernière ligne du script.

	<p>Lancement du script : La fenêtre graphique apparaît, avec un dimensionnement par défaut. Noter la présence des trois icônes usuelles dans la barre de titre ainsi que le titre par défaut tk.</p> <p>En fond, la console :</p>
---	---

5.2. étape 2

La seconde étape consiste à doter le logiciel de trois fonctionnalités importantes :

- le réglage de la dimension et de la position de la fenêtre
- la possibilité de sortir du logiciel autrement qu'un invoquant l'icône de sortie, mais un bouton appartenant à l'interface.

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *

# fenêtre principale
fenPrincipale = Tk()
fenPrincipale.title("exemple de script graphique")

# dimensionnement
fenPrincipale.geometry ("400x300+150+100")

# quitter le logiciel
def quitter():
    fenPrincipale.quit()

# bouton quitter
btQuitter = Button(fenPrincipale, text="QUITTER", command=quitter)
btQuitter.pack (pady=20)

# boucle de surveillance des événements
```

```
fenPrincipale.mainloop()
#
# on mettrait ici les actions à exécuter en fin de script
#
# fichier tk00ex01.py
```

commentaires

```
* fenPrincipale.title()
```

On a donné la possibilité de personnaliser le titre

```
* fenPrincipale.geometry ("300x200+150+100")
```

Les fonctions de placement sont désignées sous l'appellation **géométrie (geometry)**

La géométrie d'une fenêtre se décline comme une chaîne :

largeur x hauteur + décalage horizontal sur l'écran + décalage vertical sur l'écran

Ce gestionnaire de géométrie n'est pas impératif en ce qui concerne les dimensions : si un gestionnaire vient contrecarrer le dimensionnement, il prend le pas sur la définition de géométrie initiale. C'est particulièrement vrai avec les gestionnaires de géométrie `pack()` et `grid()`.

```
* def quitter():
```

Cette fonction est un **gestionnaire d'événement** c'est-à-dire qu'elle est appelée lorsqu'un événement se produit, ici lorsque le bouton est cliqué.

```
    fenPrincipale.quit()
```

La fonction est obligée au retour de la méthode bloquante `mainloop`.

```
* btQuitter = Button(fenPrincipale, text="QUITTER", command=quitter)
```

On ici la création d'un **composant graphique fenêtré** en occurrence un bouton. Noter le premier argument qui est le **maître (master, parent, propriétaire, conteneur)**. Ce qui suit est la liste des attributs, facultatifs puisque ce sont des paramètres initialisés. Comme le nombre est indéfini, on utilise la notation clef/valeur avec **égale**. L'attribut `command` a comme valeur la fonction gestionnaire qui est appelé quand on clique le bouton.

```
* btQuitter.pack(pady=20)
```

Dans **tkinter**, à la différence d'autres logiciels graphiques, les données de placement ne sont pas des attributs du composant graphique fenêtré. Mais celui-ci comporte des méthodes, appelées **gestionnaires de géométrie** et nommées **pack()**, **grid()** et **place()**. L'un des trois doit être invoqué pour que l'objet bouton soit affiché ; le mode de placement dépend du gestionnaire de géométrie invoqué, ici, **pack()**.

dificultés

* Dans un environnement normal (appel en ligne de commande), la fin du script entraîne la destruction de l'application. Mais sur certains environnements (sous **Idle** par exemple), il faut explicitement détruire l'application.

* Dans la pratique, on met souvent un avertissement avant la sortie d'un logiciel, pour éviter les sorties en cas de manœuvre maladroite. Il est logique de placer cet avertissement dans la fonction `quitter()`. Mais ceci n'évite pas une utilisation intempestive de l'icône de barre de titre. Il est possible de capturer la clic sur cette icône et de l'identifier à l'activation du bouton, ce qui est fait ici ; on peut aussi l'inhiber.

5.3 étape 3

La présente étape donne le script minimal pour créer une application **tkinter**. On peut ensuite broder autour du thème proposé, par exemple utiliser un autre procédé de contrôle.

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *
```

```

from tkinter.messagebox import askyesno

# fenêtre principale
fenPrincipale = Tk()
fenPrincipale.title("exemple de script graphique")

# dimensionnement
fenPrincipale.geometry ("400x300+150+100")

# quitter la boucle principale
def quitter():
    reponse = askyesno("terminer le script",
        "Voulez-vous réellement terminer\u00a0? \n cliquer «oui» pour finir")
    if reponse :
        fenPrincipale.quit()

# bouton quitter
btQuitter = Button(fenPrincipale, text="QUITTER", command=quitter)
btQuitter.pack (pady=20)

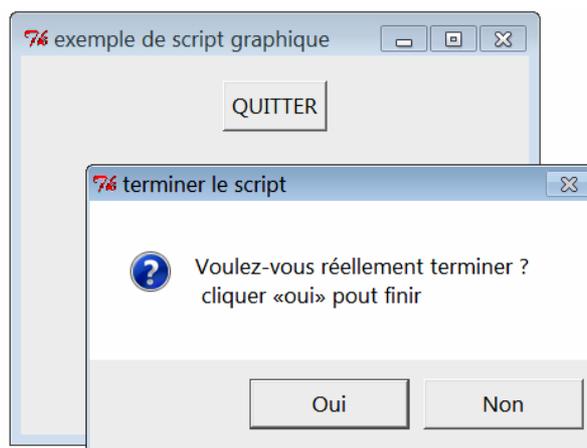
# capture de l'icone de la barre de titre
fenPrincipale.protocol("WM_DELETE_WINDOW", quitter)

# boucle de surveillance des événements
fenPrincipale.mainloop()
#
# on mettrait ici les actions à exécuter en fin de script
#
fenPrincipale.destroy() # par précaution

# fichier tk00ex02.py

```

exécution



6. les widgets de tkinter

6.1. liste des widgets

Un widget est l'instance d'une classe. Voici les classes de widgets du **tkinter** basique dans l'ordre alphabétique. Les modules permettent d'ajouter d'autres classes de widgets.

widgets de haut niveau : les fenêtres de haut niveau **Tk**, **Toplevel**, le widget **Menu**.

Les autres widgets de la version 8.5. : **Frame**, **LabelFrame**, **PanedWindow**, **Scrollbar**, **Label**, **Button**, **Checkbutton**, **Listbox**, **Radiobutton**, **Entry**, **OptionMenu**, **Text**, **Canvas**, **Scale**, **Spinbox**.

Les widgets obsolètes : **Message** (remplacé par **Label**) et **Menubutton** (remplacées par **OptionMenu**)

6.2. les fonctions des widgets

1. **Button** chapitre : 14

Le widget est un bouton avec un intitulé. On peut le cliquer par la souris pour activer un gestionnaire d'événement (chapitre 6)

2. **Canvas** chapitre : 21

Le widget présente un canevas, surface sur laquelle on peut dessiner lignes, textes ou images.

3. **Checkbutton** chapitre : 15

Il s'agit de la case à cocher classique. Son changement d'état peut activer un gestionnaire d'événement (chapitre 7)

4. **Entry** chapitre : 19

Le widget **Entry** permet de saisir une ligne de texte au clavier. Pour cela, il doit avoir le focus. On peut lui adjoindre une base de scroll.

5. **Frame** chapitre : 10

Le widget **Frame** est le conteneur universel.

6. **Label** chapitre : 13

Le widget **Label** permet de créer un étiquette. Le texte de l'étiquette peut avoir plusieurs lignes. Il peut également afficher un bitmap ou une image.

7. **LabelFrame** chapitre : 10

Le widget **LabelFrame** permet de créer un cadre étiqueté sur sa bordure

8. **Listbox** chapitre : 17

9. **Menu** : 09

Le widget **Menu** sert à créer des barres de menu, les menus en cascade associés, les menus popup.

9. **OptionMenu** chapitre : 18

Le widget **OptionMenu** est un widget pratique pour créer des menus insérés n'importe où dans le logiciel ; c'est ce que l'on appelle communément un **Combobox**.

10. **PanedWindow** chapitre : 11

Le widget permet de gérer les fenêtres à panneaux, où les panneaux sont séparés par de échancrures dont le placement est manipulable par l'utilisateur.

11. **Radiobutton** chapitre : 16

Les boutons radio fonctionnent par groupe : un seul dans une famille est actif à la fois. Le changement d'état d'un bouton peut également provoquer l'exécution d'un gestionnaire d'événement.

6.12. **Scale** chapitre : 22

Le widget **Scale** permet de disposer un curseur, en vue de la saisie graphique d'une valeur numérique sur une échelle donnée.

6.13. **Scrollbar** chapitre : 12

Le widget **Scrollbar** définit un ascenseur, vertical ou horizontal pour certains widget comme **frame**, **entry**, **canvas** etc.

6.14. **Spinbox** chapitre : 23

Le widget Spinbox comporte une zone de saisie pour un nombre entier sur une échelle donnée et deux petits boutons permettant l'évolution des valeurs de l'échelle, croissant ou décroissant.

6.15. Text chapitre : 20

Le widget Text permet une saisie et un affichage de texte multiligne. Ce widget complexe autorise le formatage de texte, l'insertion d'images, de canevas, la sélection de zones de texte etc.

6.16. Tk chapitre : 08

Ce widget particulier n'a qu'une instance, la **fenêtre principale** (fenêtre racine */root window*).

6.17. Toplevel chapitre : 08

Ce widget définit un fenêtre de haut niveau dans une application ayant déjà sa fenêtre principale.

6.3. les constructeurs

Le constructeur de **Tk** sera abordé dans l'étude de ce composant.

Les autres constructeurs sont définis par la méthodes suivante :

```
def __init__(self, master=None, cnf={}, **kw):
```

Ce qui se traduit en pratique par :

```
widget = Constructeur (master=None, options_clef_valeur)
```

* **self** : l'instance définie

* **master** : le conteneur (maître, conteneur, *parent*) ; **attention** : **master** est aussi un champ du widget, qui a la même valeur que la paramètre passé, sauf s'il n'est pas présent, auquel cas, c'est la fenêtre principale..

* **cnf**={ } : à usage interne ;

* ****kw** : épuise, en nombre indéfini les paramètres d'appel à mot clef, autres que, éventuellement, **master** et **cnf**.

Si le conteneur n'est pas défini lors de l'appel, le conteneur par défaut est la fenêtre principale. Sauf pour **Toplevel**, il faut toujours spécifier le conteneur.

Le paramètre **cnf** est à usage interne .

Les paramètres réels déclarés avec un mot clef sont ce que l'on appelle **les options du constructeur**. Pour chaque widget, il y a une **liste de clefs définies** comme clefs d'option, chacune étant associée à **une valeur d'un type et d'une structure bien précise** (chapitres 1 et 2)

6.4. le problème du nom d'une instance.

Chaque instance est nommée automatiquement lors de sa construction. Le nom est un identificateur, c'est-à-dire qu'il est impératif qu'il soit unique dans une application. Il est rare que l'on ait besoin du nom de l'instance en programmation, encore moins de devoir donner un nom autre que celui donné par la machine **Python/tkinter**. Les règles suivantes sont alors à appliquer :

- on peut imposer un nom (chaîne de caractères alpha numériques, avec le souligné) lors de la construction par le paramètre valeur initialisé **name="identificateur"** .
- la fenêtre principale est nommée **tk** ; le constructeur **Tk()** ne supporte pas le paramètre **name**.
- l'identificateur doit être unique
- le nom d'un widget est accessible par la méthode **winfo_name()** ; il n'existe aucun moyen de changer le nom d'une instance existante.

tk01 : valeurs d'attributs standards

Les widgets possèdent des attributs qui portent sur

- les couleurs (de fond ou de texte) ,
- les bordures (largeur, style),
- les caractères (fontes, taille, épaisseur, inclinaison) ou
- la taille des constituants.

Les valeurs de ces attributs ont une structure qui est l'objet de ce chapitre. Pour l'attribut image voir les chapitres sur les classes BitmapImage et PhotoImage

1. couleurs

Les attributs dont la valeur est une couleur attendent un codage de cette couleur sous forme d'une chaîne de caractères.

1.1. couleurs nommées

Certaines couleurs sont nommées ; il existe 8 valeurs sûres, que l'on retrouve dans toutes les implémentations :

"white", "black", "red", "green", "blue", "cyan", "yellow" , "magenta"

D'autres couleurs sont reconnues, qui peuvent dépendre des plate-formes :

"maroon", "navy", "purple", "grey", "light grey"

```
fenPrincipale.config(bg="purple")
```

On peut écrire aussi bien en majuscules qu'en minuscules ; "RED", "Red", "ReD" sont acceptés.

1.2. couleur codées hex

Il y a trois modes possibles pour l'encodage :

* le mode 4 bits par couleur

Le schéma est "#RGB", où R, G, B sont des **chiffres** hexadécimaux, correspondant aux composantes rouge, verte et bleue de la couleur dans une représentation RGB. On peut coder une palette de 4096 nuances. La casse n'est pas significative.

```
fenPrincipale.config(background="#f8f") # un pourpre
```

* le mode 8 bits par couleur

Le schéma est "#RRGGBB", où R, G, B sont des chiffres hexadécimaux et RR, GG, BB correspondent aux composantes rouge, verte et bleue de la couleur dans une représentation RGB. On peut coder une palette de 16777216 nuances. La casse n'est pas significative.

```
btQuitter = Button(fenPr, text="QUITTER", foreground="#FF4040" )
```

* le mode 12 bits par couleur (16 bits)

Le schéma est identique. L'encodage est similaire aux précédents, mais les composantes sont sur 3 chiffres (12 bits). Un mode 16 bits est utilisé en interne pour coder les couleurs.

1.3. méthodes de widget

* `winfo_rgb(nom_couleur)` : cette méthode est une méthode de widget ; elle retourne un tuple des valeurs décimales des composantes RGB de `nom_couleur`. On rappelle qu'en interne, les couleurs sont codées sur 48 bits.

exemple :

```
print (self.winfo_rgb(boutonNon.cget("highlightcolor")))
(65535, 0, 0)
```

* `winfo_depth()` : retourne le nombre de bits par pixels utilisés dans l'application.

2. bordures

Tous les composants fenêtrés ont une bordure. La bordure est caractérisée par son épaisseur (`borderwidth`) et son relief (`relief`). L'épaisseur relève des mesures (section suivante).

2.1. six reliefs



exemple :

```
btQuitter.config(borderwidth=12, relief=RAISED)
```

2.2. codage

Un relief est désigné par une chaîne de caractères, mais `tkinter` a défini des constantes :

`RAISED='raised'`

`SUNKEN='sunken'`

`FLAT='flat'`

`RIDGE='ridge'`

`GROOVE='groove'`

`SOLID='solid'` : bordure régulière

3. unités

Les unités peuvent faire problème pour le programmeur `tkinter`. Il faut se rappeler la règle sur les descripteurs de fonte : un entier positif a comme unité le point et un entier négatif le pixel. Dans les instances de `Button` et `Label`, l'unité pour `width` est le quadratin (largeur type de caractère pour le descripteur donné). En général cependant, les mesures littérales peuvent être dotées d'unités ou non :

	La valeur est en entier ; il n'y a pas d'unité explicite : c'est le pixel
<code>c</code>	L'unité est le centimètre. La valeur doit être une chaîne : "3c"
<code>i</code>	L'unité est le pouce (inche). La valeur doit être une chaîne : "12i"
<code>m</code>	L'unité est le millimètre. La valeur doit être une chaîne : "3m"
<code>p</code>	L'unité est le point (1/72 de pouces). La valeur doit être une chaîne : "120p"
<code>??</code>	Avec certains widgets (<code>PanedWindow</code>) les valeurs de dimension sont rendues avec une unité interne à <code>tkinter</code> . On doit alors utiliser les méthodes ci-dessous.

Les grandeurs sont immédiatement traduites en pixels et l'utilisation d'unités explicites rend dépendant de l'écran utilisé.

```
monCadre = Frame(fenPrincipale, width="8i", height="5c")
```

Pour voir comment se comporte un écran par rapport aux dimensions, il faut utiliser une méthode de widget.

* `winfo_fpixels (dimension)` : transforme la dimension en pixels sans arrondi

* `winfo_pixels (dimension)` : arrondit le résultat précédent

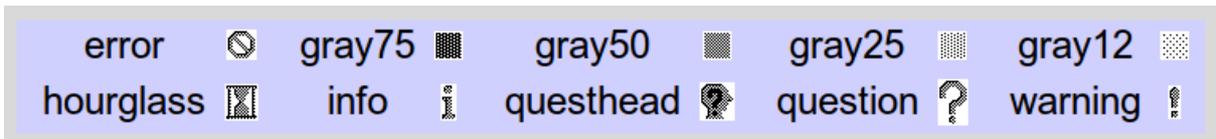
exemple :

```
print (fenPrincipale.winfo_fpixels("3c")) # 3 centimètres
print (fenPrincipale.winfo_pixels("3c"))
106.32911392405063
106
```

4. l'attribut bitmap

Les widgets tels que `Button` ou `Label` ont un attribut `bitmap` qui est exclusif de `text` mais est lui-même supplanté par `image`. Il s'agit par commodité de dix images au trait, **garanties** par `tkinter`, et qui s'adaptent à la coloration donnée par l'attribut `foreground` (noir par défaut). La valeur de l'attribut la chaîne d'identification du dessin. L'usage de l'UNICODE et des images a rendu très marginal l'utilisation de cet attribut pour les étiquettes.

```
labelTxt7 = Label(monCadre, bg="#d0d0ff", text="questhead", font=maFonte)
labelBt7 = Label(monCadre, bg="white", bitmap="questhead" )
```



Sous Linux, on peut utiliser un fichier au format `xbm` ; faire précéder le path du fichier d'un `@`.

Sous Windows, voir avec un `*.ico` valide.

5. l'attribut cursor

Il existe de nombreux curseur préfinis dont voici la liste :

```
arrow, based_arrow_down, based_arrow_up, boat, "bogosity",
bottom_left_corner, bottom_right_corner, bottom_side, bottom_tee,
box_spiral, center_ptr, circle, clock, coffee_mug, cross, cross_reverse,
crosshair, diamond_cross, dot, dotbox, double_arrow, draft_large,
draft_small, draped_box, exchange, fleur, gobbler, gumby, handl, hand2,
heart, icon, iron_cross, left_ptr, left_side, left_tee, leftbutton,
ll_angle, lr_angle, man, middlebutton, mouse, pencil, pirate, plus,
question_arrow, right_ptr, right_side, right_tee, rightbutton, rtl_logo,
sailboat, sb_down_arrow, sb_h_double_arrow, sb_left_arrow, sb_right_arrow,
sb_up_arrow, sb_v_double_arrow, shuttle, sizing, spider, spraycan, star,
target, tcross, top_left_arrow, top_left_corner, top_right_corner,
top_side, top_tee, trek, ul_angle, umbrella, ur_angle, watch, xterm
```

Cependant, sur les systèmes Windows, on ne trouve que :

```
arrow, center_ptr, crosshair, fleur, ibeam, icon, sb_h_double_arrow,
sb_v_double_arrow, watch, xterm qui sont natifs.
```

Les curseurs suivants sont également disponibles :

```
no (pas de curseur), starting, size, size_ne_sw, size_ns, size_nw_se,
size_we, uparrow, wait
```

L'attribut `cursor` prend comme valeur une chaîne qui est le nom d'un curseur valide. Il faut faire

attention dès que l'on n'utilise pas des curseurs communs aux plate-formes car un nom de curseur non reconnu provoque une erreur.

6. justification des textes

(les mêmes constantes servent pour les gestionnaire de placement `Pack` et `Grid`)

```
LEFT='left'  
TOP='top'  
RIGHT='right'  
BOTTOM='bottom'  
CENTER="center"
```

7. complément sur les types de données

Quelques rares attributs prennent de valeur entières, flottantes ou chaîne, mais ces valeurs nécessitent des méthodes qui n'appartiennent pas au types primitifs Python. Il suffit de créer de nouvelles classes qui étendent `int`, `float`, `str`, les types `IntVar`, `DoubleVar`, `StringVar`. Ces classes comportent des méthodes spécifiques `get()`, `set()`, `trace()`, `trace_variable()`, `trace_vdelete()`, `trace_vinfo()`. Ces extensions sont utilisées de façon très spécifique dans les widgets qui permettent la saisie de données : `Entry`, `Text`, `Spinbox`, `Scale`.

constructeurs :

* `StringVar (master=None, value=None)` : `master` est un widget propriétaire, `value` une chaîne de caractères, valeur utile de la variable. Même chose pour `IntVar()` et `DoubleVar()`.

méthodes :

* `set (valeur)` : valeur peut être entier, flottant ou chaîne de caractères selon le type. La méthode fixe la valeur utile de la variable.

* `get ()` : retourne la valeur utile de la variable.

* `trace (mode, fonction)` : appelle la `fonction` lorsque la variable change. Le mode est "r", (lecture), "w" (écriture), "u" (indéfini). Retourne une valeur `cbtrace`.

* `trace_variable (mode, fonction)` : même chose que `trace()`.

* `trace_vdelete (mode, cbname)` : efface la capacité d'appel posée par la méthode `trace()` qui avait retourné `cbname`.

* `trace_vinfo ()` : retourne toutes informations sur les appels de fonction posés par `trace`

tk02 : BitmapImage et PhotoImage

1. questions de formats

1.1. l'attribut image

Les widgets `Label`, `Button`, `Canvas`, `Text` permettent d'insérer une image. Il existe deux types d'instances qui peuvent être valeurs d'un attribut `image` de ces widgets : les instances de `BitmapImage`, les instances de `PhotoImage`.

1.2. modes d'une image bitmap

On rappelle qu'il existe essentiellement **quatre modes** possibles pour une image en mode bitmap :

- le mode trait : les pixels peuvent prendre deux valeurs. Le pixel est codé sur un bit, qui peut donc prendre les valeurs 0 ou 1. Le pixel à 0 est dit de couleur de fond et le pixel à un est dit noir. Le résultat à l'affichage dépend de l'afficheur ; en général la couleur de fond est la transparence, et les pixels noirs sont opaques. Mais on peut très bien avoir deux teintes quelconques pour le fond et l'opacité.
- le mode monochrome, ou en niveaux de gris. Le pixel est codé sur 8 bits, et il y a 256 niveaux de gris, allant du blanc (codé 0) au noir (codé 255).
- le mode trichrome RGB. Le pixel est codé sur 3 octets, le premier étant le niveau de rouge, le second le niveau de vert, le troisième le niveau de bleu dans la synthèse additive.
- le mode palette. Il existe une palette de 256 teintes codées sur 3 octets ; chaque pixel est codé sur un octet, qui est la référence à une teinte de la palette. Les teintes de la palette peuvent être quelconques, et l'une d'elle peut être une transparence.

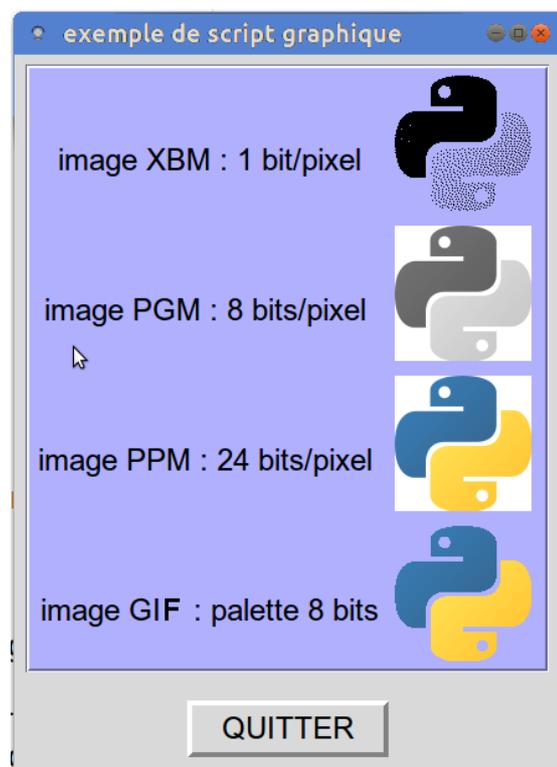
tkinter connaît ces quatre modes, même s'il utilise en interne un mode RGB sur 6 octets.

1.3. question de fichier

Les classes `BitmapImage` et `PhotoImage` utilisent dans leur constructeur la référence à un fichier image, permettant ainsi de passer d'un fichier à la valeur de l'attribut `image` ou `bitmap` lui correspondant. Seulement, si **tkinter** est capable de traiter les 4 modes, il n'accepte en entrés qu'un **format de fichier graphique** pour chaque mode :

- pour le mode trait : les **fichiers xbm**
- pour le mode monochrome : les **fichiers pgm**
- pour le mode trichrome : les **fichiers ppm** (aussi appelés **pnm** ; c'est le nombre magique qui est vérifié)
- pour le mode palette : les fichiers **gif**

Comme ces formats sont parfaitement traités par les éditeurs graphiques comme **GIMP**, il n'y a donc aucun problème pour les fichiers d'images associés à une application.



Noter le tramage de l'image au trait

Pour un travail dynamique, il n'en est plus de même, et il est nécessaire de faire appel à une librairie spécialisée, ce qui ne devrait pas poser de problème avec la librairie PIL, sauf qu'en ce début 2013, la librairie PIL compatible avec Python 3x n'existe pas encore.

La version 8.6. de **tkinter** devrait accepter les formats **.png**, ce qui serait une actualisation importante.

1.4. un exemple

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *

# fenêtre principale
fenPrincipale = Tk()
fenPrincipale.title("exemple de script graphique")
maFonte = ("Arial", -24 )

# quitter le logiciel
def quitter():
    fenPrincipale.quit()

# cadre et images en label
monCadre = Frame(fenPrincipale, bg="#b0b0ff", borderwidth=3,
                 relief=GROOVE)
monCadre.pack(pady=10, padx=10, fill="both", expand=True)
imgXBM = BitmapImage(file="./img/pylogo.xbm")
labelXBM = Label(monCadre, text="image XBM : 1 bit/pixel", font=maFonte,
                 bg="#b0b0ff")
labelImgXBM = Label(monCadre, image = imgXBM, font=maFonte, bg="#b0b0ff")
imgPGM = PhotoImage(file="./img/pylogo.pgm")
labelPGM = Label(monCadre, text="image PGM : 8 bits/pixel", font=maFonte,
                 bg="#b0b0ff")
labelImgPGM = Label(monCadre, image = imgPGM, font=maFonte, bg="#b0b0ff")
imgPPM = PhotoImage(file="./img/pylogo.ppm") # ou "./img/pylogo.pnm"
labelPPM = Label(monCadre, text="\ nimage PPM : 24 bits/pixel",
                 font=maFonte, bg="#b0b0ff")
labelImgPPM = Label(monCadre, image = imgPPM, font=maFonte, bg="#b0b0ff")
imgGIF = PhotoImage(file="./img/pylogo.gif")
labelGIF = Label(monCadre, text="image GIF : palette 8 bits",
                 font=maFonte, bg="#b0b0ff")
labelImgGIF = Label(monCadre, image = imgGIF, font=maFonte, bg="#b0b0ff")

# géométrie
labelXBM.grid(row=0, column=0)
labelImgXBM.grid(row=0, column=1, padx=10, pady=5)
labelPGM.grid(row=1, column=0)
labelImgPGM.grid(row=1, column=1, padx=10, pady=5)
labelPPM.grid(row=2, column=0)
labelImgPPM.grid(row=2, column=1, padx=10, pady=5)
labelGIF.grid(row=3, column=0)
```

```

labelImgGIF.grid(row=3, column=1, padx=10, pady=5)

# bouton quitter
btQuitter = Button(fenPrincipale, text="QUITTER", font= maFonte ,
command=quitter, borderwidth=4, width=10 )
btQuitter.pack(pady=10)

# boucle de la fenetre principale
fenPrincipale.mainloop()

# fichier tk03ex00.py

```

2. la classe BitmapImage

3.1. syntaxe

variable = BitmapImage(options)

3.2. les options

file	string	chemin de fichier image au format xbm qui fournit les données de l'image
data	string	fournit les données de l'image à partir d'une chaîne ; ignorée si l'option file est posée
background	couleur	couleur du fond de l'image(transparence par défaut)
foreground	couleur	couleur des pixels opaques (noir par défaut)

3.3. les méthodes

* config (options), configure (options) : permet de poser des options

* width () : retourne un entier, largeur de l'image bitmap

* height () : retourne un entier, hauteur de l'image bitmap

* type : retourne "bitmap"

3. la classe PhotoImage

3.1. syntaxe

variable = PhotoImage(options)

3.2. les options

file	string	chemin de fichier image au format gif, pgm, ppm (ou pnm) qui fournit les données de l'image
data	string	fournit les données de l'image à partir d'une chaîne ; ignorée si l'option file est posée
width	entier	largeur d'affichage (pas celle de l'image, qui ne change pas)
height	entier	hauteur d'affichage (pas celle de l'image, qui ne change pas)
gamma	float	par défaut, 1.0

3.3. les méthodes

paramètres

- * `config (options)`, `configure (options)` : permet de poser des options
- * `width ()` : retourne un entier, largeur de l'affichage de l'image
- * `height ()` : retourne un entier, hauteur de l'affichage de l'image
- * `type` : retourne "photo"

pixels

- * `get (x,y)` : retourne un chaîne ; en mode monochrome, elle contient la valeur du pixel, en trichrome, la suite des composantes RGB, avec l'espace comme séparateur.
- * `put (data)`, `put (data, to=None)` : écrit une séquence de pixels colorés dans l'image, en commençant à la position `to` : `img.put("{red green} {blue yellow}", to=(4,6))`
- * `blank ()` : change le contenu de l'instance en rendant le contenu transparent.
- * `write (fichier, format, from_coords)` : écrit un fichier `gif` ou `pgm` à partir d'une instance de `PhotoImage` ; `fichier` est le chemin du fichier, `format` est "gif" ou "pgm", `from_coords` un tuple du cadre à découper et écrire. Si le tuple comporte deux entiers, le cadre commence au point de coordonnées précisé et va jusqu'en bas à droite. Si il en comporte 4, les deux derniers sont les coordonnées de la fin du cadre.

```
imgPPM = PhotoImage(file="./img/pylogo.pgm")
imgPPM.write("./img/pylogorecopy.gif", "gif", (8,8,70,70))
```

copies

- * `copy ()` : duplique l'instance de `PhotoImage` et retourne le duplicata.

changements d'échelle

- * `zoom (x, y)` : retourne une instance de `PhotoImage` où chaque colonne est dupliquées x fois et chaque ligne y fois. **x et y sont des entiers**. Si y est absent ou "", alors y est pris égal à x.
- * `subSample (x, y)` : retourne une instance de `PhotoImage` où une seule colonnes toutes le x colonnes est prise et une seule ligne toutes les y lignes. **x et y sont des entiers**. Si y est absent ou "", alors y est pris égal à x.

```
imgZoom=imgPPM.zoom(8).subsample(3)
labelImgPPM = Label(monCadre, image = imgZoom, font=maFonte)
```

Il n'y a pas à proprement parler de changement d'échelle au sens où on l'entend dans les outils graphiques (GIMP, PIL). Il n'y a aucune interpolation, et le résultat obtenu en permutant les deux fonction est très différent. Il faut éviter d'encombrer la mémoire avec des objets trop gros. Les deux fonctions ont simplement pour vocation de permettre l'ajustement d'une image ou d'un logo.

4. Un problème avec l'attribut «image»

Il y a un problème dans Tkinter à propos de l'attribut image des widgets. C'est que **la valeur prise par l'attribut n'est pas référencée dans le compteur d'occurrences de l'objet**. Autrement dit, si on écrit `image=monImage`, le nombre d'occurrences de `monImage` dans l'environnement du script ne change pas !

4.1. on considère le script suivant :

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
```

```

from tkinter import *

fenetre = Tk()
maFrame = Frame (fenetre)
maFrame.pack(packx=20, pady=20)
monPathIcône = "./pylogo.gif"

class MaClasseLabel (Label) :
    def __init__ (self, proprietaire= None) :
        Label.__init__ (self, proprietaire, border = 2, relief = "solid")
        monIcône = PhotoImage (file = monPathIcône)
        self.config (image=monIcône)

monLabel = MaClasseLabel (maFrame)
monLabel.pack ()

fenetre.mainloop ()

```

Si on déroule l'exécutions :

- lignes 1 à 10 : RAS
- ligne 11 à 16 : création de la classe
- ligne 17 : création de `monLabel` comme instance de `maClasseLabel` ; appel de `__init__()`
 - ligne 14 : création de la variable locale `monIcône` ; appelons `Python1` l'objet image. Il y a alors une référence sur `Python1`, la variable `monIcône`.
 - ligne 15 : image pointée sur `Python1`, mais sans changer le compteur d'occurrences ! C'est le bug.
 - ligne 16 : la fonction se termine, et `monIcône` n'existe plus ; `Python1` passe au ramasse miettes.
- ligne 18 : le label est affiché, avec les caractères (dimension, fond etc) qu'il avait au retour du constructeur, mais l'image a disparu !

4.2. traiter le bug

L'anomalie ne se produit que pour les images ; cela doit être dû à la manière dont `Tkinter` gère ses images, comme ressource interne. Pour palier à cette déficience, la recette est simple : s'arranger pour qu'il y ait toujours une référence sur l'image, au moins tant que vit l'instance ce qui suggère un code de la forme :

```

class MaClasseLabel (Label) :
    def __init__ (self, proprietaire= None) :
        Label.__init__ (self, proprietaire, border = 2, relief = "solid")
        monIcône = PhotoImage (file = monPathIcône)
        self.config (image=monIcône)
        self.nInporteQuoi = monIcône

```

Mais pour d'autres genre d'applications multi images, on pourrait avoir :

```

from tkinter import *

fenetre = Tk()
maFrame = Frame (fenetre)
maFrame.pack(packx=20, pady=20)
lePathIcône = "./pylogo.gif"
autrePath = "./py_gif.gif"

class MaClasseLabel (Label) :
    def __init__ (self, proprietaire, monPathIcône, img={}) :
        Label.__init__ (self, proprietaire, border = 2, relief = "solid")
        if not (monPathIcône in img):
            img[monPathIcône] = PhotoImage(file = monPathIcône)
        self.config (image=img[monPathIcône])
        print(len(img))

monLabel = MaClasseLabel (maFrame, lePathIcône)
monLabel.pack (pady=10)
monLabelPy =MaClasseLabel (maFrame, autrePath)
monLabelPy.pack (pady=10)
monAutreLabel = MaClasseLabel (maFrame, lePathIcône)
monAutreLabel.pack (pady=10)

fenetre.mainloop()

```

tk03 : les fontes

L'accès aux fontes se fait de deux façons :

- soit en utilisant **des descripteurs de fontes** ; les descripteurs sont connus du cœur de **tkinter** et ne nécessitent de passer par un module importé. C'est l'usage commun.
- soit en utilisant une instance de la classe `Font`, qui est un module à importer. Le module `Font` comporte des propriétés supplémentaires et il est requis pour un travail plus approfondi.

1. descripteur de fonte

La valeur d'un attribut fonte se code communément comme **un descripteur de fonte**

1.1. caractéristiques d'une fonte

On caractérise une fonte de caractères par les données suivantes :

la famille	<code>family</code>	Le nom de famille de la fonte : <i>Arial, Times, Comics</i>
la taille	<code>size</code>	La dimension du caractère ; il s'exprime soit en relatif (<i>pixels</i>), soit en absolu (<i>points</i>)
le poids ou la graisse	<code>weight</code>	L'épaisseur du trait dans le glyphe ; il y a deux valeurs pour le poids, normal (" <code>normal</code> ") et gras (" <code>bold</code> ").
l'inclinaison	<code>slant</code>	Il y a deux valeurs pour l'inclinaison, normal (" <code>roman</code> ") et italique (" <code>italic</code> ")
le souligné	<code>underline</code>	Il y a un seul souligné
le barré	<code>overstrike</code>	Il y a un seul barré

1.2. valeur de fonte : les descripteurs de fonte

* valeur de fonte dans un tuple

La première méthode est de décrire la fonte par un tuple d'au plus trois éléments :

(famille, taille, modificateurs).

- Famille et modificateurs sont des chaînes et taille un entier.
- si le tuple a un seul élément, c'est la famille, deux éléments, la famille et la taille.
- la taille est présumée être en **points** ; **une taille en pixels s'exprime par un entier négatif.**
- Les paramètres de la chaîne des modificateurs sont dans un ordre indifférent, séparés par des espaces.

```
maFonte = ("Helvetica")
maFonte = ("Helvetica", -30)
maFonte = ("Helvetica", -30, "overstrike bold")
```

* valeur de fonte dans une liste

Le tuple peut être remplacé par une liste de structure identique. La fonte devient alors modifiable. (Ne pas oublier qu'on ne peut pas modifier un tuple)

* valeur de fonte dans une chaîne

Le tuple peut être transformé en une chaîne unique. le séparateur est alors l'espace ; mais il y a un problème avec les noms de famille qui contiennent déjà des espaces ; on les met alors entre accolades.

```
maFonte = "Courier 30 bold"
maFonte = "{Deja Vu Sans Light} -30 underline"
```

note : En ce qui concerne le nom de famille des fontes, la casse n'est pas significative. Il n'en est pas de même avec les modificateurs, impérativement en minuscules.

1.3. familles prédéfinies

Dans le cas où la famille n'existe pas, elle est remplacée «au mieux», ce qui ne donne pas un résultat assuré. Il existe quatre familles «théoriquement» présentes : **Helvetica**, **Times**, **Courier et System**. Si elles sont présentes, il n'y a pas de problème ; sinon, ce sont les fontes définies sur la plate-forme qui sont choisies comme fontes Sans, Sérif et espacement fixe.

En principe, les fontes Sans, Sérif, et Monospace sont des fontes vectorielles (True Type, Post Script). La fonte **System** est la fonte par défaut du système, dont les caractères sont en bitmap, et où seulement certaines tailles sont disponibles ; dans le cas où la taille n'est pas disponible, c'est la taille la plus voisine qui est retenue.

Il est recommandé d'utiliser les fontes présentes sur les plate-forme actuelles comme **DejaVu** :

```
maFonte = ("Deja Vu Sans Light", -30)
```

2. Le module Font

2.1. le constructeur Font

Le constructeur permet de construire une instance de la classe **Font**, qui est la seconde forme de passage de valeur au paramètre **font** des widgets. Attention, cette instance n'est pas un descripteur de fonte ! Si on veut caster cet objet en descripteur, il faut lui appliquer la méthode usuelle en Python, **toString()** qui retourne la forme chaîne du descripteur.

constructeur :

```
maFonte = Font (options)
```

2.2. les options de Font

Les options se présentent nécessairement sous la forme clef/valeur.

- * **font** : la valeur est un descripteur sous la forme tuple ; normalement si l'option font est présente, il n'y a plus d'autres options dans le constructeur.
- * **name** : permet de changer le nom de la fonte qui est octroyé par défaut. Le nom doit être unique dans le logiciel ! Ce paramètre ne devrait jamais être employé.
- * **family** : "Courier", "Times", "Helvetica" par défaut
- * **size** : entier qui fixe la taille en points ; négative pour une taille en pixels
- * **weight** : NORMAL ou BOLD
- * **slant** : ROMAN ou ITALIC
- * **underline** : booléen
- * **overstrike** : booléen

2.3. les méthodes d'instance de Font

- * **copy ()** : retourne une autre instance, avec les mêmes options.
- * **actual()**, **configure()** : retourne un dictionnaire des attributs de la fonte
- * **actual (nom d'options)**, **cget(nom d'option)** : retourne la valeur de l'option
- * **configure (options clef/valeurs)** :
- * **measure (text)** : retourne la largeur du texte en pixels.
- * **metrics ()** : retourne le dictionnaire des «métriques», type d'espacement ("**fixed**"), ascendant

"*ascent*"), descendant ("*descent*"), largeur de ligne ("*linesize*".

* *metrics* (*noms d'options*) : retourne les valeurs pour les «métriques». Pour ces deux méthodes, s'assurer que la fonte est utilisée au moins une fois avant de les appeler.

2.4. fonctions du module `font`

* *families* () : retourne **la liste** des familles présentes sur la plate-forme.

* *names* () : retourne la liste de noms de fontes définies dans l'application.

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *
from tkinter.messagebox import askyesno
from tkinter.font import *

# fenêtre principale
fenPr = Tk()
fenPr.title("bindtags")
# quitter le logiciel
def quitter():
    reponse = askyesno("terminer le script",
        "Voulez-vous réellement terminer\u00a0? \n cliquer «oui» pout finir")
    if reponse :
        fenPr.quit()
fenPr.protocol("WM_DELETE_WINDOW", quitter)
# *****
maFonte = Font(name="nomExceptionnel", family="DejaVu Sans Mono",
    size= -25)
# *****
# bouton quitter
btQuitter = Button(fenPr, font=maFonte, text="QUITTER", command=quitter)
btQuitter.pack(pady=20)

def fnFontes() :
    print ("maFonte.actual() :", maFonte.actual())
    print ("maFonte.config(slant=ITALIC) :", maFonte.config(slant=ITALIC))
    print ('maFonte.actual("family") :', maFonte.actual("family"))
    print ("maFonte.metrics() :", maFonte.metrics())
    print ('maFonte.measure("mon message") :',
        maFonte.measure("mon message"))
    print ("families() :", families())
    print ("names() :",names())

btFontes = Button(fenPr, font=maFonte, text="fontes", command=fnFontes)
btFontes.pack(pady=20)
```

```
# boucle de la fenêtre principale
fenPr.mainloop()
fenPr.destroy() # par précaution
# fichier tk03ex00.py
```

le résultat : (reformaté)

```
maFonte.actual() : {'family': 'DejaVu Sans Mono', 'weight': 'normal',
                    'slant': 'roman', 'overstrike': 0, 'underline': 0,
                    'size': -25}

maFonte.config(slant=ITALIC) : None

maFonte.actual("family") : DejaVu Sans Mono

maFonte.metrics() : {'fixed': 1, 'ascent': 24, 'descent': 6, 'linespace': 30}

maFonte.measure("mon message") : 165

families() : ('UnDotum', 'Monotype Corsiva', 'LMMonoLt10', 'Century Schoolbook L',
             'OpenSymbol', 'Wingdings 3', 'Wingdings 2', 'Code EAN13', 'Khmer OS System',
             'LMSansQuot8', 'LMMathSymbols10', 'LMRomanSlant8', 'LMRomanSlant9', 'LMSans8',
             'LMSans9', 'Mukti Narrow', 'Meera', 'Lucida Bright Math Symbol', 'Lucida Stars',
             'Webdings', 'Vemana2000', 'LMMonoSlant10', 'Umpush', 'DejaVu Sans Mono', 'Purisa',
             'Pothana2000', 'Norasi', 'Loma', 'Wingdings', 'URW Palladio L', 'Untitled1',
             'Phetsarath OT', 'Sawasdee', 'Tlwg Typist', 'Code 128', 'Lucida Fax', 'Lucida
             Bright', 'URW Gothic L', 'Dingbats', 'URW Chancery L', 'Ubuntu', 'FreeSerif',
             'Times New Roman', 'Ubuntu Condensed', 'orilUni', 'DejaVu Sans', 'Gabriola', '文泉
             驛等寬微米黑', 'Kedage', 'Lucida Icons', 'DejaVu Sans', 'Lucida Calligraphy',
             'DejaVu Serif', 'Kinnari', ...
             ...'LMRomanSlant12', 'LMRomanSlant17', 'Courier New', 'Waree', 'gargi', 'Code 3 de
             9', 'Lohit Hindi', 'DejaVu Serif', 'Saab', 'LMMonoProp10', 'Garuda', 'Rekha',
             'FreeMono', 'Ubuntu Mono', 'URW Bookman L', 'LMMonoPropLt10')

names() : ('nomExceptionnel', 'TkCaptionFont', 'TkSmallCaptionFont',
           'TkTooltipFont', 'TkFixedFont', 'TkHeadingFont',
           'TkMenuFont', 'TkIconFont', 'TkTextFont', 'TkDefaultFont')
```

tk04 : géométries

Le terme de «**géométrie**» (*geometry*) recouvre tous les questions relatives au placement des éléments graphiques fenêtrés, les fenêtres de haut niveau (principale et Toplevel) et les autres widgets (boutons, cadres etc). La mise en place des composants est réalisée par un **gestionnaire de fenêtre** (*window manager*) qui est un **arbitre** de toutes les contraintes dimensionnelles proposées implicitement (contraintes d'un gestionnaire de placement, dimension de la fonte effectivement utilisée) ou explicitement (dimension requise pour un widget).

1. géométrie des fenêtres

1.1. geometry

Les fenêtres de haut niveau sont situées par rapport à l'écran où elle s'affichent. Il faut donc 4 paramètres pour constituer le dimensionnement d'une telle fenêtre : sa largeur (**w**), sa hauteur(**h**), le nombre de pixels entre le bord gauche de l'écran et la gauche de la fenêtre (**l**), et le nombre de pixels entre le bord haut de l'écran et le haut de la fenêtre (**t**).

exemple : "**w**x**h**+**l**+**t**" = "400x500+300+250" ; si on prend les bordure de la fenêtre en considération, remplacer les signes + par - ; exemple : "400x500-300-250"

Note : il se peut que «l'écran» physique ne coïncide pas avec l'écran disponible. Cela dépend de l'interface graphique. La géométrie travaille avec l'écran disponible.

La géométrie est caractérisé par la chaîne de géométrie . Son format est : "**w**x**h**+**l**+**t**" où **w**, **h**, et **t** sont des entier décimaux sans signe.

1.2. fixer la géométrie d'une fenêtre

Toute fenêtre a une géométrie par défaut, imposée par les gestionnaires de placement qui doivent trouver la place pour les composants ! Si l'affichage n'a pas eu lieu, la chaîne est "**1x1+0+0**".

* la méthode `geometry()` d'une fenêtre de haut niveau permet de requérir un changement de géométrie. L'argument est une chaîne de géométrie.

Attention : la géométrie d'une fenêtre subit les aléas du placement de ses widgets, et peut être remise en cause par les gestionnaires de placement des widgets, comme on le verra ultérieurement.

1.3. retrouver la géométrie d'une fenêtre

* la méthode `geometry()` sans argument retourne la valeur actuelle de la géométrie. Mais la référence n'existe que si le **gestionnaire de fenêtre a fait ses calculs !**

- si la fenêtre est affichée, il n'y a pas de problème.

- Mais, le plus souvent, on a besoin des renseignements de géométrie avant l'affichage, par exemple pour réaliser un centrage de popup. Il faut alors forcer le calcul, sans cependant provoquer l'affichage. La méthode `update_idletasks()` y pourvoit.

Il existe des **méthodes** qui permettent d'accéder directement aux paramètres et sur lesquelles on reviendra dans le chapitre sur le gestionnaire de fenêtres.

On peut citer :

* `winfo_width()`, `winfo_height()` : retournent **w** et **h**

* `winfo_rootx()`, `winfo_rooty()` : retourne les coordonnées du coin supérieur gauche de la fenêtre principale dans l'écran ;

Ces méthodes supposent également un calcul préalable de la géométrie. Elles travaillent avec l'**écran physique**.

<pre>print (fenPr.geometry()) fenPr.update_idletasks() print (fenPr.geometry(), fenPr.winfo_geometry())</pre>	<pre>1x1+0+0 763x159+59+590</pre>
--	-----------------------------------

```
print (fenPr.winfo_width(),fenPr.winfo_height())
print (fenPr.winfo_rootx(),fenPr.winfo_rooty())
```

```
763x159+67+625
763 159
67 625
```

2. Les gestionnaires de placement

2.1. les trois gestionnaires

Un gestionnaire de placement est une classe dont les méthodes permettent de placer un widget dans son maître (*conteneur, master*). Les conteneurs peuvent être les fenêtres de haut niveau ou les cadres (*Frame*). On verra dans l'étude des widgets *Text* et *Canvas* des cas où le placement ne se fait pas avec les gestionnaires de placement.

Il existe 3 gestionnaires de placement : *Pack*, *Grid*, *Place*

2.2. règles d'usage

* les widgets plaçables du noyau de *tkinter* sont les suivants :

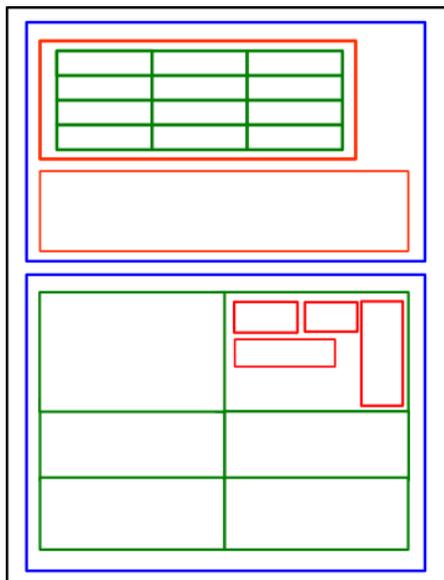
Frame, *LabelFrame*, *PanedWindow*, *Scrollbar*, *Label*, *Button*, *Checkbutton*, *Listbox*, *Radiobutton*, *Entry*, *OptionMenu*, *Text*, *Canvas*, *Scale*, *Spinbox*.

(ne sont donc pas plaçables : *Tk()*, *Toplevel*, *Menu*)

Les widgets additionnels (*tix*, *ttk*) sont plaçables également.

* les widgets plaçables héritent des trois classes *Pack*, *Grid*, *Place*

* si un gestionnaire de placement est utilisé dans une instance de conteneur, aucun autre gestionnaire ne peut être utilisé dans la même instance. Cette règle est souvent mal comprise : on peut très bien changer de gestionnaire quand on passe d'un conteneur à un de ses enfants.



exemple :

- * la fenêtre principale est en noir .
- * les widgets en bleu utilisent le gestionnaire *Pack*
- * les widgets en rouge utilisent le gestionnaire *Place*
- * les widgets en vert utilisent le gestionnaire *Grid*.

Il n'y a pas d'erreur lorsqu'on mélange les gestionnaires dans un même conteneur ; mais le gestionnaire de fenêtre négocie indéfiniment entre eux, ce qui met l'application dans une boucle infinie et font planter l'application.

3. Le gestionnaire *Pack*.

3.1. fonctionnement du gestionnaire *Pack*

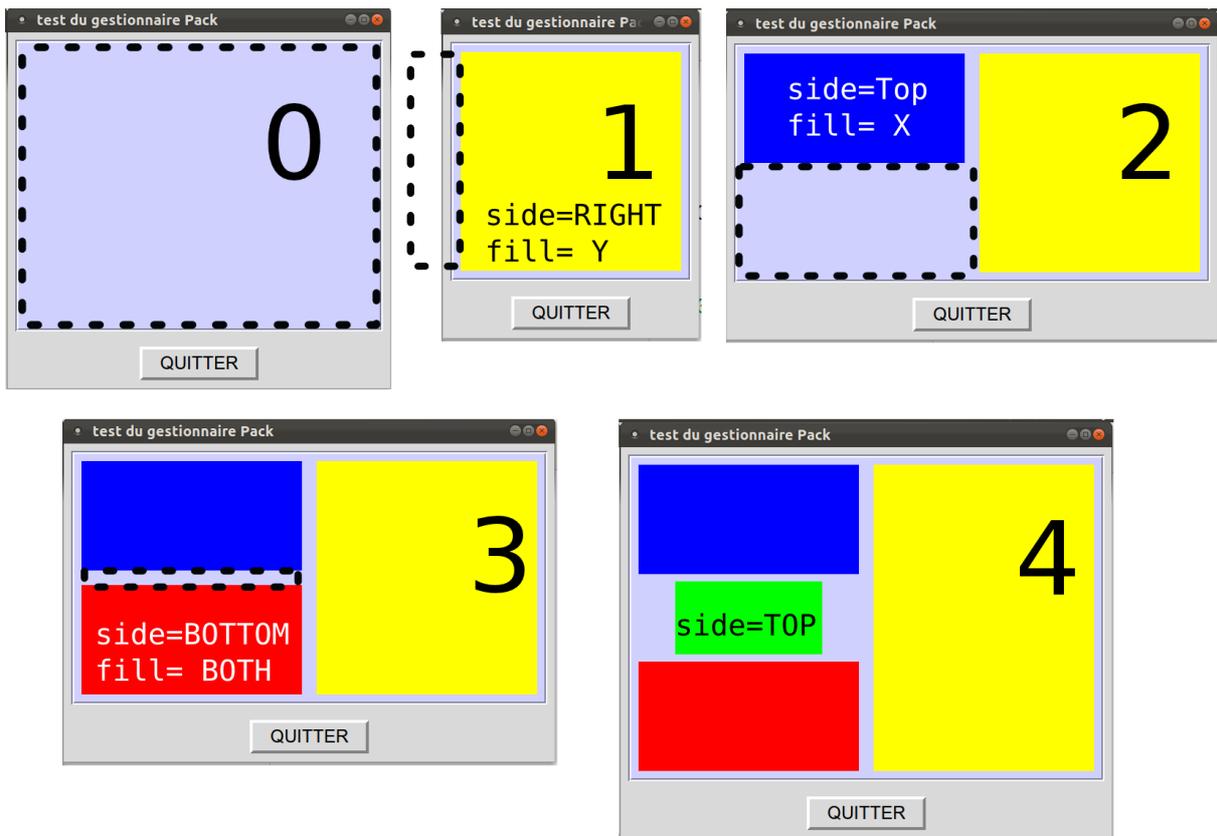
L'utilisation de ce gestionnaire est délicate dès qu'une interface est un peu complexe. Son interprétation est assez difficile à comprendre.

Les enfants sont pris en charge dans l'ordre où ils se présentent, avec leur dimension. Ils sont placés sur le bord défini par `side` dans le «rectangle restant». Pour le premier enfant, c'est le conteneur tel qu'il est. L'enfant est placé, le conteneur s'ajuste. Si le premier enfant est placé sur le bord haut (TOP), le rectangle restant est dessiné en dessous, avec sa largeur. (ce serait à gauche pour RIGHT, à droite pour LEFT, au dessus pour BOTTOM). Le second enfant est alors placé avec pour conteneur le «rectangle restant»... On continue ainsi, en tenant compte du remplissage éventuel ; les dimensions requises en cas de remplissage peuvent être perturbés.

Le gestionnaire `Pack` dans un conteneur a pour effet de l'ajuster à ses composants sauf s'il y a une contrainte posée par le gestionnaire de son propre conteneur. **La dimension proposée pour le conteneur n'est en général pas prise en compte !**

Dans l'exemple suivant, le «rectangle restant» est symbolisé par un pointillé

exemple



```

from tkinter import *

application = Tk()
application.title("test du gestionnaire Pack")
maFonte = ("Arial", -24 )

# quitter le logiciel
def quitter():
    application.quit()

monCadre = Frame(application, bg="#d0d0ff", width=500, height=400,
                  relief=GROOVE, border=4)

```

```

monCadre.pack(pady=10, padx=10, fill=BOTH, expand='1')
# fin étape 0
frame1 = Frame(monCadre, bg="yellow", width=300, height=300, padx=10,
pady=10)
frame1.pack(side=RIGHT,fill=Y, padx=10, pady=10)
# fin étape 1
frame2 = Frame(monCadre, bg="blue", width=300, height=150)
frame2.pack(side=TOP, fill=X, padx=10, pady=10)
# fin étape 2
frame3 = Frame(monCadre, bg="red", width=150, height=150)
frame3.pack(side=BOTTOM, fill=BOTH, expand=1, padx=10, pady=10)
# fin étape 3
frame4 = Frame(monCadre, bg="green", width=200, height=100, padx=10,
pady=10)
frame4.pack()
# fin étape 4

# bouton quitter
btQuitter = Button(application, text="QUITTER", font= maFonte,
command=quitter, borderwidth=4, width=10 )
btQuitter.pack(pady=10)

application.mainloop()

# fichier tk04ex00.py

```

3.2. les méthodes de widget

- * `pack(options)` : c'est la méthode qui réalise le placement, suivant les options posées.
- * `pack_configure(options)` : définit la même méthode que `pack()`
- * `pack_forget()` : la méthode détruit l'effet du gestionnaire de géométrie. Ceci implique que le widget existe toujours, mais qu'il n'est plus affiché et qu'un nouvel appel à la méthode `pack()` peut être envisagé.
- * `pack_info()` : la méthode retourne un dictionnaire des options.

<pre> cadre0 = Frame(monCadre,bg="red", width=150, height=40, relief=GROOVE, border=4) cadre0.pack_configure(padx=10, pady=10, side=LEFT) print (cadre0.pack_info()) </pre>	<pre> {'side': 'left', 'ipady': '0', 'ipadx': '0', 'in': '< ;;;>', 'pady': '10', 'padx': '10', 'anchor': 'center', 'expand': '0', 'fill': 'none'} </pre>
---	--

3.3. les valeurs d'attribut :

attribut side

LEFT='left'
 TOP='top'
 RIGHT='right'
 BOTTOM='bottom'

attribut anchor pour pack

N='n'	NW='nw'	CENTER='center'
S='s'	SW='sw'	
W='w'	NE='ne'	
E='e'	SE='se'	

attribut fill

NONE='none'
 X='x'
 Y='y'
 BOTH='both'

3.4. les options

side	voir ci-dessus	côté où placer le widget ; par défaut, c'est TOP
fill	voir ci-dessus	NONE par défaut ; sinon dit si le widget doit occuper toute la ligne disponible (x) ou toute la colonne (y), tout l'espace disponible (BOTH)
expand	flag booléen	Mettre expand à True pour que fill soit pris en compte dans un changement de taille du conteneur
in (in_)	widget	À utiliser si le widget n'est pas dans son conteneur mais un descendant de son parent. Comme in est réservé, utiliser in_
padx, pady	integer	marges extérieures au widget
ipadx, ipady	integer	marges intérieures au widget (la bordure sépare marges intérieures et marges extérieures)
anchor	voir ci-dessus	placement à l'intérieur de l'espace dévolu au widget ; par défaut, CENTER

4. le gestionnaire Grid

4.1. principe du gestionnaire Grid

Le gestionnaire **Grid** est plus simple à comprendre que le précédent : il rappelle le fonctionnement du placement par tableau dans le HTML. Le gestionnaire **Grid**, au premier appel dans un conteneur, crée un **tableau dynamique** (le nombre de lignes, de colonnes, leur dimension peuvent évoluer au fur et à mesure des appels) et place les widgets dans ce tableau, ajustant le tableau aux dimensions des widgets.

En principe, **on ne met qu'un widget par cellule** ; si on en met plusieurs, ils se superposent facilement et le résultat est illisible.

La **hauteur d'une ligne** est celle de la plus haute cellule de la ligne ; la **largeur de colonne** est celle de la plus large cellule de la colonne.

Les lignes et colonnes sont **numérotées** (à partir de 0), et placées en ordre croissants ; les numéros n'ont pas besoin d'être consécutifs (par exemple, les lignes peuvent être numérotées de 10 en 10) ; une cellule peut rester vide.

La **cellule est ajusté à son contenu si une dimension est insuffisante**. Mais il est fréquent qu'une cellule soit plus grande que son contenu ; il est donc important de pouvoir **ancrer** le widget dans sa cellule. Ici également, le conteneur est ajusté, mais au tableau résultant.

Les programmeurs utilisent volontiers ce gestionnaire, et même recommandent de n'utiliser que lui seul (Shipman).

4.2. les méthodes de Grid

méthodes concernant le widget

- * `grid(options)` : la méthode est la méthode appelée pour placer le widget
- * `grid_configure(options)` : la méthode est identique à `grid(options)`.
- * `grid_info()` : la méthode retourne un dictionnaire des options.

méthodes concernant la grille

- * `columnconfigure(N, options de configuration de grille)` : permet à la colonne **N** d'échapper aux valeurs de dimension par défaut.
- * `rowconfigure(N, de configuration de grille)` : permet à la ligne **N** d'échapper aux valeurs de dimension par défaut.
- * `grid_box(column=None, row=None, col2=None, row2=None)` :
- * `grid_location(x, y)` : retourne un tuple donnant la ligne et la colonne du point de coordonnées (x, y) du widget.
- * `grid_size()` : retourne le nombre de lignes et de colonnes dans un tuple.
- * `grid_slaves(row=None, column=None)` : énonce la liste des widgets esclaves
- * `grid_forget()` : la méthode est identique à celle de `Pack`, avec perte de la configuration de grille.
- * `grid_remove()` : la méthode est semblable à `grid-forget()`, sans perte de la configuration de grille.

4.3. les options de configuration de grille

minsize	integer	largeur minimale (colonne) ou hauteur minimale (ligne). Exige que la colonne (resp. la ligne) ne soient pas vide.
pad	integer	ajout de pixel de padding aux dimensions par défaut
weight	integer	manière de distribuer les pixels supplémentaire lors d'un changement de taille de la ligne ou de la colonne. Par exemple, si une colonne a le poids 3, une autre le poids 1, toutes les autres ayant le poids par défaut 0, lors d'un changement de taille, si on doit ajouter 4n pixels à la largeur, la première en prend 3n, la seconde n, et les autres restent identiques.

4.4. les options de cellule

*Les options `padx`, `pady`, `ipadx`, `ipady`, `in_` sont les mêmes que pour `Pack`

* l'option `sticky` qui renseigne sur la manière de "coller" le widget aux bords si la cellule est trop grande.

row	integer	numéro de ligne de la cellule du widget (à partir de 0)
------------	---------	---

column	integer	numéro de colonne de la cellule du widget (à partir de 0)
rowspan	integer	nombre de cellules à fusionner sur la colonne à partir de la ligne actuelle.
columnspan	integer	nombre de cellules à fusionner sur la ligne à partir de la colonne actuelle.
sticky	constante chaîne	les valeurs possibles sont toutes celles de <code>anchor</code> pour <code>pack()</code> , et <code>N+S="ns"</code> , <code>E+W="ew"</code> , et <code>N+S+E+W="nesw"</code> . (les sous séquences de <code>"nesw"</code> et <code>CENTER="center"</code>)

exemple :

<pre>lab3 = Label (monCadre, text="troisième label", font= maFonte) lab3.grid(row=1, column=1, padx=10, pady=10) print (lab3.grid_info())</pre>	<pre>{'rowspan': '1', 'column': '1', 'sticky': '', 'ipady': '0', 'ipadx': '0', 'columnspan': '1', 'in': '< ;;>', 'pady': '10', 'padx': '10', 'row': '1'}</pre>
--	--

5. Le gestionnaire Place

5.1. principe du gestionnaire Place

Le gestionnaire `Place` est le plus simple des gestionnaires de placements ; il permet de placer un widget en explicitant des coordonnées (centre, coins..) soit en position absolue, soit relativement à un élément fenêtré existant. Il permet en outre de fixer explicitement les dimensions du widget.

5.2. les méthode du gestionnaire Place

- * `place(options)` : la méthode place et dimensionne le widget selon les options choisies.
- * `place_configure(options)` : identique à `place(options)`
- * `place_forget()` : même chose que pour `Pack`
- * `place_info()` : même chose que pour `Pack` et `Grid`
- * `place_slaves()` : retourne la liste des widgets «esclaves».

5.3. le valeurs d'options

`INSIDE='inside'`
`OUTSIDE='outside'`

5.4. les options

x	integer	position horizontale absolue dans le conteneur ; 0 par défaut (gauche vers droite)
y	integer	position verticale absolue dans le conteneur ; 0 par défaut (haut vers le bas)
relx	float	position horizontale relative dans le conteneur (voir <code>_in</code>); flottant compris ente 0.0 et 1.0

rely	float	position verticale relative dans le conteneur (voir <code>in_</code>) ; flottant compris entre 0.0 et 1.0
in (<code>_in</code>)	widget	permet de placer un widget relativement à un descendant de ses parents.
width	integer	largeur en pixels du widget
height	integer	hauteur en pixels du widget
relwidth	integer	largeur relative du widget
relheight	integer	hauteur relative du widget
anchor	voir ci-dessus	donne le point qui sert de référence pour la position absolue ou relative du widget. Les valeurs valides sont N,S,E,W, NE, SE, SW, NW et CENTER. En général la valeur par défaut est NW (haut à gauche). N correspond au milieu du bord haut.
bordermode	ci-dessus 5.3	indique s'il faut inclure (OUTSIDE) ou exclure (INSIDE / défaut) la bordure du conteneur dans le calcul de taille et position.

6. la troisième dimension

6.1. empilement des «calques»

Pour modéliser l'affichage dans un interface graphique, on peut adopter la métaphore d'un empilement de calques : toute fenêtre est un graphisme rectangulaire sur un calque indéfini transparent ; tout widget est aussi un graphisme rectangulaire sur un calque. Le calque de chaque widget est lié à la fenêtre dans laquelle il a été défini. Les calques de widget sont empilés dans l'ordre de leur définition, le plus récent étant au dessus. De même les fenêtres d'une application sont empilées dans l'ordre de leur définition graphique (de leur affichage). Il en est de même des applications. Ce modèle permet de comprendre comment se fait l'affichage : le pixel affiché est le pixel qui se situe le plus «au dessus» lorsque l'on passe en revue tous les calques dans l'ordre d'empilement (par défaut, du plus ancien au plus récent).

Cet ordre d'empilement peut être modifié ; on sait qu'une fenêtre qui prend le focus se met immédiatement au dessus de toutes les autres.

6.2. Les méthodes

- * **lift(aboveThis=None)** : élève la fenêtre de haut niveau dans la pile des fenêtres au-dessus de **aboveThis**. Sans argument, met la fenêtre au sommet de la pile. (**aboveThis** est une instance de **Tk** ou **Toplevel**). Appliquée à un widget plaçable, elle permet de manipuler les superpositions de widget d'un même conteneur.
- * **lower(belowThis=None)** : descend la fenêtre de haut niveau dans la pile des fenêtres en dessous de **belowThis**. Sans argument, met la fenêtre en bas de la pile. (**aboveThis** est une instance de **Toplevel** ou **Tk**). Appliquée à un widget plaçable, elle permet de manipuler les superpositions de widget d'un même conteneur.
- * **transient(parent=None)** : Une fenêtre est qualifiée de "passagère" (*transcient*) si elle est attachée à une autre fenêtre (son parent), qu'elle est iconifiée avec son parent, qu'elle disparaît avec lui (**withdraw**), qu'elle est posée directement au dessus de son parent. La paramètre **parent** désigne la fenêtre **parent**, le **conteneur**. Cette méthode avec la méthode **grab_set()** permet en quelque sorte de rendre "modale" une fenêtre dans une application simple, c'est-à-dire qu'elle apparaît en haut de la pile car elle capture le focus et reste liée avec son parent. Voir tk08 §3.4 **Tk** et **Toplevel**

tk05 : attributs partagés

Les paramètres des widgets, fixés à la création ou par le programme sont appelés ses attributs. De nombreux attributs sont **partagés** par une partie, voire tous les widgets (**les attributs standards**) de **tkinter**. Ce sont ces attributs qui vont être décrits ; ceci évitera de nombreuses redites. Les attributs très spécifiques et n'appartenant qu'à un widget seront abordés dans la description de chacun d'eux : **Canvas**, **Text**....

1. méthodes d'attributs

1.1. attributs donnés à un widget

* tous les attributs d'un widget ont une valeur par défaut, qui peut varier selon les widgets.

* tous les attributs d'un widget peuvent être des arguments (facultatifs) de **son constructeur** ; le constructeur d'un widget est de la forme **Widget(constucteur, options)**. Ou de son gestionnaire de placement, appelé sous la forme **widget.gestionnaire(options)**. Ou encore des deux, comme **padx** ou **pady**.

```
monCadre = Frame(fenPr, bg="#b0b0ff", borderwidth=3, relief=GROOVE)
```

1.2. modification d'un ou plusieurs attributs

* **config(options)** : méthode qui modifie des options du widget

* **configure(options)** : méthode qui double **config(options)**

```
# change l'image et le fond d'un label
labelPhoto.config( image = imgGIF, bg="#b0b0ff")
```

On peut modifier les attributs par une affectation sur le modèle suivant :

```
labelPhoto ["image"] = imgGIF
labelPhoto ["background"] = "#b0b0ff"
```

1.3. accès aux attributs

* **config()** : retourne un dictionnaire de tous les attributs d'un widget.

* **configure()** : doublé de **config()**.

On peut aussi accéder aux attributs en utilisant la notation «dictionnaire» :

```
print (labelPhoto ["background"])
```

* **pack_info()**, **grid_info()** et **place_info()** : retournent un dictionnaire de tous les attributs de **placement** d'un widget ; ce dictionnaire est vide si le gestionnaire correspondant n'est pas utilisé.

```
monCadre = Frame(fenPt, bg="#b0b0ff", borderwidth=3, relief=GROOVE)
monCadre.pack(pady=10, padx=10, fill="both", expand=True)
```

```
-----
print (monCadre.config())
```

```
{
'background': ('background', 'background', 'Background',
               <border object at 0xe85a00>, '#b0b0ff'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth',
```

```

        <pixel object at 0xe85a30>,3)
'class': ('class', 'class', 'Class', 'Frame', 'Frame'),
'colormap': ('colormap', 'colormap', 'Colormap', '', ''),
'container': ('container', 'container', 'Container', 0, 0),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'height': ('height', 'height', 'Height', <pixel object at 0xe86540>, 0),
'highlightbackground': ('highlightbackground', 'highlightBackground',
                        'HighlightBackground', <color object at 0xe86810>,
                        '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0xe85be0>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness',
                      'HighlightThickness', <pixel object at 0xe85910>, 0),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0xe86420>,
        <pixel object at 0xe86420>),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0xe85d60>,
        <pixel object at 0xe85d60>),
'relief': ('relief', 'relief', 'Relief', <index object at 0x101d400>,
          'groove'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '0', '0'),
'visual': ('visual', 'visual', 'Visual', '', ''),
'width': ('width', 'width', 'Width', <pixel object at 0xe85cd0>, 0),
}

```

La structure de ce dictionnaire des options est assez complexe :

- * les clefs sont les dénominations d'attributs
- * les valeurs sont des tuples, selon deux modes :
 - certains tuple définissent un alias : 'bd': ('bd', '-borderwidth'),
 - les autres l'attribut sous forme d'un tuple à 5 éléments

Le premier élément du tuple est la clef ;

Pour les tuple à 5 éléments : le deuxième répète la clef, le troisième est la classe de l'attribut dans la base de Tcl/Tk, le quatrième la référence en mémoire ou une valeur booléenne du type C, et le cinquième la valeur actuelle.

* **cget(clef)** : retourne une **chaîne de caractère** donnant la valeur actuelle ; attention, un entier ou un flottant sont convertis en chaîne. Les booléens suivent la convention du C (0/1)

* **keys()** : retourne une liste des clefs des attributs disponibles.

```

print (monCadre.keys())

['bd', 'borderwidth', 'class', 'relief', 'background', 'bg', 'colormap',
'container', 'cursor', 'height', 'highlightbackground', 'highlightcolor',
'highlightthickness', 'padx', 'pady', 'takefocus', 'visual', 'width']

```

2. attributs système

* **class** : c'est la classe du widget. Attention, **class** est réservé en Python et il faut écrire **class_** pour tous usage que l'on peut en faire. Par défaut, la classe est le type de widget, on peut définir des classes plus étendue que l'on utilise pour les **options** de classe de widget, qui portent sur ces "classes".

* **command** : référence le gestionnaire d'événement (par exemple pour **Button**)

* **cursor** : référence le curseur à utiliser lors d'un survol du widget.

* **takefocus** : spécifie si un widget est autorisé à recevoir le focus (il est focusable) ou non. La valeur est 0 (**True**) ou 1 (**False**). La valeur par défaut dépend du type de widget.

3. attributs de bordure et de marge

Tous les widgets possèdent les deux premiers attributs de bordure.

- * **borderwidth** : largeur (integer) de bordure (défaut 1 ou 2 pixels), en pixels
- * **relief** : la forme de la bordure (voir les constantes de relief) pour un widget à l'état normal. La valeur par défaut dépend du widget (**FLAT** pour label, **GROOVE** pour un cadre, **RAISED** pour un bouton) et peut dépendre de l'état du widget si celui-ci en a plusieurs possibles.
- * **highlightthickness** : largeur de surbordure lors d'une prise de focus.
- * **padx, pady** : espacements à ajouter au widgets à l'intérieur de la bordure lors du placement. Il s'agit ici de paramètres de configuration, pas des paramètres de placement !
- * **ipadx, ipady** : espacements à ajouter au widgets à l'intérieur de la bordure lors du placement pour certains widgets.
- * **width** : largeur souhaitée du widget ; peut être exprimé en unité de dimensions. Pour certains widgets ayant un attribut **text** (**Entry**, **Button** etc.) la largeur s'exprime en nombre de caractères.
- * **height** : hauteur souhaitée du widget ; peut être exprimé en unité de dimensions.

4. attributs de couleur.

- * **background, bg** : couleur de fond ; partagé par tous les widgets.
- * **foreground, fg** : couleur de caractère ; partagé par les widgets acceptant du texte ou un bitmap.
- * **highlightbackground** : couleur de surbordure qui n'a pas le focus ;
- * **highlightcolor** : couleur de la surbordure qui a pris le focus).
- * **disabledforeground** : couleur de fond à accorder aux éléments rendus inactifs par (**state="disabled"**).
- * **colormap** : palette pour les écrans anciens.

5. attributs pour les gestionnaires de géométrie (rappel)

Partagés par tous les widgets.

- * **in (in_)** : permet de placer un widget relativement à un descendant de ses parents ; option du gestionnaire de placement.
- * **relwidth** : largeur souhaitée du widget ; valeur relative au conteneur (gestionnaire **Place**)
- * **relheight** : hauteur souhaitée du widget ; valeur relative au conteneur (gestionnaire **Place**)
- * **sticky** : ancrage dans le gestionnaire **Grid**
- * **anchor** : ancrage dans les gestionnaires **Pack** et **Place**
- * **row** : numéro de ligne dans le gestionnaire **Grid**.
- * **column** : numéro de colonne dans le gestionnaire **Grid**.
- * **rowspan** : fusion de lignes dans le gestionnaire **Grid**.
- * **padx, pady** : espacements à ajouter au widgets à l'extérieur de la bordure lors du placement.
- * **columnspan** : fusion de colonnes dans le gestionnaire **Grid**.
- * **x** : offset écran horizontal de l'ancre (gestionnaire **Place**)
- * **relx** : offset relatif horizontal dans le conteneur de l'ancre (gestionnaire **Place**)
- * **y** : offset écran vertical l'ancre (gestionnaire **Place**)
- * **rely** : offset relatif vertical dans le conteneur de l'ancre (gestionnaire **Place**)
- * **bordermode** : indique s'il faut inclure (**OUTSIDE**) ou exclure (**INSIDE** / défaut) la bordure du conteneur dans le calcul de taille et position pour le gestionnaire **Place**.

6. les attributs `default` et `state`

Les widget de commande, comme `Button`, `Checkbox`, `Entry`, `Radiobutton` etc. peuvent avoir trois états :

Ils peuvent être vivants/normaux, vivants/actifs, ou paralysés. À chacun de ces états correspond un aspect particulier du widget.

* `state` : l'attribut `state` a lui aussi trois valeurs : "`normal`", "`active`" et "`disabled`". Le plus souvent, le passage à "`active`" se fait automatiquement lorsque la souris survole le widget et l'attribut `state` n'a pas lieu d'être changé par programme pour cela.

* L'attribut `default` n'existe plus que pour les boutons dans `tkinter` 8.5. Et il a changé de signification ; il a deux valeurs utiles "`normal`" et "`active`" correspondant à une bordure en creux.



`tkinter` connaît des constantes de circonstance :

```
ACTIVE = "active"
```

```
NORMAL = "normal"
```

```
DISABLED = "disabled"
```

L'attribut "`readonly`" n'a pas de constante associée.

7. l'attribut `command`.

Cet attribut est partagé par les widgets qui réagissent à **une cause extérieure** : le bouton cliqué, la checkbox cochée ou décochée, le radio-bouton inversé etc. Dans ce cas, les widgets disposent de l'attribut `command` qui peut prendre comme valeur soit une fonction nommée et dans ce cas le nom de la fonction lui est attribué (attention, ce n'est pas une chaîne mais un identificateur) soit une lambda-fonction.

```
command = maFonction
```

```
command =self.maMethode
```

```
command = lambda x : monExpression
```

- Pour un bouton, la fonction nommée ne peut pas contenir de paramètre non initialisé. Elle est appelée sans paramètre valeur lorsque la commande s'active.

- On rappelle qu'une lambda-fonction peut comporter un ou plusieurs paramètres et une seule expression. Elle retourne le résultat de l'évaluation de l'expression (éventuellement rien). La lambda fonction peut être une façon d'appeler un gestionnaire, en lui passant des paramètres particuliers à l'instance d'appel.

- pour d'autres widgets comme la barre de scroll, les nombre de paramètres passé est codifié. Cela sera détaillé pour chaque cas d'espèce.

8. attributs de désaffectation (`disable`)

Les widgets de saisie de données peuvent être temporairement désaffectés, indiquant par là qu'ils ne peuvent être utilisés, mais tout en restant présents à l'affichage. Celui-ci est cependant marqué par des couleurs particulières, qui montrent à l'utilisateur qu'il ne sont pas en activité, où qu'ils sont en consultation seulement (`readonly`).

* `disabledbackground` : la couleur de fond en cas de non activité.

* `disabledforeground` : la couleur de caractères en cas de non activité.

* `readonlybackground` : le widget qui était habituellement en saisie clavier avec écho (ce que l'on a frappé au clavier apparaît à l'écran), a perdu sa capacité de saisie, ce qui n'empêche pas l'affichage, la sélection et le coller de sélection.

tk06: les méthodes partagées

Le fait pour un objet de **tkinter** d'être un **widget** implique que l'instance partage toutes les méthodes de la classe **Widget**. On a déjà vu une partie de ces méthodes pour le placement ou les méthodes touchant aux attributs. On les rappellera pour mémoire.

Il est important aussi de savoir qu'il n'y a pas de fonctions globales spécifiques de l'interface graphique dans **tkinter**. Toutes ces fonctions sont donc attachées à un widget, aussi étonnant que cela puisse paraître pour des fonctions concernant la temporisation, les caractéristique de l'écran ou simplement le codage des couleurs.

1. méthodes portant sur les attributs

1.1. méthodes de configuration

On revoie au chapitre **tk05 §1 attributs** pour plus de détails.

- * **config(options)**, **configure(options)** : méthode qui modifie des options du widget
- * **config()**, **configure()** : sans argument, la méthode retourne un dictionnaire de tous les attributs de **constructeur** d'un widget.
- * **pack_info()**, **grid_info()** et **place_info()** : retournent un dictionnaire de tous les attributs de **placement** d'un widget ; ce dictionnaire est vide si le gestionnaire correspondant n'est pas utilisé.
- * **cget(clef)** : retourne une **chaîne de caractère** donnant la valeur actuelle ; attention, un entier ou un flottant sont convertis en chaîne ou objet interne, convertible en chaîne par **str()**. Il peut y avoir difficulté avec **"height"** par exemple, ou le retour est un objet interne (voir **PanedWindow**) à transformer par **str()** puis **int()** si on veut par exemple un nombre de pixels. Les widgets étant subscriptable pour les attributs, on utilise de façon équivalente la notation avec crochets (**widget.cget(clef)** équivaut **widget["clef"]**)
- * **keys()** : retourne une liste des clefs des attributs disponibles.
- * **image_name()** : énonce la suite des noms d'image (il s'agit des noms internes donnés par l'application) dans l'application du widget dans une chaîne.
- * **image_types()** : énonce de même les types utilisés(**bitmap** ou **photo**)

1.2. méthodes d'options de classes

On peut modifier les valeurs par défaut des attributs de widget.

Pour cela il faut décrire l'option dans le format **Xdefaults**, qui sert à coder les options globales.

Le format est le suivant **"*classe*option"**. Avec **tkinter** on a par exemple **"*Button*font"** ou **"*Toplevel*background"**. On peut définir des classes non standard avec l'attribut **class** (écrire **class_**). Voir chapitre **tk05 §2**

- * **option_add(option, valeur, priority=None)** : **option** est l'option au format **Xdefault** et **valeur** la valeur de l'option :

exemple :

```
fenPrincipale.option_add("*Frame*background", "red")
```

- * **option_clear()** : supprime toutes les options utilisateur ; retourne aux valeurs par défaut.
- * **option_get(nom_option, classe_widget)** : la méthode retourne la valeur de l'option.
- * **option_readfile(fileName, priority=None)** : permet d'ajouter des options enregistrées dans un fichier au format **Xdefault**. Chaque ligne comporte une option suivie de sa valeur :

par exemple : `*Frame*background red`

La priorité est un paramètre qui peut prendre les valeurs None, 20, 40, 60, 80, qui signifient : , *propriété globale par défaut du widget, propriété s'il y a un fichier d'option au chargement de l'application, propriété qui provient d'un fichier appelé dans l'application, niveau utilisateur*. Dans le cas où il y a concurrence des valeurs d'option, c'est celle de plus haute priorité qui est prise. Par défaut, avec `option_add()` la valeur est 80. Il est toujours possible de modifier les options d'une instance, qui devient alors prioritaire, mais pour l'instance uniquement.

2. méthodes concernant les événements

Les méthodes suivantes font l'objet d'une notice dans le chapitre **tk07 événement**.

`bind, bind_all, bind_class unbind, unbind_all, unbind_class, bindtags, event_add, event_delete, event_generate, event_info`

3. méthodes de presse-papier

* `clipboard_get()` : retourne les données du presse_papier de Tk

* `clipboard_append(text)` : ajoute le texte `text` au contenu du presse papier de Tk

* `clipboard_clear()` : vide le presse papier

* `selection_clear()` : supprime une éventuelle sélection.

* `selection_get()` : s'il y a une sélection de texte sur le widget, retourne cette sélection. Lève une erreur sinon.

* `selection_own()` : le widget d'appel devient propriétaire de la sélection actuelle.

* `selection_own_get()` : retourne la sélection dont le widget est propriétaire. Lève une erreur sinon.

4. méthodes relatives aux placements

4.1. gestionnaires de placement

`grid, column_configure, row_configure, grid_forget, grid_remove, grid_propagate, pack, pack_forget, place, place_forget,`

Ces méthodes font l'objet d'une notice dans le chapitre **tk04 Géométries**

* `winfo_ismapped()` : le widget d'appel est soumis à un gestionnaire de placements, ainsi que tous ceux qui le contiennent jusque sa fenêtre de haut niveau. C'est évidemment le cas si celle-ci est affichée ; dans le cas contraire, invoquer `update_idletasks()`.

* `winfo_vewable()` : la méthode retourne un booléen qui indique si le widget d'appel ainsi que la chaîne de ses propriétaires est mappée.

* `winfo_manager()` : si le widget n'a pas de gestionnaire de géométrie, la méthode retourne une chaîne vide. Dans le cas contraire, c'est une des chaînes suivantes : "`grid`", "`pack`", "`place`", "`canvas`" ou "`text`". Une fenêtre de haut niveau retourne "`wm`".

* `pack_info()`, `grid_info()` et `place_info()` : voir section 1

* `winfo_geometry()` : retourne la chaîne de géométrie de la fenêtre de haut niveau contenant le widget d'appel. Cette donnée n'est à jour que si aucun calcul latent n'est en cours. Si l'application n'est pas affichée, la valeur peut ne pas être correcte. On peut forcer l'exécution des calculs latents avec `update_idletasks()`. On peut aussi retarder l'appel avec `wait_visibility()`.

4.2. empilement des widgets

* `lift()`, `lower()` : Voir le chapitre **tk04 Géométries**

* `transient(master)` : voir le chapitre **tk08 §3.4 Tk et Toplevel**

* `winfo_geometry()` : Voir le chapitre **tk04 Géométries**

5. mesures relatives au widget

- * `winfo_height()` : la méthode retourne la hauteur du widget tel qu'il est ou sera affiché. Cette donnée n'est à jour que si aucun calcul latent n'est en cours : lorsque le widget est affiché, c'est sûr. On peut forcer les calculs de géométrie (ceux qui servent à l'affichage après négociation avec les gestionnaires de placement) en appelant la méthode `update_idletasks()`.
- * `winfo_width()` : la méthode retourne la largeur du widget tel qu'il est ou sera affiché. Cette donnée n'est à jour que si aucun calcul latent n'est en cours. On peut forcer les calculs de géométrie (ceux qui servent à l'affichage après négociation avec les gestionnaires de placement) en appelant la méthode `update_idletasks()`.
- * `winfo_x()` : la méthode appliquée à un widget `w` retourne l'offset horizontal (en pixels) de `w` par rapport à son propriétaire. Le calcul se fait sur l'extérieur de la bordure de `w`. Voir les remarque sur `winfo_with()` qui s'appliquent ici.
- * `winfo_y()` : la méthode appliquée à un widget `w` retourne l'offset vertical (en pixels) de `w` par rapport à son propriétaire. Le calcul se fait sur l'extérieur de la bordure de `w`. Voir les remarque sur `winfo-with()` qui s'appliquent ici.
- * `winfo_reqheight()` : retourne la hauteur requise pour un widget, celle en deçà de laquelle le widget ne peut être affiché en entier. Il s'agit de la dimension calculée avant qu'un gestionnaire de placement lui soit appliqué. La dimension réelle après affichage peut ne pas correspondre.
- * `winfo_reqwidth()` : retourne la largeur requise pour un widget, celle en deçà de laquelle le widget ne peut être affiché en entier. Il s'agit de la dimension calculée avant qu'un gestionnaire de placement lui soit appliqué. La dimension réelle après affichage peut ne pas correspondre.

exemple :

```
# label de la question
question = Label(self, font=Kt.fonte, text=message)
wr = question.winfo_reqwidth()
hr = question.winfo_reqheight()
```

- * `winfo_rootx()` : retourne l'offset écran du bord gauche du widget, bordure comprise. Demande que l'affichage soit effectif.
- * `winfo_rooty()` : retourne l'offset écran du bord haut du widget, bordure comprise. Demande que l'affichage soit effectif.
- * `winfo_containing(rootx, rooty, displayof=0)` : la méthode retourne le widget qui a le point de coordonnées écran `rootx` et `rooty` ; si `displayof` est `False`, la recherche se fait par rapport à la fenêtre `Toplevel` du widget d'appel ; sinon, par rapport à la racine de l'application. Retourne `None` éventuellement.

6. méthodes portant sur la console

- * `winfo_screenheight()` : cette méthode retourne la hauteur de l'écran en pixels. La forme de l'appel est donc `widget.winfo_screenheight()` ; ce qui suppose qu'au moment de l'appel il existe au moins un widget. Or, on peut avoir besoin de l'information dans un module, ou avant toute création de widget. On peut procéder ainsi pour créer une fonction `screenHeight()` :

```
import tkinter
```

```
def screenHeight () :
    return tkinter.Frame().wininfo_screenheight()
```

- * `wininfo_screenwidth()` : cette méthode retourne la hauteur de l'écran en pixels. Voir `wininfo_screenheight()` pour un appel indépendant de l'environnement.
- * `wininfo_screenmmwidth()` : cette méthode retourne la hauteur de l'écran en millimètres. Idem.
- * `wininfo_screenmmheight()` : cette méthode retourne la hauteur de l'écran en millimètres. Idem.
- * `wininfo_screenvisual()` : cette méthode retourne le type modèle de couleur par défaut (celui de l'écran). `truecolor` pour le mode trichrome 24 bits par pixel. Autres valeurs : `directcolor`, `grayscale`, `pseudocolor`, `staticcolor`, `staticgray`. Méthode devenue désuète.
- * `wininfo_rgb(nom_couleur)` : Voir le chapitre **tk01 §2 couleurs**.
- * `wininfo_depth()` : retourne le nombre de bits par pixels utilisés dans l'application. Voir chapitre **tk01 § 3.1.méthodes de widget**.
- * `wininfo_fpixels(dimension)` : transforme la dimension en pixels sans arrondi. Voir chapitre **tk01 §3 unités**
- * `wininfo_pixels(dimension)` : arrondit le résultat précédent.
- * `bell()` : la méthode provoque l'émission d'un bip.

7. méthodes de souris

- * `wininfo_pointerxy()` : la méthode retourne un tuple d'entier qui sont les coordonnées (offsets horizontal et vertical) de la souris dans la fenêtre `Tk` ou `Toplevel` du widget
- * `wininfo_pointerx()` : retourne l'offset horizontal de la souris
- * `wininfo_pointery()` : : retourne l'offset vertical de la souris

8. les boucles

En programmation, une boucle est une séquence exécutée de façon itérative. Une boucle mobilise le processeur : il est donc souhaitable d'avoir un moyen de l'interrompre, sinon c'est le plantage bien connu et redouté de la «boucle infinie». Cependant, les langages pour interfaces graphiques proposent des boucles indéfinies, mais qui ne sollicitent pas le processeurs de façon exclusive.

Il y a deux type de boucles :

- * le boucle principale du programme ; deux méthodes lui sont associées, `mainloop()` et `quit()`. La méthode `mainloop()` est bloquante c'est-à-dire qu'il faut un événement extérieur pour que **l'instruction qui suit son appel soit exécutée**, en l'occurrence l'appel de la méthode `quit()`.
- * les boucles locales. Une boucle locale est une boucle d'attente qui est bloquante en l'attente d'un événement qu'elle est capable de détecter, mais qui est «indolore» pour tout l'environnement. Les événements restent accessibles à la boucle principale :

Le blocage n'empêche pas le système de surveillance d'événement de fonctionner, ni les gestionnaires d'événement de s'exécuter, ni le gestionnaire de fenêtre de fonctionner.

- * `mainloop()` : Cette méthode peut être appelée depuis n'importe quel widget, même à l'intérieur d'un gestionnaire d'événement. Pour comprendre cette méthode, il faut rappeler que lors d'une application `tkinter` une boucle infinie est lancée, **la boucle principale**, et que cette boucle va se dérouler jusqu'à la destruction de la fenêtre principale (méthode `destroy()` appelée depuis la fenêtre principale). La méthode `mainloop()` ne fait qu'indiquer que désormais, le déroulement de l'application se résume à la surveillance des événements et l'exécution des gestionnaires associés. On peut appeler autant de fois que l'on veut la méthode `mainloop()`. Chaque appel de `quit()` fait sortir de la méthode `mainloop()` dans l'ordre

inverse des appels. Il est cependant rare que le besoin se fasse sentir de quitter la méthode `mainloop()`, sauf pour **mettre fin proprement à l'application**, ce qui explique la recommandation du chapitre 0.

* `quit()` : met fin à la méthode `mainloop` dernière appelée.

exemple :

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *

# fenêtre principale
fenPr = Tk()
fenPr.title("mainloop")
fenPr.protocol("WM_DELETE_WINDOW", lambda x=None : x)
maFonte = "Helvetica -25"
numAppel=-1
listAppel = ["premier", "deuxième", "troisième", "quatrième"]

xLabel = StringVar()
leLabel = Label(fenPr, textvariable=xLabel, font=maFonte)
leLabel.pack(pady=10, padx=10)

# bouton mainloop
def fnMainloop():
    global xLabel, numAppel
    if numAppel == 3 : return
    numLocal = numAppel = numAppel + 1
    xLabel.set(listAppel[numLocal]+" appel de [mainloop]")
    print (xLabel.get())
    fenPr.mainloop()
    xLabel.set(listAppel[numLocal]+" appel terminé")
    print (xLabel.get())
    numAppel -= 1

btMainloop = Button(fenPr, font=maFonte, text=" mainloop ",
                    command=fnMainloop)
btMainloop.pack(pady=10, padx=10, side=LEFT )
# bouton quit
btMainloop = Button(fenPr, font=maFonte, text=" quit ",
                    command=lambda v=None : fenPr.quit())
btMainloop.pack(pady=10, padx=10, side=LEFT )

fnMainloop()
fenPr.destroy()

# fichier tk06ex03.py
```

On a cliqué trois fois le bouton [mainloop], puis trois fois [quit]

premier appel de [mainloop]
deuxième appel de [mainloop]
troisième appel de [mainloop]
troisième appel terminé
deuxième appel terminé
premier appel terminé



- * `wait_variable(v)` : crée une boucle d'attente locale ; la séquence en exécution est stoppée tant que la variable `v` n'a pas été affectée (une affectation du type `v=v` est parfaitement valide). `v` doit être entier, flottant, chaîne, booléen.
- * `wait_visibility()` : méthode réservée aux widget `Tk` et `Toplevel`. Elle crée une boucle d'attente locale ; la séquence en exécution est stoppée tant que le widget n'est pas visible.
- * `wait_window()` : méthode réservée aux widget `Tk` et `Toplevel`. Elle crée une boucle d'attente locale ; la séquence en exécution est stoppée tant que le widget n'est pas détruit par `destroy()`.

exemple :

```
- - - - -  
# destruction de la fenêtre  
self.wait_window() # attend la fin  
Kt.flagUnique = True # flag signalant la destruction  
  
def btOui (self) :  
    Kt.flagChoix = True  
    self.destroy()
```

9. méthodes de gestion du déroulement de programme.

- * `destroy()` : détruit sans recours le widget d'appel.
- * `update()` : la méthode met à jour tout ce qui est latent : les gestionnaire d'interruption non encore appelés, calcul et tracé des widgets qui doivent être redessinés, réaffichage de l'application à l'écran.
Cette méthode doit être utilisée avec précaution, surtout si elle est appelée depuis un gestionnaire d'événements, car il est parfois difficile de prévoir ce qui va se passer (à cause des gestionnaire d'événement qui sont sollicités à l'affichage).
- * `update_idletasks()` : cette méthode ressemble à la précédente, sauf en ce qui concerne l'affichage et les événements ; en effet, si l'affichage est recalculé, il n'est pas sollicité et les gestionnaires d'événements éventuellement en attente ne sont pas exécutés. Cette méthode doit être utilisée chaque fois que l'on a besoin de données qui ne sont normalement calculées qu'à l'affichage de l'application (par exemple, les dimensions des composants d'une fenêtre popup non affichée).
Attention cependant à un forçage des calculs par `update_idletasks()` alors que tous les éléments contribuant au placement ne sont pas encore négocié, par exemple avant un appel de la méthode `transient()`. Le forçage peut perturber la géométrie des éléments.
- * `after(delay_ms, callback=None, *args)` : `delay_ms` st un entier et `callback` une fonction. La fonction `callback()` est appelée après au moins `delay_ms`

millisecondes, avec les arguments de `args` (arguments valeurs, séparés par une virgule). La fonction n'est appelée qu'une fois, mais peut être retardée par des traitements en cours. Lorsque la méthode `after()` est exécutée, elle retourne un identificateur `id` qui peut être utilisé par `after_cancel()`.

* `after_cancel(id)` : `id` est l'identificateur renvoyé par `after()`. La temporisation initiée par `after()` est abandonnée et la fonction `callback()` n'est pas appelée.

* `after_idle(func, *args)` : la fonction `func()`, avec les arguments `args` est appelée lorsque tous les gestionnaire d'événement en instance sont exécutés.

10. méthodes d'identification

Chaque widget a par défaut un nom unique dans l'application, chaîne qui peut être changé lors de la création du widget. Il dispose également d'un identificateur, qui est un entier. Il dispose également d'un `pathname` qui est la succession des noms de conteneurs lorsque l'on remonte du widget à l'application racine (le nom de celle-ci n'est pas affichée). Ces noms son séparés par un point et le `pathname` commence donc par un point !

* `winfo_id()` : la méthode retourne l'identificateur du widget d'appel

* `winfo_name()` : la méthode retourne le nom du widget d'appel

* `winfo_pathname(id, displayod=False)` : `id` est l'identificateur du widget dont on veut le `pathname`. Le widget d'appel est indifférent. Le `pathname` est parfois appelé le **nom complet** du widget, et il est aussi retourné par la fonction `str(widget)`. La forme de la chaîne est :

```
".nom_toplevel.nom_cont1.nom_cont2.nom_widget"
```

s'il y a 2 conteneurs. Le nom est par défaut la suite de chiffre donnée par le système ; si on a utilisé `name` dans le constructeur, c'est le nom utilisateur.

* `winfo_toplevel()` : la méthode retourne le `Toplevel` qui contient le widget d'appel.

* `winfo_parent()` : la méthode retourne le `pathname` du conteneur immédiat du widget d'appel. (**attention, ne retourne pas le parent !**)

* `winfo_children()` : la méthode retourne une liste des enfants du widget d'appel. **Cette fois, ce sont bien les widgets qui sont retournés !**

* `winfo_class()` : la méthode retourne le nom de la classe du widget.

* `nametowidget(pathname)` : retourne le widget dont le **nom complet** est `pathname`. Le paramètre doit correspondre à un nom existant, sinon il y a erreur.

exemple :

```
boutonNon = Button(self.cadreBoutons,name="btNon", text="non",
                  font=Kt.fonte, command=self.btNon)
self.boutonOui.focus_set()
idBtNon = boutonNon.winfo_id()
print ("id de boutonNon :", idBtNon)
print ("nom de boutonNon :", boutonNon.winfo_name() )
print ("pathname :", self.winfo_pathname(idBtNon))
print ("str de boutonNon :", str(boutonNon))
print ("pathname du toplevel de boutonNom :",
      boutonNon.winfo_toplevel())
print ("nom du toplevel de boutonNom :",
      boutonNon.winfo_toplevel().winfo_name())
infoParent = boutonNon.winfo_parent()
print ("parent de boutonNon :", infoParent,
      "de type :", type(infoParent))
```

```
for enfant in self.winfo_children() :
    print ("enfant du TopLevel :", enfant,
          "de classe :", enfant.winfo_class())
print ("classe de self :",self.winfo_class())
```

```
id de boutonNon : 67108905
nom de boutonNon : btNon
pathname : .25236560.25236880.btNon
str de boutonNon : .25236560.25236880.btNon
pathname du toplevel de boutonNom : .25236560
nom du toplevel de boutonNom : 25236560
parent de boutonNon : .25236560.25236880 de type : <class 'str'>
enfant du TopLevel : .25236560.25236816 de classe : Label
enfant du TopLevel : .25236560.25236880 de classe : Frame
classe de self : Toplevel
```

11. méthodes de focus

Un widget a le focus s'il est en mesure de **recevoir les événements du clavier**. Une façon d'obtenir le focus pour un widget est de presser la touche de tabulation (soit **tab**, soit **ctrl-tab** si le widget utilise en interne la touche **tab**, ce qui est le cas pour le widget **Text**, soit **shift-tab**). La tabulation fait tourner le focus entre les widgets de la fenêtre active **Tk** ou **Toplevel** dans l'ordre de leur création dans la fenêtre (l'ordre inverse avec **shift**). Pour activer une fenêtre, il suffit de la cliquer ; il n'y a qu'une fenêtre active dans une application, que l'on repère par la barre de titre qui est d'une tonalité en général plus lumineuse (cela dépend du système d'affichage). Dans cette fenêtre active, un seul widget peut avoir le focus.

Les widgets ont un attribut booléen **takefocus** qui spécifie si le widget est autorisé à prendre le focus ou pas. L'attribut **takefocus** peut être concédé à tous les widgets qui réagissent alors de façon identique à ceux ayant **takefocus** à **True** par défaut, lors de la prise ou la perte du focus.

Un widget focusable qui prend le focus active ses attributs **highlightcolor**, **highlightthickness**. Certains widgets sont prévus pour réagir «naturellement» au clavier : les boutons de commande (touche espace par défaut), les widgets de saisie, comme **Entry**, **Radio**, **Checkbox**, **Text** etc. Dans ce cas, les attributs **highlightbackground**, **highlightcolor** ont une valeur par défaut bien spécifique ; la valeur par défaut de **takefocus** est 1 (**True**) ; on peut retirer la possibilité d'avoir le focus en mettant le **takefocus** à 0. Tous les widgets ont une surbordure d'épaisseur **highlightthickness** et de couleur **highlightbackground**. Lors de la prise de focus, la couleur de cette bordure devient **highlightcolor**. Attention cependant : si un conteneur contient un widget qui a le focus, sa surbordure a la couleur **highlightcolor**.

- * **focus_displayof()** : la méthode retourne le widget qui a le focus sur le moniteur qui affiche le widget d'appel. Retourne **None** s'il n'y en a pas. La différence avec **focus_get()** est à rechercher si l'application s'affiche sur plusieurs moniteurs.
- * **focus_get()** : la méthode retourne le widget qui a le focus dans l'application (ce peut être la fenêtre active elle-même). Retourne **None** s'il n'y en a pas
- * **focus_set()** : Si l'application n'a pas le focus de saisie au clavier, elle enregistre l'ordre de transfert de focus. Si elle a le focus, ou quand elle le récupère, le widget d'appel prend le focus. Si le widget n'a pas l'attribut **takefocus**, cette prise de focus est temporaire.
- * **focus_force()** : la méthode impose le focus au widget d'appel, même si l'application n'a pas le focus de saisie. Méthode dangereuse qu'il vaut mieux oublier (voir la recommandation de **grab_set_global()**)

- * `focus_lastfor()` : la méthode retourne le dernier widget qui a eu le focus dans la fenêtre `Tk` ou `Toplevel` qui contient le widget d'appel. Retourne la fenêtre si aucun ne l'a eu.
- * `tk_focusFollowsMouse()` : Lors de l'appel, le focus se met à suivre la souris et donne le focus aux éléments survolés qui ont l'attribut `takefocus`. Attention, une fois posé, il n'y a pas de manière simple de désactiver cette méthode !
- * `tk_focusNext()` : retourne le widget qui suit normalement (touche **tab**) le widget qui a le focus
- * `tk_focusPrev()` : retourne le widget qui précède normalement (touche **shift-tab**) le widget qui a le focus.

12. méthodes d'accaparement

Le `grab` (*accaparement*) est un mode de détournement des événements : tous les événements de l'application (si le `grab` n'est pas global à la session) sont détournés vers la fenêtre qui a le `grab`. La boucle principale ne tourne donc que pour la fenêtre accapareuse.

- * `grab_set()` : pose le `grab` sur le widget d'appel. Si un autre widget a déjà le `grab`, il est dépossédé.
- * `grab_status()` : un widget qui n'a pas le `grab` a un statut `None`. Si le `grab` a été posé par `grab_set()`, il a pour statut `"local"`. Sinon le statut est `"global"`.
- * `grab_current()` : retourne le widget qui a le `grab`.
- * `grab_release()` : supprime le `grab` sur le widget s'il le possède.
- * `grab_set_global()` : la méthode pose le `grab` sur le widget, mais étend son emprise sur toutes les applications graphiques de la session. Cette méthode est particulièrement dangereuse et ne doit être utilisée qu'à bon escient.

exemple :

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
"""
ILLUSTRATION DU grab_set. Le script a le scénario suivant :
***** quatre fenêtre Toplevel sont créées, et
mises dans une liste, listeGrab(). La dernière est doté du grab.
Lors de la fermeture d'une de ces fenêtres (celle qui a le grab !),
le grab passe à la fenêtre précédente. On peut tester la réactivité
des fenêtres en les survolant. Pour éviter la fermeture par icône
d'une fenêtre qui n'a pas le grab (possible avec grab "local") et
provoquer ensuite un erreur, on a protégé la fermeture par icône.
La fenêtre principale n'est pas visible.
"""
from tkinter import Tk, Toplevel, Button
from tkinter.messagebox import askyesno, showinfo

# fenêtre principale
fenPr = Tk()
listeGrab = []
Toplevel.poserGrab = Toplevel.grab_set # Toplevel.grab_set_global
flagAttention = False

class FenetreGrabable (Toplevel):
    def __init__(self, master, numero) :
        Toplevel.__init__(self, master)
        self.numero = numero
```

```

self.title("fenêtre "+str(numero))
bt = Button(self, text=" supprimer ", command=self.detruire)
bt.place( relx=0.5, rely=0.5, anchor = 'c')
self.bind('<Enter>', self.rougir)
self.bind('<Leave>', self.griser)
self.protocol('WM_DELETE_WINDOW', self.detruire)

def detruire(self, event=None) :
    global listeGrab, fenPr, flagAttention
    # section inutile pour grab_set_global()
    if flagAttention :
        return
    if self.grab_current().grab_status() == "local" :
        if self.grab_current() != self :
            flagAttention = True # n'ouvrir qu'une fois showinfo
            showinfo("attention !", "une fenêtre ne doit pas être\n"
                    +"fermée si une autre fenêtre\n"
                    +"de l'application a le grab ")
            flagAttention = False
        return
    # fin de section pour grab_set()
    self.destroy()
    if self.numero == 0 :
        fenPr.quit()
    else :
        listeGrab[self.numero-1].poserGrab()

def rougir(self, event) :
    self.config(background='red')

def griser(self, event) :
    self.config(background='gray80')

for t in range(4) :
    listeGrab.append(FenetreGrabable(master=fenPr, numero=t))

fenPr.update_idletasks() # utile pour grab_set_global uniquement
listeGrab[3].poserGrab()
fenPr.withdraw() # fenêtre principale invisible
fenPr.mainloop()

# tk06ex01.py

```

tk07 : Événements

Un événement est quelque chose qui se passe sur l'ordinateur : frappe clavier, action sur la souris, modification de la présentation d'une application etc. Certains événements sont traités par le BIOS ou le système d'exploitation et en restent là : ils ne propagent pas jusque l'application. Pour les événements dont l'information parvient à l'application, **c'est la boucle principale** qui les intercepte, les empile et la machine **Python/tkinter** est amenée à analyser comment réagir face à chacune d'elle ; le plus souvent, rien n'a été prévu, et l'événement est perdu.

L'objet de ce chapitre est d'analyser

- * comment l'événement «circule dans l'application»
- * comment déclarer qu'un événement précis doit être pris en compte
- * quel traitement doit lui être associé ; en général, c'est l'exécution d'une fonction, appelé un **gestionnaire d'événement**

La déclaration et le traitement sont (presque) toujours associés à un widget ; on dit que ce couple déclaration/gestionnaire est lié au widget. La liaison se réalise par une méthode `bind()`. Mais certains widgets n'ont pas besoin de la méthode `bind()` car ils ont un attribut `command` qui est associé à une fonction gestionnaire d'événement, pour **un événement par défaut** : c'est le cas du bouton pour l'événement de clic de la souris.

Quelques événements sont dérogatoires, et on les a déjà signalés : ce sont les événement lié à un objet `StringVar`, `IntVar`, `DoubleVar`, et les événements des boucles locales `wait_window`, `wait_variable`, `wait_visibility` ou la boucle principale `mainloop`. Ce ne sont pas eux qui sont abordés dans ce chapitre. Ni les particularités qui seront signalées pour les widgets `Canvas` et `Text`.

1. les événements reconnus

1.1. événements clavier

- * `KeyPress`, `Key` : Une touche du clavier est enfoncée
- * `KeyRelease` : Une touche du clavier est relâchée

1.2. événements souris

- * `ButtonPress`, `Button` : Un bouton de la souris est enfoncé.
- * `ButtonRelease` : Un bouton de la souris est relâché.
- * `Motion` : La souris est déplacée.
- * `Enter` : Le pointeur de souris vient de pénétrer dans une partie visible du widget.
- * `Leave` : Le pointeur de souris vient de sortir d'une partie visible du widget.
- * `MouseWheel` : La molette de la souris a été activée. Cet événement n'est reconnu que sous **Windows** ; sous **Linux**, il est reconnu comme bouton enfoncé (voir §2.4)

1.3. événements fenêtre

- * `Visibility` : lorsqu'une partie au moins du widget devient visible à l'écran
- * `Unmap` : le widget perd sa visibilité (par exemple avec `grid_forget()`)
- * `Map` : le widget est mappé, c'est à dire qu'il a activé un gestionnaire de placement.
- * `Expose` : la visibilité du widget change dans le sens d'une meilleur exposition.
- * `Configure` : la taille du widget change.
- * `FocusIn` : le widget prend le focus.
- * `FocusOut` : le widget perd le focus
- * `Circulate`, `Gravity`, `Reparent` : événements relatifs à l'interface homme/machine. Reconnus dans le lexique, mais non implémentés sur toutes les plate-formes. Non documentés.

* **Colormap** : La palette de l'écran a changé. Désuet.

* **Property** : Les propriétés de l'interface homme/machine sont changées (par exemple lors de la création d'une fenêtre) ou sont supprimées.

* **Destroy** : Le widget est détruit.

* **Activate** : Le widget passe de l'état non actif à l'état actif (ex : un bouton). S'applique aux widgets ayant un attribut state ayant les valeurs suivantes : "normal", "active". L'état "normal" est l'état du widget qui peut réagir à un survol de souris par exemple. L'état "active" est signalé par un changement de couleur du widget.

* **Deactivate** : Le widget passe de l'état actif à l'état non actif.

2. la notion de séquence

2.1. écrire une séquence

La séquence est une définition opérationnelle d'un événement ; par exemple, elle précise à quel caractère clavier on doit se référer, ou à quel bouton de la souris.

La séquence est **une chaîne** de patrons d'événements concaténés et l'événement résultant est la succession chronologique d'événement élémentaires décrit par les patrons. Un patron d'événement a la structure suivante :

<MODIFICATEUR-MODIFICATEUR-TYPE-DÉTAIL>

Un **MODIFICATEUR** est un de mots suivants :

Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5, Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5, Triple, Mod1, M1.

Meta est une touche dotée d'une logo, Apple ou Windows. **Mod1, M1** est la touche **Meta 1**, **Mod2, M2** la touche **Meta 2** etc. si ces touches existent.

Un **TYPE** est un des mots suivants :

Activate (type 36), Enter (type 7), Map (type 19), ButtonPress (type 4), Button (type 4), Expose (type 12), Motion (type 6), ButtonRelease (type 5), FocusIn (type 9), MouseWheel (type 38), Circulate (type ?), FocusOut (type 10), Property (type ?), Colormap (type ?), Gravity (type ?), Reparent (type ?), Configure (type 22), KeyPress (type 2), Key (type 2), Unmap (type 18), Deactivate (type 37), KeyRelease (type 3), Visibility (type 15), Destroy (type 17), Leave (type 8)

Un **DÉTAIL** est

- un numéro de bouton pour **ButtonPress, ButtonRelease** (1 : à gauche ; 3 : à droite)

- un **keysym** (symbole de touche) pour **KeyPress** (qui peut être omis ou remplacé par **Key**) et **KeyRelease**. Voir le tableau en fin de chapitre pour les codes de **keysym**

Les **keysym** sont définis pour **Tcl/Tk** ; on trouvera sur le site officiel de **Tcl/Tk** les **keysym**, ainsi que le **keysym_num** associés (très nombreux).

<http://www.tcl.tk/man/tcl8.4/TkCmd/keysyms.htm>

Les codes utiles sont cependant moins nombreux ; voir en annexe une sélection pratique.

2.2. exemples :

<Control-Button-1> touche Ctrl enfoncé, bouton 1 pressé.

<A>, **<Key-A>**, **<KeyPress-A>** : presser la touche A

<Alt-A> : presser la touche A conjointement à la touche Alt (**KeyPress** est omis).

Ceci fonctionne bien pour la majorité des caractères : **<a>**, **<H>** sont valides ; mais attention **<1>**, **<2>** ... réfèrent non pas un caractère mais un bouton de souris. Si on veut se faire comprendre, on n'omet pas le mot **Key**.

<KeyPress-H> : la touche H a été appuyée.

<Control-Shift-KeyPress-H> : touches Ctrl, Shift et H pressées.
<B1-Motion> : déplacement de la souris avec le bouton 1 enfoncé (gauche)
<ButtonRelease-1> : le bouton 1 a été relâché ; souvent préférable pour le clic.
<Button-1>, <ButtonPress-1>, <1> : même chose pour bouton 1 pressé.
<Double-Button-1> : double clic.
<Return><KP_Enter> : touche entrée normale puis celle du pavé numérique.

2.3. événements virtuels

Un patron d'événement peut également décrire un événement virtuel ; dans ce cas le patron a la forme :

<<UNE CHAÎNE>>

* Un événement virtuel est un événement qui regroupe en un seul plusieurs événements alternatifs ; pour réaliser un événement virtuel, on utilise la méthode `event_add()`, selon le modèle suivant :

```
widget.event_add("<<evtVirtuel>>", "<Button-1>", "<Enter>", "<2>")
```

Le `widget` est une instance valide quelconque ; le premier argument est une chaîne qui identifie l'événement virtuel ; la suite est constituée de chaînes qui sont des descripteurs d'événement.

La méthode `event_info()` retourne les descripteurs des événements virtuels de l'application. On notera que l'application possède déjà en interne des événements virtuels (comme le couper/coller).

2.4. le cas de la molette

La molette est traitée différemment sous Windows ou Linux. Avec **Windows**, le modificateur est **MouseWheel** alors qu'il faut utiliser **Button-4** et **Button-5** sous **Linux**. Sous Windows, `delta` est incrémenté ou décrémenté de 120 pour un pas de molette ; sous Linux, `delta` reste 0, et on détecte le mouvement de molette avec le numéro de bouton. On peut lever la difficulté en créant un événement virtuel et un traitement convenable des événements :

```
widget.event_add("<<molette>>", "<MouseWheel>", "<Button-4>", "<Button-5>")
. . .
def molette (event):
    compteur = 0 # global ???
    if event.num == 5 or event.delta == -120:
        compteur = 1
    if event.num == 4 or event.delta == 120:
        compteur = -1
```

3. gestionnaire d'événement

3.1. fonction gestionnaire

Le gestionnaire d'événement est la fonction appelée lorsqu'un événement se produit sur un widget et défini dans un lien pour cet événement.

- **Si la liaison se fait par défaut**, le gestionnaire peut agir sans avoir besoin de plus de détail. Autrement dit, le gestionnaire par défaut doit avoir de paramètres formels non initialisés correspondant strictement avec la norme définie. Pour un bouton, il n'y a pas de paramètre passé. Rien n'empêche cependant qu'il y ait des paramètres initialisés.

- **si la liaison se fait par la méthode `bind()`**, alors le gestionnaire d'événement a besoin de renseignements complémentaires : ceux si sont fournis sous forme d'un objet comme premier argument de la fonction gestionnaire d'événement. Mais elle ne fournit rien d'autre que ce qui est dans l'objet retourné. En pratique, le gestionnaire doit comporter au moins un paramètre formel (initialisé ou non) et il ne peut y avoir qu'un paramètre formel non initialisé. (à la déclaration, ne pas oublier le `self`, si le gestionnaire est une méthode d'instance)

3.2. l'objet Event

L'objet `event` passé au gestionnaire d'événements appartient à la classe `Event` dont la description suit. On rappelle que `event.__dict__` permet de récupérer l'événement sous forme d'un tableau, que l'on peut rendre énumérable avec la méthode `iteritems()`.

champ	signification	type d'événement qui gère ce champ
<code>serial</code>	numéro de série de l'événement (incrémenté à chaque événement)	
<code>num</code>	numéro du bouton de la souris	<code>ButtonPress</code> , <code>ButtonRelease</code>
<code>focus</code>	dit si le widget a le focus	<code>Enter</code> , <code>Leave</code>
<code>height</code>	hauteur de la fenêtre concernée	<code>Configure</code> , <code>Expose</code>
<code>width</code>	largeur de la fenêtre concernée	<code>Configure</code> , <code>Expose</code>
<code>keycode</code>	keycode de la touche clavier	<code>KeyPress</code> , <code>KeyRelease</code>
<code>state</code>	état de l'événement souris (nombre) tableau ci-dessous pour les valeurs.	<code>ButtonPress</code> , <code>ButtonRelease</code> , <code>Enter</code> , <code>KeyPress</code> , <code>KeyRelease</code> , <code>Leave</code> , <code>Motion</code>
<code>state</code>	état de l'événement (chaîne)	<code>Visibility</code>
<code>time</code>	quand l'événement s'est produit	
<code>x</code>	position horizontale de la souris	
<code>y</code>	position verticale de la souris	
<code>x_root</code>	position horizontale de la souris sur l'écran	<code>ButtonPress</code> , <code>ButtonRelease</code> , <code>KeyPress</code> , <code>KeyRelease</code> , <code>Motion</code>
<code>y_root</code>	position verticale de la souris sur l'écran	<code>ButtonPress</code> , <code>ButtonRelease</code> , <code>KeyPress</code> , <code>KeyRelease</code> , <code>Motion</code>
<code>char</code>	caractère pressé (chaîne) ; pour les touches imprimables	<code>KeyPress</code> , <code>KeyRelease</code> peut dépendre du système.
<code>send_event</code>	booléen	
<code>keysym</code>	<code>keysym</code> de l'événement (chaîne)	<code>KeyPress</code> , <code>KeyRelease</code>
<code>keysym_num</code>	<code>keysym</code> de l'événement (nombre)	<code>KeyPress</code> , <code>KeyRelease</code>
<code>type</code>	type d'événement (nombre)	
<code>widdget</code>	widget de détection de l'événement	
<code>delta</code>	delta de la molette de la souris	molette de la souris sous Windows

les masques de `state` pour l'événement souris

Modif.	Shift	CapsLock	Ctrl	Att g.	NumLock	Alt d.	Bt1	Bt2	Bt3
masque	<code>x0001</code>	<code>x0002</code>	<code>x0004</code>	<code>x0008</code>	<code>x0010</code>	<code>x0080</code>	<code>x0100</code>	<code>x0200</code>	<code>x0400</code>

Les masques sont des hexadécimaux à ajouter pour obtenir le `state`. Ils permettent de savoir les

touches *meta* pressées en même temps que la souris pour déclencher un événement (exemple : `<Shift-Button>`). Pour un exemple complet voir chapitre 21 § 2.

3.3. cas mixtes

Il peut arriver qu'un même gestionnaire serve pour un événement par défaut et un événement lié. Par exemple, un bouton peut être programmé par défaut pour appeler un gestionnaire, et que le même gestionnaire soit appelé par les touches *Entrée* (celle du clavier de base et celle du pavé numérique), pour lesquelles deux liaisons s'imposent : dans ce cas, il convient d'initialiser à `None` (ou toute autre chose) le paramètre formel obligatoire et de n'utiliser éventuellement l'objet `Event` que si c'est le clavier qui a été sollicité.

Il arrive aussi que l'on puisse avoir un nombre variable de valeurs passées en paramètres. Il faut alors prévoir suffisamment de variables pour le cas le plus extrêmes, les initialiser pour ménager les cas à plus petits nombres de paramètres ; l'utilisation d'un paramètre de type `*arg` peut être utile. Voir au chapitre **tk12, § 5** un exemple de cette situation très particulière.

4. la liaison

4.1. les bindtags

* Chaque widget comporte un tuple de **tags de liaison** : les tags du tuple sont les noms des genres de liaison qui peuvent être faites à partir de lui : par défaut, on a lui-même, sa classe, sa fenêtre `Toplevel`, et tout ("all"). Lorsqu'un événement se produit sur un widget, `tkinter` cherche dans l'ordre s'il y a une liaison sur lui-même, puis sa classe, puis son `Toplevel` (sauf si la classe est `Toplevel`) ou effectivement «tout».

exemples :

```
('.popup', 'Toplevel', 'all')
('.monCadre.monBouton', 'Button', '.', 'all')
```

Pour la clarté de l'exposé, on a nommé explicitement les widgets (*cadre*, *monBouton*). Le point tout seul désigne la racine, qui n'a pas de nom redéfini.

* `bindtags()` : cette méthode de widget permet de connaître les **bindtags** associés sous forme d'un tuple.

* On peut enrichir la liste des **bindtags** d'un widget en ajoutant une (ou plusieurs) «classe utilisateur», en l'occurrence une simple chaîne la désignant.

La méthode est la suivante :

* `bindtags(nouvelle liste)`: comme il vaut mieux ne pas perdre les anciens **bindtags** (utilisable pour d'autres liaisons), on procède en général ainsi :

```
widget.bindtags( ("nom de classe utilisateur",) + widget.bindtags() )
```

Attention aux parenthèses !

exemple :

```
bouton = tkinter.Button(cadre, name= "monBouton", text="un bouton")
print (bouton.bindtags())
# ('.monCadre.monBouton', 'Button', '.', 'all')
bouton.bindtags( ("ClasseUtile",) + bouton.bindtags() )
print (bouton.bindtags())
# ('classeUtile', '.monCadre.monBouton', 'Button', '.', 'all')
```

4.2. les niveaux de liaison

Il y a quatre niveaux de liaison de l'événement à des widgets

1. la liaison est faite pour un seul widget :

```
def bind(self, sequence=None, func=None, add=None):
```

`add` : indique, au cas où il y aurait plusieurs gestionnaires sur le même événement du même widget si

la fonction doit être ajoutée aux autres, ou si elle les écrase ; par défaut : écrasement.

La syntaxe est de la forme :

```
widget.bind(sequence, fonction, add=None)
```

La séquence doit être **quotée** : c'est une chaîne ; fonction est une référence de fonction (surtout ne pas parenthéser). Une **lambda fonction** est acceptée car il s'agit bien de code Python.

2. la liaison est faite pour la classe d'un widget :

```
def bind_class(self, className, sequence=None, func=None, add=None):
```

C'est ici que le add prend tout son sens !

La syntaxe est de la forme :

```
widget.bind_class(className, sequence, fonction, add=None)
```

attention : n'importe quel widget fait l'affaire, même s'il n'appartient pas à la classe !

3. la liaison est globale

```
def bind_all(self, sequence=None, func=None, add=None):
```

La syntaxe est la suivante :

```
widget.bind_all(sequence, fonction, add=None)
```

La liaison est faite sur toute l'application.

Exemple : l'événement `Print` (la touche **Imprime écran** est pressée)

4. la liaison est faite pour une classe créée avec bintags:

La syntaxe est celle des classes :

```
widget.bind_class(className, sequence, fonction, add=None)
```

Mais cette fois `className` est une classe utilisateur.

exemple :

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *
from tkinter.messagebox import askyesno

# fenêtre principale
fenPr = Tk()
fenPr.title("bindtags")

# quitter le logiciel
def quitter():
    reponse = askyesno("terminer le script",
        "Voulez-vous réellement terminer\u00a0? \n cliquer «oui» pout finir")
    if reponse :
        fenPr.quit()

# bouton quitter
btQuitter = Button(fenPr, font=maFonte, text="QUITTER", command=quitter)
btQuitter.pack(pady=20)

# capture de l'icone de la barre de titre
fenPr.protocol("WM_DELETE_WINDOW", quitter)
maFonte = "Helvetica -25"

# le gestionnaire d'événements
```

```

def jeClique(event) :
    print("\nvous avez cliqué un widget :\nle nom est :",
          event.widget.winfo_name(), "\nle bintags est :",
          event.widget.bindtags() )

# les widgets
fenPr.config(bg="#a0a0ff")
laFrame = Frame(fenPr, bg="yellow", name="nomCadre")
leLabel = Label(laFrame, name="nomLabel", text="une étiquette",
                font=maFonte, bg="#8080ff")
leBouton = Button(laFrame, name="nomBouton", text="un bouton",
                  font=maFonte, bg="red")
laFrame.pack(pack=60, pady=60)
leLabel.pack(pack=20, pady=20)
leBouton.pack(pack=20, pady=20)
# bind() et bindtags ()
for widg in [fenPr, laFrame, leBouton, leLabel] :
    widg.bindtags(("classeUtile",) + widg.bindtags())
fenPr.bind_class("classeUtile", "<Button>", jeClique)

# boucle de la fenêtre principale
fenPr.mainloop()
fenPr.destroy() # par précaution

# fichier tk07ex00.py

```

le résultat du clic sur les quatre composants

```

vous avez cliqué un widget :
le nom est : nomBouton
le bintags est : ('classeUtile', '.nomCadre.nomBouton', 'Button', '.',
'all')

vous avez cliqué un widget :
le nom est : nomLabel
le bintags est : ('classeUtile', '.nomCadre.nomLabel', 'Label', '.',
'all')

vous avez cliqué un widget :
le nom est : nomCadre
le bintags est : ('classeUtile', '.nomCadre', 'Frame', '.', 'all')

vous avez cliqué un widget :
le nom est : tk
le bintags est : ('classeUtile', '.', 'Tk', 'all')

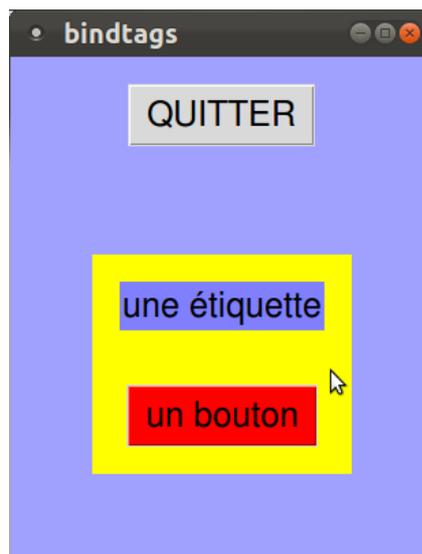
```

note : Avec **tkinter**, il n'y a pas de bouillonnement, comme en JavaScript/DOM.

Par exemple, le bouton est contenu dans un cadre, lui-même contenu dans la fenêtre principale.

Avec le bouillonnement, un gestionnaire d'événement pour un événement **evt** (clic de souris par exemple) lié à la fenêtre principale va traiter l'événement **evt** même s'il est détecté sur le bouton (ou l'étiquette) ou le cadre.

Ici, si on veut que **evt** soit traité de façon identique pour le bouton (l'étiquette), le cadre et la fenêtre principale ; il faut passer par la génération d'une classe dans des **bindtags**.



4.3. le retour de `bind()`

La méthode retourne un identificateur de liaison, qui peut être utilisé dans la suppression de la liaison pour éviter la fuite de mémoire.

Dans le cas où la séquence ou la fonction sont oubliés, le tuple des `bindtags` est retourné.

4.4. méthodes de suppression de liaisons

* `unbind(sequence, funcid=None)` : supprime la liaison correspondant à la séquence ; si on veut éviter les fuites de mémoire, `funcid` doit être initialisé avec la valeur de retour de la méthode `bind()` correspondante.

* `unbind_all(sequence)` : supprime la liaison universelle correspondant à la séquence.

* `unbind_class(className, sequence)` : supprime la liaison à la classe correspondant à la séquence.

* `event_delete(virtual, sequences)` : supprime des liaisons (celles concernées par `sequences`) pour un événement virtuel.

4.5. génération d'événement : `event_generate()`

Il peut arriver d'avoir besoin de **simuler un événement**, par exemple pour une initialisation. On dispose de la méthode `event_generate()` dont le premier paramètre obligatoire est une séquence, au sens de la section 1.

* `event_generate(sequence, options_clef_valeur)` : la méthode crée un événement décrit par la séquence, sur le widget d'appel, avec éventuellement des paramètres clef/valeur correspondant aux attributs d'un objet `Event`.

exemple :

```
def coordonnees(event) :  
    print ("où ? ",event.x,"",event.y)  
monCadre.bind("<Button>", coordonnees)  
  
monCadre.event_generate("<Button>", x=0, y=0)
```

L'appel dans l'application du `event_generate` crée un événement souris sur `monCadre` ; son gestionnaire qui affiche les coordonnées de cet événement, va donc afficher 0, 0

5. Les protocoles

5.1. les trois modes de liaison

On a déjà rencontré deux modes de liaison : la liaison par défaut, comme l'attribut `command` des widgets `Button`, et la méthode `bind()`. Il existe un troisième mode, la liaison pour des événements déclenchés par le gestionnaire de fenêtre (**window manager**). Ce mode de liaison est donc très très spécifique.

5.2. la syntaxe

`widget.protocol(nom, fonction)`

* `nom` peut être : `"WM_TAKE_FOCUS"`, `"WM_SAVE_YOURSELF"` ou `"WM_DELETE_WINDOW"`. Le paramètre `nom` (`name`) appellations d'événement :

`"WM_TAKE_FOCUS"` : l'événement se produit quand l'application reçoit le focus.

`"WM_SAVE_YOURSELF"` : l'événement se produit quand l'application devrait sauvegarder un «instantané» de son état de travail. Implémentation dépendant des plate-formes. Non documentée.

`"WM_DELETE_WINDOW"` : l'événement se produit lorsque l'on clique l'icône de fermeture dans la barre de titre d'une fenêtre `Toplevel`. Le protocole est une méthode de la fenêtre. C'est le seul protocole couramment utilisé.

* `fonction` (`func`) : nom du gestionnaire associé ; il n'y a pas d'objet `Event` fourni.

On peut récupérer la valeur du protocole par la méthode `widget.protocol(nom)`.

Pour inhiber l'iconisation, il suffit de mettre pour fonction une `lambda` fonction qui ne fait rien et retourne `None`, sur le modèle suivant :

```
widget.protocol("WM_DELETE_WINDOW", lambda : None)
```

pour s'y retrouver dans la gestion des événements	
un widget / un événement / un gestionnaire	
	liaison : <code>widget.bind (séquence, gestionnaire)</code> .
un widget / plusieurs événements qui se succèdent / un gestionnaire	
	concaténer les séquences des événements en un seul événement.
un widget / plusieurs événements en alternative / un gestionnaire	
	événement virtuel qui regroupe plusieurs événements en un seul.
un widget / un événement / plusieurs gestionnaires	
	option <code>add</code> de <code>bind()</code> mise à <code>True</code> .
plusieurs widgets / un événement / un gestionnaire	
	utiliser les <code>bindtags</code> .

annexe : les keysym

clavier de base

keysym	keycode	keysym_num	la touche
Alt_L	64	65513	Alt gauche
Alt_R	113	65514	Alt droit
BackSpace	22	65288	backspace
Cancel	110	65387	Pause (Break)
Caps_Lock	66	65549	CapsLock
Control_L	37	65507	Ctrl gauche
Control_R	109	65508	Ctrl droit
Delete	107	65535	Delete
Down	104	65364	↓ flèche bas
End	103	65367	end
Escape	9	65307	esc
Execute	111	65378	Impr écran (SysReq)
F1	67	65470	touche fonction F1
F2	68	65471	touche fonction F2
F _i	66+i	65469+i	touche fonction F _i
F12	96	65481	touche fonction F12
Home	97	65360	home
Insert	106	65379	insert
Left	100	65361	← flèche gauche
Linefeed	54	106	Linefeed (control-J)
Next	105	65366	PageDown
Num_Lock	77	65407	NumLock
Pause	110	65299	pause
Print	111	65377	PrintScrn
Prior	99	65365	PageUp
Return	36	65293	la touche entrée du clavier de base
Right	102	65363	→ flèche gauche
Scroll_Lock	78	65300	ScrollLock
Shift_L	50	65505	shift gauche
Shift_R	62	65506	schift droit
Tab	23	65289	touche de tabulation
Up	98	65362	↑ flèche haut

pavé numérique

keysym	keycode	keysym_num	la touche
KP_1	87	65436	1 pavé numérique
KP_2	88	65433	2 pavé numérique
KP_3	89	65435	3 pavé numérique
KP_4	83	65430	4 pavé numérique
KP_5	84	65437	5 pavé numérique
KP_6	85	65432	6 pavé numérique
KP_7	79	65429	7 pavé numérique
KP_8	80	65431	8 pavé numérique
KP_9	81	65434	9 pavé numérique
KP_Add	86	65451	+ pavé numérique
KP_Begin	84	65437	touche 5 du pavé numérique
KP_Decimal	91	65439	point décimal pavé numérique
KP_Delete	91	65439	delete pavé numérique
KP_Divide	112	65455	/ pavé numérique
KP_Down	88	65433	↓ flèche bas pavé numérique
KP_End	87	65436	end pavé numérique
KP_Enter	108	65421	enter pavé numérique
KP_Home	79	65429	home pavé numérique
KP_Insert	90	65438	insert pavé numérique
KP_Left	83	65430	← pavé numérique
KP_Multiply	63	65450	* pavé numérique
KP_Next	89	65435	PageDown pavé numérique
KP_Prior	81	65434	PageUp pavé numérique
KP_Right	85	65432	→ pavé numérique
KP_Subtract	82	65453	- pavé numérique
KP_Up	80	65431	↑ pavé numérique

Il convient de contrôler les touches particulières **Meta, Print** ... qui peuvent varier selon les systèmes d'exploitation et le matériel. Voici un script qui peut y aider :

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *
from tkinter.messagebox import askyesno

# fenêtre principale
fenPr = Tk()
```

```

fenPr.title("événements clavier")

# quitter le logiciel
def quitter():
    reponse = askyesno("terminer le script",
        "Voulez-vous réellement terminer\u00a0? \n cliquer «oui» pout finir")
    if reponse :
        fenPr.quit()

maFonte ="Courier -25 bold"
# bouton quitter
btQuitter = Button(fenPr, font=maFonte, text="QUITTER", command=quitter)
btQuitter.pack(pady=20)
# capture de l'icone de la barre de titre
fenPr.protocol("WM_DELETE_WINDOW", quitter)

# le gestionnaire d'événements
def jeFrappe(event) :
    laSaisie.delete(0,END)
    chn = "le caractère : " + event.char
    chn += " \ncode de touche : " + str(event.keycode)
    chn += " \nkeysym (Id) : " + event.keysym
    chn += " \nkeysym (num) : " + str(event.keysym_num)
    chn += " \ntype : " + event.type
    leLabel.config(text=chn)

# les widgets
laFrame = Frame(fenPr, bg="yellow", name="nomCadre")
leLabel = Label(laFrame, name= "nomLabel", text=" \n\n\n\n\n",
    font=maFonte, bg="#e0e0ff", justify=LEFT, width=40)
laSaisie = Entry(laFrame, name= "nomBouton", font=maFonte, justify=CENTER)

laFrame.pack()
leLabel.pack(padx=20, pady=20)
laSaisie.pack(padx=20, pady=20)
laSaisie.focus_set() # forcer le focus pour les événements clavier
laSaisie.bind("<Key>", jeFrappe)
laSaisie.bind("<FocusOut>", lambda x : laSaisie.focus_set())

# boucle de la fenêtre principale
fenPr.mainloop()
fenPr.destroy() # par précaution

# fichier tk07ex01.py

```

tk08 : Tk et Toplevel

Un composant fenêtré de haut niveau (**Tk** ou **Toplevel**) a une existence indépendante ; elle n'a pas de conteneur. Elle possède une barre de titre, avec les boutons classiques (fermeture/plein écran ou non/ iconisation). Elle est redimensionnable et déplaçable sur l'écran ; elle peut avoir le focus. Il y a autant de fenêtres de ce type que l'on veut dans une application, dont l'une jouit d'un statut particulier, la fenêtre principale, de classe **Tk**.

1. les constructeurs

Tk, Toplevel

```
racine = Tk(options)
widget = Toplevel(master=None, options)
```

La majeure des propriétés (attributs, méthodes) des fenêtres de haut niveau est commun à l'instance de **Tk** et celles de **Toplevel**. Celles qui sont particulières à **Tk** seront signalées

2. les attributs

2.1. liste des attributs

background, **bd**, **bg**, **borderwidth**, **class**, **colormap**, **container**, **cursor**, **height**, **highlightbackground**, **highlightcolor**, **highlightthickness**, **menu**, **relief**, **screen**, **takefocus**, **use**, **visual**, **width**.

2.2. les attributs spécifiques

* **menu** : permet d'ajouter une barre de menu. La valeur de cet attribut est une widget **Menu**.

* **visual** : permet de créer un aspect d'affichage autre que celui de la fenêtre principale.

exemple : fenêtre Tk

```
'background': ('background', 'background', 'Background', <border object at 0x261f920>,
              '#a0a0ff'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x2620280>, 0),
'class': ('class', 'class', 'Class', 'Toplevel', 'Tk'),
'colormap': ('colormap', 'colormap', 'Colormap', '', ''),
'container': ('container', 'container', 'Container', 0, 0),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'height': ('height', 'height', 'Height', <pixel object at 0x2620460>, 0),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                       <color object at 0x2620730>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x261fb00>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                      <pixel object at 0x261f830>, 0),
'menu': ('menu', 'menu', 'Menu', '', ''),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x2620340>, <pixel object at 0x2620340>),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x261fc80>, <pixel object at 0x261fc80>),
'relief': ('relief', 'relief', 'Relief', <index object at 0x26201f0>, 'flat'),
'screen': ('screen', 'scribe', 'Screen', '', ''),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '0', '0'),
'use': ('use', 'use', 'Use', '', ''),
'visual': ('visual', 'visual', 'Visual', '', ''),
'width': ('width', 'width', 'Width', <pixel object at 0x261fbf0>, 0),
```

exemple : Toplevel

```
'background': ('background', 'background', 'Background', <border object at 0x1e289e0>,
'#d9d9d9'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x1e29340>, 0)
'class': ('class', 'class', 'Class', 'Toplevel', 'Toplevel'),
'colormap': ('colormap', 'colormap', 'Colormap', '', ''),
'container': ('container', 'container', 'Container', 0, 0),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'height': ('height', 'height', 'Height', <pixel object at 0x1e29520>, 0),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
<color object at 0x1e297f0>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
<color object at 0x1e28bc0>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
<pixel object at 0x1e288f0>, 0),
'menu': ('menu', 'menu', 'Menu', '', ''),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x1e29400>, <pixel object at 0x1e29400>),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x1e28d40>, <pixel object at 0x1e28d40>),
'relief': ('relief', 'relief', 'Relief', <index object at 0x1e292b0>, 'flat'),
'screen': ('screen', 'screen', 'Screen', '', ''),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '0', '0'),
'use': ('use', 'use', 'Use', '', ''),
'visual': ('visual', 'visual', 'Visual', '', ''),
'width': ('width', 'width', 'Width', <pixel object at 0x1e28cb0>, 0),
```

3. les méthodes spécifiques

3.1. relations avec le gestionnaire de fenêtres

- * `positionfrom(who=None)` : les valeurs de `who` peuvent être `"user"` ou `"program"`. Indique au gestionnaire de fenêtre que la position doit être manuelle ou calculée par le programme.
- * `sizefrom(who=None)` : même chose pour la position.
- * `protocol(nom, fonction)` : voir le chapitre **tk07 § 5**
- * `group(fenêtre)` : ajoute l'instance de fenêtre au groupe des fenêtres désigné par `fenêtre` ; les fenêtres d'un groupe sont iconifiées ou tracées en même temps que leur propriétaire. Implémentation incertaine.

3.2. Icone et visibilité

- * `iconbitmap(arg)` : sous Windows, `arg` est une référence (chaîne de caractère) à un fichier `.ico` valide. Sous Linux il faut ajouter `@` et donner un fichier `.xpm` :

```
fenPrincipale.iconbitmap("@pylogo.xpm") # Linux
fenPrincipale.iconbitmap("pylogo.ico") # Windows
```
- * `iconwindow(arg)` : devrait permettre de poser l'icone (`arg`) ou renseigner sur l'icone (pas d'`arg`) lorsque la fenêtre est iconifiée. Semble inégalement implémentée.
- * `iconify()` : passe de l'état étendu à l'état d'icone.
- * `deiconify()` : fait passer du mode icône au mode étendu.
- * `overrideredirect(flag=False)` : mise hors service (`True`) ou en service de la fenêtre (`False`). La fenêtre hors service est ignorée du gestionnaire de fenêtre. Le drapeau ne doit être posé qu'après apparition de la fenêtre, ou tout au moins de son calcul par `update_idletasks()`. Peu ne pas être implémenté sous

certaines Unix.

- * `state(newstate=None)` : Recherche (sans argument) l'état de la fenêtre ; les valeurs sont "normal", "icon", "iconic", "withdraw" (invisible) ou "zoomed" (Windows seulement). On peut poser la valeur, mais il vaut mieux utiliser les méthodes `iconify()`, `deiconify()`, `iconwindow()`, et `withdraw()`. Si l'état est `withdraw()`, faire `deiconify()` pour retracer l'image.

3.3. Style

- * `title(chaine)` : la chaîne passée en paramètre est le titre dans la barre de titre.

3.4. géométrie et position

- * `aspect(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)` : fixe les proportions de fenêtre ; le rapport largeur/hauteur souhaité doit être entre MINNUMER/MINDENOM et MAXNUMER/MAXDENOM. Retourne un tuple des valeurs actuelles si la méthode est appelée sans argument (None s'il n'y a pas).
- * `geometry()` : voir chapitre **tk04 géométries**
- * `lift(aboveThis=None)` : élève la fenêtre dans la pile des fenêtres au-dessus de `aboveThis`. Sans argument, met la fenêtre au sommet de la pile. (`aboveThis` est une instance de `Tk` ou `Toplevel`). Voir **t04 géométries**
- * `lower(belowThis=None)` : descend la fenêtre dans la pile des fenêtres en dessous de `belowThis`. Sans argument, met la fenêtre en bas de la pile. (`aboveThis` est une instance de `Toplevel` ou `Tk`). Voir **t04 géométries**
- * `maxsize(width=None, height=None)` : taille maximale de la fenêtre ; si les paramètres sont omis, retourne la taille maximale courante (par défaut la taille de l'écran) sous forme d'un tuple.
- * `minsize(width=None, height=None)` : taille minimale de la fenêtre ; si les paramètres sont omis, retourne la taille minimale courante (par défaut (1, 1)) sous forme d'un tuple.
- * `resizable(width=True, height=True)` : fixe le droit de changer la dimension de la fenêtre. Par défaut la fenêtre est redimensionnable dans les deux propriétés, largeur et hauteur. On peut ne bloquer qu'une seule de deux propriétés. Sans argument, donne sous forme d'un tuple (deux entiers), les deux valeurs booléennes `width` et `height`.
- * `transient(master)` : un widget `Toplevel` est `transient` par rapport à une fenêtre de haut niveau (`Tk` ou `Toplevel`) si il est affiché au dessus de son propriétaire (`master`) et si sa visibilité (iconifié, affiché) suit également son propriétaire : par exemple un menu popup. Appelé sans argument (ou avec `master=""`), la méthode indique que le widget d'appel ne peut être `transient`.

La propriété `transient` est une propriété passée au gestionnaire de fenêtre ; il importe donc que celui-ci ait déjà calculé la fenêtre propriétaire (`master`). La fenêtre `transient` est affichée, par défaut, de façon symétrique par rapport à son propriétaire sous Linux et sans particularité sous Windows. Il vaut donc mieux redéfinir le placement si on veut une application portable.

Les données `+x+y` de sa géométrie sont suspectes. Récupérer la position par `wininfo_x()`, `wininfo_y()`, `wininfo_rootx()`, `wininfo_rooty()`.

4. Un exemple de fenêtre Toplevel

L'objectif est de créer un module pour disposer d'une fenêtre de haut niveau avec les caractères suivants :

- c'est une fenêtre popup, qui peut être appelée depuis n'importe quelle autre fenêtre d'une application ;
- elle doit être modale, c'est à dire qu'elle capte les événements à son profit, et qu'elle reste au

- dessus de toute fenêtre qui a pu l'appeler ;
- on peut au besoin lui inclure une vignette (fichier au format compatible **tkinter**) et un texte multiligne ; son titre est aussi modifiable ;
- on peut choisir la fonte, sa taille, sa graisse.
- elle est centrée en fonction de sa fenêtre d'appel ; si celle-ci n'est pas explicite c'est la fenêtre principale.
- le module peut être testée (méthode du

l'exemple proposé :

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-

from tkinter import *

class PopAlerte (Toplevel) :
    """
    les paramètres sont dans l'ordre :
    maitre désigne le propriétaire, (None ==> fenêtre principale)
    titre, le titre de la popup
    message, le texte multiligne à afficher
    vignette, le nom d'un fichier image valide pour la vignette
    nfonte est un nom de fonte valide
    tfonte, sa taille en points ou, négatif en pixels
    gfonte est un booléen pour «gras».
    """

    def __init__(self, maitre, titre, message, vignette,
                 nfonte, tfonte, gfonte) :
        Toplevel.__init__(self, maitre)
        self.protocol("WM_DELETE_WINDOW", self.destroy)
        locFonte = [nfonte,tfonte]
        self.title(titre)
        if gfonte : locFonte.append("bold")
        # composants
        bmi = BitmapImage(file="./img/tk08ex01.xbm")
        btQuitter = Button(self, text="quitter",image=bmi,
                           compound='left', highlightcolor="red", font=locFonte,
                           command=self.destroy)
        btQuitter.pack(side="bottom", pady=10)
        btQuitter.focus_set()
        if vignette :
            bpi = PhotoImage(file=vignette)
            Label(self,bg="yellow", image=bpi).pack(side="left", padx=10)
            Label(self, text=message, font=locFonte).pack(side="left",
                                                           padx=10)
        # placement
        self.update_idletasks()
        self.transient(self.master) # fenêtre modale
```

```

w,h,mx,my,mw,mh = (self.winfo_width(), self.winfo_height(),
                    self.master.winfo_rootx(),self.master.winfo_rooty(),
                    self.master.winfo_width(), self.master.winfo_height())
x = "+" + str(mx + (mw - w)//2)
y = "+" + str(my + (mh - h)//2)
self.geometry (x+y)
self.grab_set() # capte les événements à son profit
# boucle locale qui maintient la fenêtre affichée
self.wait_window()

def alerter(maitre=None, titre ="Alerte", message="???",
           vignette="./img/tk08ex00.ppm", nfonte="Helvetica", tfonte=-25,
           gfonte=False) :
    PopAlerte(maitre, titre, message, vignette, nfonte, tfonte, gfonte)

if __name__ == "__main__":
    racine = Tk()
    racine.protocol("WM_DELETE_WINDOW", racine.quit)
    racine.geometry ("300x200+400+300")
    Button(racine, text="quitter", font="Helvetica -25",
           command=racine.quit).pack (pady=30)
    Button(racine, text="alerte", font="Helvetica -25",
           command=lambda : alerter(None, "essai de Alerter",
           "un message \nmultiligne\net une vignette trichrome",
           gfonte=1)).pack (pady=30)
    racine.mainloop()
    racine = racine.destroy()

# fichier tk08ex00.py

```

résultat :



tk09 : Menu

Le widget Menu sert fabriquer des menus en cascade (par exemple, barre de menu, sous menus, sous sous menus...) et des menus popup. Un widget **Menu** peut être une valeur de l'attribut **menu** pour un widget qui en possède un (**Tk**, **Toplevel**, **OptionMenu**). Un item de menu peut lui-même être un menu (en cascade) ou une fenêtre popup.

Le widget **Menu** par certains aspect est assimilable aux fenêtres de haut niveau. On peut d'ailleurs fabriquer des fenêtres de haut niveau avec ce widget. Sa caractéristique importante, en tant que fenêtre de haut niveau, est qu'il n'a pas de gestionnaire de géométrie au sens où on l'a vu au **chapitre tk04**. La géométrie des widgets **Menu** est une propriété non accessible de **tk**.

Les items qui constituent un menu sont indexés. On compte à partir de 0.

1. le constructeur

```
widget = Menu (master, options)
```

master est la référence à un conteneur. Mais ce paramètre n'a pas d'importance. On peut ajouter après le conteneur un nom en clair (**name=xxx**), comme pour tout widget.

2. les attributs

2.1. liste des attributs

activebackground, **activeborderwidth**, **activeforeground**, **background**, **bd**, **bg**, **borderwidth**, **cursor**, **disabledforeground**, **fg**, **font**, **foreground**, **postcommand**, **relief**, **selectcolor**, **takefocus**, **tearoff**, **tearoffcommand**, **title**, **type**

2.2. les attributs spécifiques

attributs de détachement :

- * **tearoff** : un menu peut être détaché et constituer une fenêtre de haut niveau. Par défaut, l'attribut **tearoff** est posé à **True**, ce qui signifie que le menu est détachable. Posé à **False**, le menu n'est pas détachable.
- * **tearoffcommand** : lors du détachement du menu, la fonction valeur de cet attribut est appelée.
- * **title** : Par défaut le titre d'une fenêtre détachée est hérité de l'item du menu de rattachement. Mais on peut changer ce titre en modifiant l'attribut **title**.

attribut d'utilisation :

- * **postcommand** : la valeur de cet attribut est une fonction appelée à chaque utilisation du menu.

cas de Checkbutton et Radiobutton :

- * **selectcolor** : un item de menu peut être un **Checkbutton** ou un **Radiobutton**. cet attribut est la couleur du widget quand il est sélectionné.
- * **type** : type du menu ; en principe "**normal**"

```
'activebackground': ('activebackground', 'activeBackground', 'Foreground',
                    <border object at 0x2036fd0>, <border object at 0x2036fd0>),
'activeborderwidth': ('activeborderwidth', 'activeBorderWidth', 'BorderWidth',
                     <pixel object at 0x2037270>, <pixel object at 0x2037270>),
'activeforeground': ('activeforeground', 'activeForeground', 'Background',
                    <color object at 0x2036ac0>, <color object at 0x2036ac0>),
'background': ('background', 'background', 'Background',
               <border object at 0x2037210>, <border object at 0x2037210>),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x2037090>,
               <pixel object at 0x2037090>)
```

```
'cursor': ('cursor', 'cursor', 'Cursor', <cursor object at 0x2037060>,
          <cursor object at 0x2037060>),
'disabledforeground': ('disabledforeground', 'disabledForeground', 'DisabledForeground',
                      <color object at 0x2037000>, <color object at 0x2037000>),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x2036ee0>, <font object at 0x2036ee0>),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x2036fa0>,
              <color object at 0x2036fa0>),
'postcommand': ('postcommand', 'postCommand', 'Command', '', ''),
'relief': ('relief', 'relief', 'Relief', <index object at 0x2036e20>, <index object at
0x2036e20>),
'selectcolor': ('selectcolor', 'selectColor', 'Background', <color object at 0x2036df0>,
               <color object at 0x2036df0>),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '0', '0'),
'tearoff': ('tearoff', 'tearOff', 'TearOff', 1, 1),
'tearoffcommand': ('tearoffcommand', 'tearOffCommand', 'TearOffCommand', '', ''),
'title': ('title', 'title', 'Title', '', ''),
'type': ('type', 'type', 'Type', <index object at 0x2036bb0>, <index object at 0x2036bb0>),
```

3. un exemple de barres de menu

Le widget `Menu` sert à créer des bases de menus. Plus précisément un fenêtre `Tk` ou `Toplevel` peut contenir une barres de menu est une seule, et son emplacement est de toute façon réservée en position haute de la fenêtre. Cette barre de menu est créée avec le constructeur `Menu`, et ses items sont créés avec la méthode `add_cascade()`. À chaque item est associé un menu déroulant, créé avec le même constructeur `Menu`. par la méthode susdite. Ce menu déroulant est enrichi grâce à la méthode `add_command()` qui spécifie l'item (une chaîne) et la commande qui lui est associée.

Si le menu déroulant est détachable, sa première ligne est faite d'une série de tirets qu'il suffit de cliquer pour effectuer le détachement.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
from tkinter import *
from tkinter.messagebox import showinfo
import sys
maFonte = "Helvetica -25"

racine = Tk() # Fenêtre principale
racine.geometry("500x400+300+300")

# création des menus
mainmenu = Menu(racine, font=maFonte) ## sera la barre de menu
menuExemple = Menu(mainmenu, font=maFonte) ## menu fils
menuInfo = Menu(mainmenu, font=maFonte, title='VERSION') ## menu fils

# enrichissement du menu Exemple
menuExemple.add_command(label="Affiche", command=
    lambda x="Exemple", y="Exemple d'un Menu Tkinter":showinfo(x,y))
menuExemple.add_command(label="Quitter", command=racine.quit)
```

```

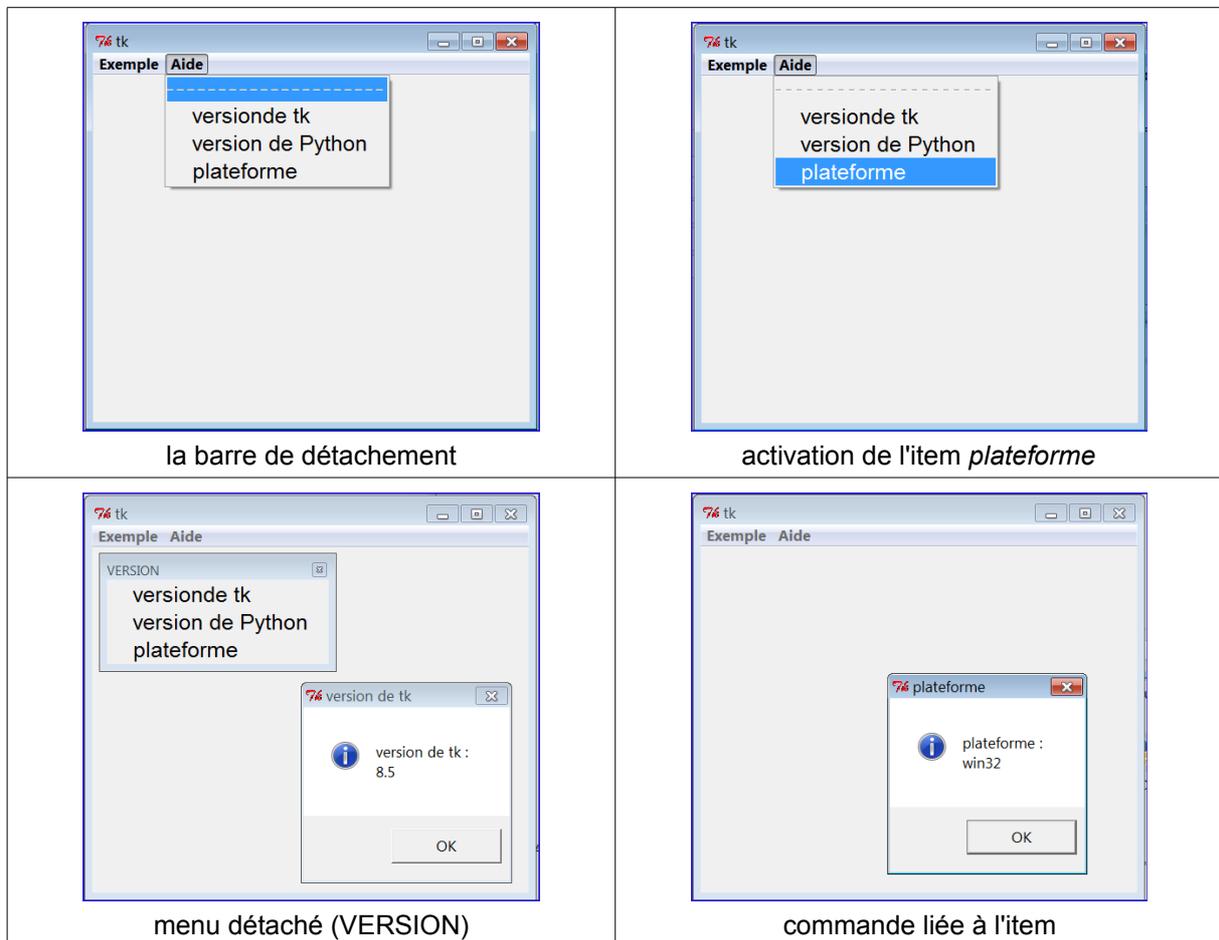
# enrichissement du menu Info
menuInfo.add_command(label="version de tk", command=
    lambda x="version de tk", y= str(TkVersion) : showinfo(x, x+" :\n"+y))
menuInfo.add_command(label="version de Python", command=
    lambda x="version de Python", y=sys.version : showinfo(x, x+" :\n"+y))
menuInfo.add_command(label="plateforme", command=
    lambda x="plateforme", y=sys.platform : showinfo(x, x+" :\n"+y))

# liaisons
mainmenu.add_cascade(label = "Exemple", menu=menuExemple)
mainmenu.add_cascade(label = "Aide", menu=menuInfo)
racine.config(menu = mainmenu) # affectation du menu principal

racine.mainloop()

```

résultats :



5. exemple de menu popup

Un menu popup est un menu transitoire, qui s'affiche en un point de coordonnées défini, en général par un clic de souris.

```

#!/usr/bin/python3
# -*- coding: utf-8 -*-
from tkinter import *
from tkinter.messagebox import showinfo

maFonte = "Helvetica -25"
racine = Tk() ## Fenêtre principale
racine.geometry("+300+300")

# le menu popup
monMenuPop = Menu(racine, tearoff=0, font=maFonte, borderwidth=2,
                  relief=SOLID, activeforeground="red")
# enrichissement du menu popup
monMenuPop.add_command(label="*** exit ***")
monMenuPop.add_command(label="version de tk", command=
    lambda x="version de tk", y= str(TkVersion) : showinfo(x, x+" :\n"+y))
monMenuPop.add_command(label="version de Python", command=
    lambda x="version de Python", y=sys.version : showinfo(x, x+" :\n"+y))
monMenuPop.add_command(label="plateforme", command=
    lambda x="plateforme", y=sys.platform : showinfo(x, x+" :\n"+y))

# création du menu popup
def popup(event) :
    monMenuPop.post(event.x_root, event.y_root)

# un cadre pour cliquer
leCadreCliquer = Frame(racine, width=500, height=400, bg="#a0ffff")
leCadreCliquer.pack()
leCadreCliquer.bind("<Button>", popup)

# bouton de fin de script
monBouton = Button (racine, text="quitter", font=maFonte,
                   bg="#c0c0c0", command=racine.quit)
monBouton.pack (side=BOTTOM, pady=10)
racine.mainloop()

# fichier tk09ex01.py

```



6. les méthodes spécifiques

6.1. les options d'items

Certaines options (clef/valeur) s'appliquent aux items du menus. En voici la description. Les options qui ne sont pas renseignées renvoient aux options de configuration définies dans **Checkbutton**, **Radiobutton** ou **Button** (un item «ordinaire» est assimilable à un widget **Button**).

option d'items	type	explication
----------------	------	-------------

<code>accelerator</code>	chaîne	cette option permet d'afficher une touche de raccourci après le label. Par exemple pour afficher Ctrl-X, faire <code>accelerator="Ctrl-X"</code> . Attention, cette option ne fait qu'afficher !
<code>activebackground</code>	couleur	lorsque la souris survole l'item
<code>activeforeground</code>	couleur	lorsque la souris survole l'item
<code>background</code>	couleur	couleur normale
<code>bitmap</code>	bitmap	permet d'afficher un bitmap en étiquette
<code>columnbreak</code>	booléen	permet de commencer une nouvelle colonne dans le menu. Autorise ainsi des menus horizontaux.
<code>command</code>	fonction	fonction appelée lors de l'activation de l'item.
<code>compound</code>	ancree	position de l'image en cas où on veut image et texte : les valeurs sont <code>LEFT="left"</code> , <code>RIGHT="right"</code> , <code>TOP="top"</code> , <code>BOTTOM="bottom"</code> , <code>CENTER="center"</code> , <code>NONE="none"</code> .
<code>font</code>	fonte	fonte du texte de l'item
<code>foreground</code>	couleur	ne pas abrégé (<code>fg</code>). Couleur normale
<code>hidemargin</code>	booléen	permet de supprimer l'interlignes éventuels entre les items.
<code>image</code>	image	permet d'afficher une image en étiquette
<code>indicatoron</code>	booléen	Voir <code>Checkbutton</code> ou <code>Radiobutton</code>
<code>label</code>	chaîne	intitulé de l'item, qui sera affiché dans le menu.
<code>menu</code>	Menu	sous-menu pour <code>add_cascade()</code>
<code>offvalue</code>	valeur	voir <code>Radiobutton</code>
<code>onvalue</code>	valeur	voir <code>Radiobutton</code>
<code>selectcolor</code>	couleur	couleur de sélection pour un <code>Checkbutton</code> ou <code>Radiobutton</code>
<code>selectimage</code>	image	voir <code>Radiobutton</code> et <code>Checkbutton</code>
<code>state</code>	constante chaîne	<code>DISABLED="disabled"</code> ou <code>'ACTIVE='active'</code>
<code>underline</code>	entier	index de la lettre à souligner
<code>value</code>	valeur	Voir <code>Radiobutton</code> et <code>Checkbutton</code>
<code>variable</code>	StringVar ou IntVar	Voir <code>Radiobutton</code> et <code>Checkbutton</code>

6.2. les méthodes d'ajouts d'items à un menu.

* `add (type, options)` : ajoute un item du type donné, avec les options précisées (voir les méthodes qui suivent). Le type peut être : `"command"`, `"cascade"` (sous-menu), `"checkboxbutton"`, `"radiobutton"`, `"separator"`. Cette fonction est **utilisée en interne** pour définir les méthodes d'ajouts qui lui sont équivalentes `add_cascade()`, `add_checkboxbutton`, `add_command` ,

`add_radiobutton`.

* `add_cascade (options)` : ajoute un item `Menu` au menu d'appel. Cet item `Menu` est attaché en sous-menu. Les options permettent de préciser le label l'affichage, le menu attaché en cascade etc. Il convient au minimum de spécifier `label` et `menu`.

* `add_checkbutton (options)` : ajoute un item au menu d'appel. Cet item est un widget `Checkbutton`. Les options sont parallèles à celles qui permettent de paramétrer un `Checkbutton` (Voir le **chapitre tk13**) avec cependant un éventail d'option restreint.

* `add_command (options)` : ajoute un item avec une commande. Il convient au minimum de spécifier `label` et `command`.

* `add_radiobutton (options)` : ajoute un item qui est un widget `Radiobutton`. Les options sont parallèles à celles qui permettent de paramétrer un `Radiobutton` (Voir le **chapitre tk14**) avec cependant un éventail d'option restreint.

* `add_separator (options)` : ajoute un séparateur. Dans le décompte de l'index, le séparateur est compté comme un item.

* `insert (index, type, option)` : Cette méthode ainsi que les suivantes est identique à `add()`. Mais là où `add()` ajoute en queue de menu (`append`, au sens des séquences de **Python**), `insert()` remplace l'item situé à la position `index`, décalant ainsi les suivants.

* `insert_cascade (index, options)` : id.

* `insert_command (index, options)` : id.

* `insert_checkbutton (index, options)` : id.

* `insert_radiobutton (index, options)` : id.

* `insert_separator (index)` : id.

* `delete(index1, index2=None)` : supprime l'item `index1` si `index2` n'est pas précisé. Sinon, supprime tous les items entre `index1` et `index2` **bornes comprises**.

6.3. méthodes de configuration

* `entrycget (index, clef_option)` : retourne la valeur de l'option d'item de clé donnée (chaîne), à l'index donné.

* `entryconfig (index, options)` : modifie les options précisées de l'item à la position `index`, et ne touche pas aux options non évoquées.

* `invoke (index)` : simule l'activation de l'item désigné par `index`. Si l'item comporte une commande, exécute la commande.

6.4. méthodes de popup

* `post (x, y)` : réalise l'affichage du menu au point de coordonnées (x, y). L'activation d'un de ses items efface le menu. (Voir l'exemple de la section 5).

* `unpost()` : efface le menu affiché par `post()`.

6.5. Divers

* `activate (index)` : active l'item désigné par `index`.

* `index(v)` : retourne l'index de `v`. Si `v` est un entier valide, retourne cet entier. Si `v` est supérieur au dernier index, retourne cet index ; même chose si `v=END` (ou `v='end'`). Si `v` est un label valide, retourne l'index de son item. Si `v` vaut `NONE="none"`, retourne `"None"` ! Erreur sinon.

* `type (index)` : retourne le `type` de l'item en position `index`.

* `yposition(index)` : cette méthode retourne un entier qui peut servir à positionner un menu transitoire (popup). Il s'agit de la position haute de l'item sur l'écran au cours de son affichage.

tk10 : Frame et LabelFrame

Une instance de **Frame** est un «cadre» sans texte, sans déclencheur d'événement par défaut, qui a la propriété de pouvoir être un conteneur pour d'autres widgets, de type **Frame** compris. Par défaut il a une bordure.

Le **LabelFrame** est un conteneur semblable à **Frame**, mais avec une étiquette sur sa bordure.

1. le constructeur

syntaxe

```
widget = Frame(conteneur, options)
```

```
widget = LabelFrame(conteneur, options)
```

2. options

2.1. liste des options de Frame

background, bd, bg, borderwidth, class, colormap, container, cursor, height, highlightbackground, highlightcolor, highlightthickness, padx, pady, relief, takefocus, visual, width

Toutes les attributs sont partagées (chapitre **tk05**)

* *padx, pady* : il s'agit ici de marges intérieures.

```
'background': ('background', 'background', 'Background', <border object at 0x1961920>,
               'yellow'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x1afa530>, 0),
'class': ('class', 'class', 'Class', 'Frame', 'Frame'),
'colormap': ('colormap', 'colormap', 'Colormap', '', ''),
'container': ('container', 'container', 'Container', 0, 0),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'height': ('height', 'height', 'Height', <pixel object at 0x1962460>, 0),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                        <color object at 0x1962730>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x1961b00>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                      <pixel object at 0x1961830>, 0),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x1962340>, <pixel object at 0x1962340>),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x1961c80>, <pixel object at 0x1961c80>),
'relief': ('relief', 'relief', 'Relief', <index object at 0x1b69a70>, 'flat'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '0', '0'),
'visual': ('visual', 'visual', 'Visual', '', ''),
'width': ('width', 'width', 'Width', <pixel object at 0x1961bf0>, 0),
```

2.2. liste des options de LabelFrame

background, borderwidth, class, colormap, container, cursor, font, foreground, height, highlightbackground, highlightcolor, highlightthickness, labelanchor, labelwidget, padx, pady, relief, takefocus, text, visual, width

Sauf pour *font, labelanchor, labelwidget*, c'est la même chose que pour le widget **Frame**.

```
'background': ('background', 'background', 'Background', <border object at 0x1ca9920>,
               '#d9d9d9'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x1facec0>, 2),
'class': ('class', 'class', 'Class', 'Labelframe', 'Labelframe'),
'colormap': ('colormap', 'colormap', 'Colormap', '', ''),
'container': ('container', 'container', 'Container', 0, 0),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x1fac980>, 'TkDefaultFont'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x1facc80>,
               '#000000'),
'height': ('height', 'height', 'Height', <pixel object at 0x1caa460>, 0),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                        <color object at 0x1caa730>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x1ca9b00>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                      <pixel object at 0x1ca9830>, 0),
'labelanchor': ('labelanchor', 'labelAnchor', 'LabelAnchor', <index object at 0x1face0>,
               'wn'),
'labelwidget': ('labelwidget', 'labelWidget', 'LabelWidget', '', '.30681040'),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x1caa340>, <pixel object at 0x1caa340>),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x1ca9c80>, <pixel object at 0x1ca9c80>),
'relief': ('relief', 'relief', 'Relief', <index object at 0x1facf20>, 'groove')
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '0', '0'),
'text': ('text', 'text', 'Text', '', ''),
'visual': ('visual', 'visual', 'Visual', '', ''),
'width': ('width', 'width', 'Width', <pixel object at 0x1ca9bf0>, 0),
```

2.3. attributs spécifiques

- * **labelanchor** : prend comme valeur une des 9 valeurs d'ancre en référence aux point cardinaux : N="n", S="s", E="e", W="w", NE="ne", NW="nw", SE="se", SW="sw", "en", "es", "wn", ws". Par défaut c'est NW="nw", en haut à gauche.
- * **labelwidget** : au lieu d'un texte, on peut utiliser n'importe quel widget comme étiquette. L'option **text** est alors ignorée.

exemple :

```
imgPPM = PhotoImage(file="./img/pylogo.ppm")
etiquette = Label(image = imgPPM) # à ne pas mapper
laFrame = LabelFrame(fenPr, labelanchor="wn", labelwidget = etiquette)
```

3. méthodes

Il n'y a pas de méthode spécifique au widget **Frame** et au widget **LabelFrame**.

tk11 : PanedWindow

Un widget **PanedWindow** offre un jeu de cadres juxtaposés qui peuvent être agrandis ou rétrécis par l'utilisateur, au dépends des voisins, en manipulant les échancrures de séparation (**sash**, avec un **handle**), horizontales ou verticales à l'aide de la souris.

1. le constructeur

syntaxe

```
widget = PanedWindow (conteneur, options)
```

exemple :

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
from tkinter import *
maFonte = "Helvetica -25"

# fenêtre principale
racine = Tk()    ## Fenêtre principale
racine.protocol("WM_DELETE_WINDOW", racine.quit)
racine.geometry("+300+300")

# les PanedWindows
panedRacine = PanedWindow(racine, bd=4, relief=SOLID, handlesize=10,
    sashrelief=SOLID, showhandle=True, bg="red", width=800, height=400)
panedRacine.pack(padx=10, pady=10, fill="both", expand=True)

panedDroite = PanedWindow(panedRacine, bg="blue", orient=VERTICAL,
    showhandle=True)
panedDroiteBas = PanedWindow(panedDroite, bg="#80ff80" )

# les widgets enfants non PanedWindows
lbGauche = Label(panedRacine, text="partie gauche", font=maFonte,
    bg="#b0ffff")
lbDroiteHaut = Label(panedDroite, text="partie droite haut", font=maFonte,
    bg="#ffb0ff")
lbDroiteBasGauche = Label(panedDroiteBas,
    text="partie \ndroite \nbas \ngauche", font=maFonte, bg="#ffffb0")
btQuitter = Button(panedDroiteBas, text="quitter", font=maFonte,
    command=racine.quit)
logoPy = PhotoImage (file="./img/logopy.pnm")
lbLogoPy = Label(panedDroiteBas, image=logoPy)

# remplissage de PanedRacine
panedRacine.add(lbGauche)
panedRacine.add(panedDroite)
```

```

# remplissage de panedDroite
hauteurDroiteHaut = panedRacine.winfo_pixels(panedRacine["height"]) - \
    lbLogoPy.winfo_reqheight()-\
    panedRacine.winfo_pixels(panedRacine["sashwidth"])-34
# attention : faire explicitement le casting avec winfo_pixels
# 34 = padx panedRacine, bd pabedRacine, 2 * pady panedDroiteBas
panedDroite.add(lbDroiteHaut, height=hauteurDroiteHaut)
panedDroite.add(panedDroiteBas)

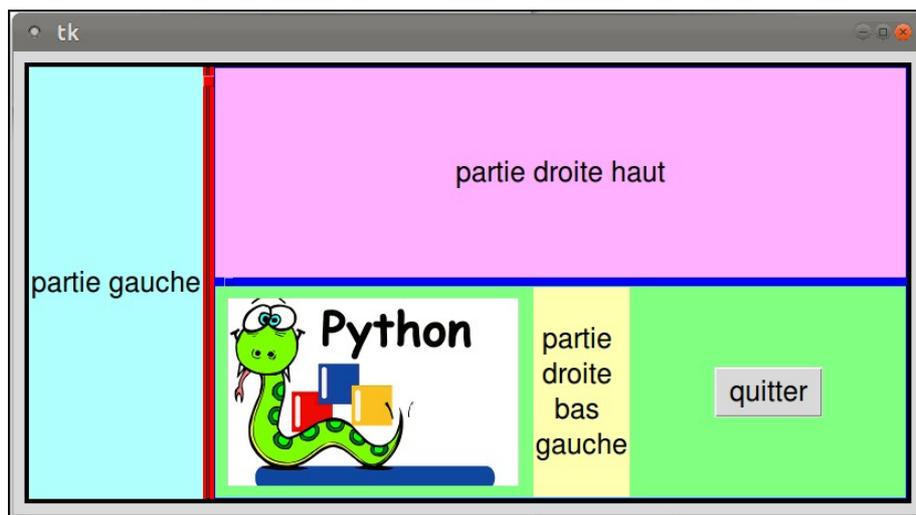
# remplissage de panedDroiteBas
panedDroiteBas.add(lbDroiteBasGauche)
panedDroiteBas.add(btQuitter, sticky='')
panedDroiteBas.add(lbLogoPy, before=lbDroiteBasGauche, padx=10, pady=10)

racine.mainloop()

# fichier tk1lex00.py

```

résultat :



2. options

2.1. liste des options de PanedWindows

background, bd, bg, borderwidth, cursor, handlepad, handlesize, height, opaquesize, orient, relief, sashcursor, sashpad, sashrelief, sashwidth, showhandle, width

2.2. options spécifiques :

- * **orient** : prendre l'une des deux valeurs `HORIZONTAL="horizontal"` (défaut) et `VERTICAL="vertical"`. Dans la première orientation, les enfants sont ajoutés de gauche à droite, et dans la seconde, du haut vers le bas.
- * **sashcursor** : les panneaux qui constituent un widget `PanedWindow` sont séparés par des échancrures (`sash`). Le `sashcursor` est le curseur de souris lors du survol des échancrures. Le curseur par défaut est en général la flèche double

("sb_v_double_arrow" ou "sb_h_double_arrow") dont la forme précise est défini par la plate-forme.

- * **sashrelief** : par défaut, l'échancrure est transparente et sans relief (**sashrelief** ="flat"). On peut changer son relief . La couleur de base est celle de l'arrière-plan.
- * **sashwidth** : largeur demandée pour l'échancrure.
- * **sashpad** : marge rajoutée de chaque côté de l'échancrure
- * **showhandle** : il y a un petit carré sur l'échancrure appelé sa poignée (**handle**). Elle sert uniquement à signaler la présence de l'échancrure. Selon les systèmes, la relation entre poignée et échancrure peut être différentes. Sous Linux par exemple, la poignée est intérieure à l'échancrure est sa taille s'impose relativement à **sashsize**.
- * **handlesize** : dimension de la poignée.
- * **handlepad** : marge ajoutée à la poignée.
- * **opaqueresize** : par défaut, **True** ; lors du déplacement de la souris, les panneaux suivent le mouvement. Mis à **False**, les panneaux ne se redimensionnent que lorsque le bouton de la souris est relâché.

```
'background': ('background', 'background', 'Background', <border object at 0x250ca10>,
              'cyan'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x2574810>, 4),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'handlepad': ('handlepad', 'handlePad', 'HandlePad', <pixel object at 0x2507310>, 8),
'handlesize': ('handlesize', 'handleSize', 'HandleSize', <pixel object at 0x2507df0>,
               <pixel object at 0x2507df0>),
'height': ('height', 'height', 'Height', '', <pixel object at 0x25073a0>),
'opaqueresize': ('opaqueresize', 'opaqueResize', 'OpaqueResize', 1, 1)
'orient': ('orient', 'orient', 'Orient', <index object at 0x2579b50>, 'horizontal'),
'relief': ('relief', 'relief', 'Relief', <index object at 0x250c9e0>, 'solid'),
'sashcursor': ('sashcursor', 'sashCursor', 'Cursor', '', ''),
'sashpad': ('sashpad', 'sashPad', 'SashPad', <pixel object at 0x250c6b0>, 0),
'sashrelief': ('sashrelief', 'sashRelief', 'Relief', <index object at 0x250c710>, 'flat'),
'sashwidth': ('sashwidth', 'sashWidth', 'Width', <pixel object at 0x250c740>,
              <pixel object at 0x250c740>),
'showhandle': ('showhandle', 'showHandle', 'ShowHandle', 0, 0),
'width': ('width', 'width', 'Width', '', <pixel object at 0x25074c0>),
```

3. configuration de panneaux

Les panneaux créés dans un widget **PanedWindow** sont occupés par un **widget enfant qui n'a donc pas de gestionnaire de géométrie propre**. C'est son adjonction au widget **PanedWindow** qui à la fois crée le panneau et régit une géométrie qui est régulée par les attributs suivante :

option de placement	valeur	explication
after, before	widget	lors de la création de panneaux par la méthode add() , les panneaux sont placés dans l'ordre de création. On peut modifier cet ordre en précisant après (after), ou avant (before) quel panneau existant on veut le placer, en citant le widget qui y est placé. On peut également changer l'ordre avec la méthode de configuration panedconfigure() .

hide	booléen	permet de cacher un panneau ; seul l'affichage est affecté, et aucun espace n'est requis. Le panneau et son échancrure subsistent,.
minsize	distance	requête d'une taille minimale pour le panneau, c'est à dire de la hauteur pour un panneau à orientation verticale, et la largeur pour un panneau à orientation horizontale.
padx	distance	marge transparente à ajouter à gauche et à droite au widget du panneau
pady	distance	marge transparente à ajouter à haut et en bas au widget du panneau.
sticky	cst chaîne	donne la façon de placer le widget dans le panneau lorsque celui-ci est trop grand. La valeur peut être N="n" , S="s" , E="e" , S="s" et leur combinaisons (exemple : N+E="ne" , ou N+E+S+W="nesw" ; il faut respecter l'ordre n,e,s,w). Le widget colle au(x) bord(s) du panneau selon l'indication du sticky . La valeur " nesw " est la valeur par défaut, et le widget colle à l'espace disponible, ne respectant plus aucune des dimensions requises. Avec une chaîne vide, le widget se place au centre, aux dimensions requises pour lui. On a rencontré ce comportement avec la méthode de placement grid()
stretch	cst chaîne	indique le comportement du remplissage par le widget lors d'un redimensionnement (changement de taille, insertion d'un nouveau panneau, redimensionnement du panneau par les échancrures. Les valeurs peuvent être " always " (recalcul complet du placement), " never " (aucun recalcul), " first " (conserve la gauche), " last " (la droite), " middle " (le milieu). Notion non documentée , au comportement déroutant. Laisser à la valeur par défaut.
height	distance	hauteur du widget inclus ; s'il y a conflit avec sticky , celui-ci est prioritaire.
width	distance	largeur du widget ; s'il y a conflit avec sticky , celui-ci est prioritaire.

```
'after': ('after', '', '', '', '');
'before': ('before', '', '', '', '');
'height': ('height', '', '', '', '');
'hide': ('hide', 'hide', 'Hide', 0, 0);
'minsize': ('minsize', '', '', <pixel object at 0x1e50bc0>, 0);
'padx': ('padx', '', '', <pixel object at 0x1e51dc0>, 0);
'pady': ('pady', '', '', <pixel object at 0x1eb9aa0>, 0);
'sticky': ('sticky', '', '', 'nsew', 'nesw');
'stretch': ('stretch', 'stretch', 'Stretch', <index object at 0x1eb9cb0>, 'last');
'width': ('width', '', '', '', '');
```

4. méthodes du widget PanedWindow

* **panedconfigure (Id, options)** : adaptation du **configure()** des widgets aux panneaux d'un widget **PanedWindow**. le paramètre **Id** est le numéro du panneau

(commence à 0)

- * `panecget(enfant, option)` : adaptation du `cget()` des widgets. *enfant* est un widget.
- * `panes()` : retourne la liste des widgets inclus ; commence en haut ou à gauche.
- * `add(enfant, options_de_placement)` : crée un panneau contenant le widget *enfant*, et le place avec les options de placement.
- * `remove(enfant)` : supprime la référence de *enfant* dans le widget `PanedWindow`. Ne supprime pas le widget enfant.
- * `forget(enfant)` : même chose que `remove()`.
- * `sash_coord(index)` : retourne l'emplacement sous forme d'un tuple (*x*, *y*) de l'échancrure de numéro *index*. (numérotation de gauche à droite ou de haut en bas, en commençant par 0). Les coordonnées sont données par rapport au panneau de même *index*. Pour un widget enfant caché, son échancrure associée a comme coordonnées (0, 0).
- * `sash_place(index, x, y)` : déplace une échancrure selon les coordonnées *x* et *y*.
- * `identify(x, y)` : retourne une référence sur ce qui est au point de coordonnées *x* et *y* dans le widget `PanedWindow`. Les divers cas qui se présentent sont :
 - (x, y) situe une échancrure : retourne un tuple de la forme (*index*, "sash") où *index* est le numéro d'index.
 - (x, y) situe une poignée : retourne un tuple de la forme (*index*, "handle") où *index* est le numéro d'index de la poignée.
 - (x, y) situe l'intérieur d'un panneau : retourne une chaîne vide.

tk12 : Scrollbar

Le widget `Scrollbar` permet d'insérer une barre de scroll (ascenseur) et de lier son effet à un widget `Entry`, `Listbox`, `Text`, `Canvas`, ou d'autres widget ayant une «vue». Le widget est focusable. Lorsqu'il a le focus, il est sensible aux touches fléchées du clavier.

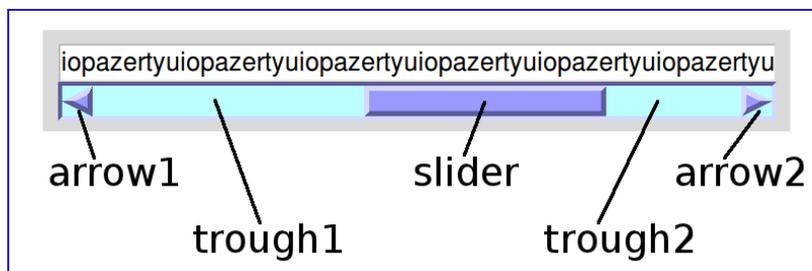
1. le constructeur

syntaxe

```
widget = Scrollbar(master, options)
```

Le **placement** du widget associé à un autre widget peut présenter quelque difficulté. Il faut respecter la disposition traditionnelle, ou la barre de scroll est à droite, ou sous la fenêtre d'affichage et d'une dimension peu différente. La barre de scroll n'a pas les options classiques de dimensionnement et il faut procéder avec les particularité des **gestionnaires de placement**. Une solution simple est de définir un cadre (`Frame`) qui les contiendra tous deux (ou trois) et eux seulement :

- dans le cas où on utilise le gestionnaire `pack()`, penser à utiliser les attributs de `pack`, `padx`, `pady` et `fill`.
- dans le cas où on utilise `grid()`, les attributs `padx`, `pady` et `sticky` avec les valeurs `E+W` `="ew"` et `N+S` `="ns"`.



2. options

2.1. liste des options

`activebackground`, `activerelief`, `background`, `bd`, `bg`, `borderwidth`, `command`, `cursor`, `elementborderwidth`, `highlightbackground`, `highlightcolor`, `highlightthickness`, `jump`, `orient`, `relief`, `repeatdelay`, `repeatinterval`, `takefocus`, `troughcolor`, `width`

2.2. attributs spécifiques

- * `activerelief` : par défaut, le style de relief du curseur est `RAISED="raised"` ; cet attribut permet de modifier le relief lorsque la souris survole le widget.
- * `command` : la fonction appelée lors du déplacement du curseur. Voir la section 3.
- * `elementborderwidth` : largeur de la bordure autour des flèches et du curseur du widget. La valeur `-1` signifie que la largeur est `borderwidth`.
- * `jump` : à `False`, la commande est appelée dès que le curseur se déplace avec le bouton de souris enfoncé. Posé à `True`, la commande est appelée au relâchement de la souris.
- * `orient` : Peut prendre les valeurs `HORIZONTAL="horizontal"` ou `VERTICAL="vertical"`.
- * `repeatdelay` : délai en millisecondes pour que la souris soit prise en compte lorsque le bouton 1 de la souris est enfoncé avec son curseur sur la gouttière du widget.
- * `repeatinterval` : intervalle entre deux prises en compte de la souris au bouton 1 enfoncé dans les conditions vues précédemment.
- * `troughcolor` : couleur de la gouttière (*trough*).
- * `width` : «largeur» du widget, c'est-à-dire largeur dans le sens horizontal si le widget est vertical, et hauteur dans le cas d'un widget horizontal.

```
'activebackground': ('activebackground', 'activeBackground', 'Foreground', '#ecec',
                    '#ecec'),
'activerelief': ('activerelief', 'activeRelief', 'Relief', 'raised', 'raised'),
'background': ('background', 'background', 'Background', '#d9d9d9', '#d9d9d9'),
'bd': ('bd', 'borderWidth'),
'bg': ('bg', 'background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', '1', '1'),
'command': ('command', 'command', 'Command', '', '29885000xview'),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'elementborderwidth': ('elementborderwidth', 'elementBorderWidth', 'BorderWidth', '-1',
                      '-1'),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                       '#d9d9d9', '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor', '#000000',
                  '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness', '0',
                      '0'),
'jump': ('jump', 'jump', 'Jump', '0', '0'),
'orient': ('orient', 'orient', 'Orient', 'vertical', 'horizontal'),
'relief': ('relief', 'relief', 'Relief', 'sunken', 'sunken'),
'repeatdelay': ('repeatdelay', 'repeatDelay', 'RepeatDelay', '300', '300'),
'repeatinterval': ('repeatinterval', 'repeatInterval', 'RepeatInterval', '100', '100'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'troughcolor': ('troughcolor', 'troughColor', 'Background', '#b3b3b3', '#b3b3b3'),
'width': ('width', 'width', 'Width', '11', '11'),
```

3. la fonction `command`

La fonction associée à la commande est spécifique au widget ; elle possède des variantes d'appel. Une forme d'appel a deux paramètres (action sur le **slider**), l'autre 3 (action sur les flèches). On utilise usuellement les méthodes `xview` et `yview` des widgets qui connaissent la connexion à une (ou deux) barre(s) de scroll, **Entry**, **Listbox**, **Text**, **Canvas**. Un exemple avec deux barres est donné au chapitre 21 § 2.

3.1. cas où le curseur de la barre de scroll est déplacé

L'appel doit être fait sur une fonction dont la signature est :

```
maFonction ("moveto", fraction)
```

la valeur de `fraction` est un flottant entre 0 et 1. La valeur 0 correspond à une position originelle du curseur (haut ou gauche) et 1 correspond à une position extrême (bas ou droite). `fraction` est le rapport `through1/gouttière`.

3.2. cas où un mouvement unitaire est requis

Il s'agit alors du bouton de souris enfoncé sur les parties réactives (flèches, cuvette) du widget.

Il y a alors 4 schémas possibles :

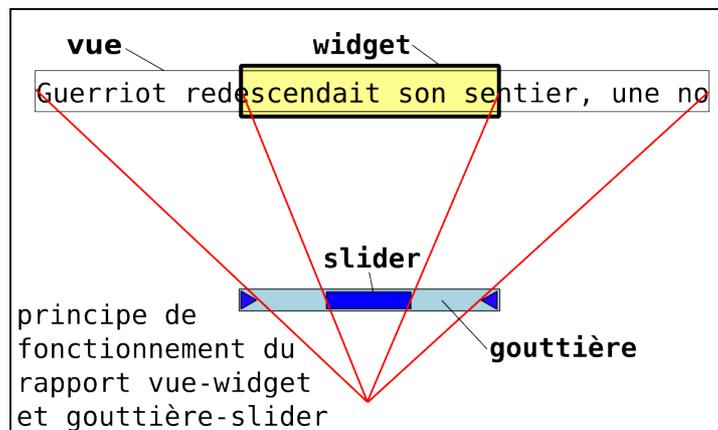
```
maFonction ("scroll", entier, mode)
```

L'entier (valeur -1 ou 1 en principe) signifie vers le haut/bas ou la gauche/droit (selon l'orientation). Le mode ("`units`", "`page`") signifie l'unité retenue. Le premier est l'unité de base du widget (ligne ou colonnes par exemple) et le second, une page, si cela a un sens pour le widget.

4. méthodes

* `activate (index=None)` : affiche avec les attributs «d'actif» (survol de la souris) la partie du widget caractérisée par `index` peut être "`arrow1`", "`slider`" (curseur) or "`arrow2`". Si l'argument est omis, retourne l'index de la partie éventuellement active.

- * **delta** (*deltax*, *deltay*) : transforme les composantes d'un mouvement de souris en valeur entre -1.0 et 1.0 qui doit être ajouté à la position du curseur du widget pour obtenir le même déplacement.
- * **fraction** (*x*, *y*) : transforme les coordonnées d'un point, *x* et *y* en valeur de position du curseur (valeur de l'intervalle 0.0, 1.0) pour le rendre le plus voisin possible de ce point.
- * **identify** (*x*, *y*) : retourne l'une de valeurs "arrow1", "slider", "arrow2", "trough1", "trough2" ou "" donnant la partie du widget qui a comme coordonnées *x* et *y*.
- * **get** () : retourne un tuple (*d*, *f*) des valeurs entre 0.0 et 1.0 qui caractérisent la position actuelle du curseur du widget : par exemple, si on retourne (0.5, 0.75), le curseur s'étend de 0.5 à 0.75 fois la longueur de la gouttière, et le curseur proprement a donc une largeur de 0.25.
- * **set** (*d*, *f*) : impose *d* et *f* à la barre de scroll, avec *d* et *f* qui ont la même signification que précédemment. Les widgets qui ont la propriété **xscrollcommand** ou **yscrollcommand** sont prévus pour que ces attributs prennent la valeur **set**. Cette méthode est surtout utilisée pour connecter un widget à une barre de scroll.



5. un script pour montrer le fonctionnement des connexions

5.1. la double connexion

Dans les exemples rencontrés, il y a une double déclaration de connexion :

- du widget vers la Scrollbar : par exemple, **xscrollcommand=set**. Le widget est renseigné sur la méthode **xscrollcommand** de mise à jour du curseur de la barre de scroll.
- de la Scrollbar vers le widget : par exemple, **command=widget.xview**. Lors d'une action sur la barre de scroll, la procédure **command** est appelée, qui met à jour à la fois l'affichage du widget et la barre de scroll.

La compréhension de ce qu'est un widget scrollable est assez délicate. Le plus simple est de créer un exemple d'école pour en manifester les détails. Le sujet retenu est le suivant : on définit un composant graphique assez simple :

- il affiche un entier entre 1 et une valeur paramétrable par un attribut, **sup**.
- il est doté de deux boutons, + et - qui incrémente ou décrémente cet entier.
- il est scrollable, c'est à dire prévu pour être interconnecté avec une barre de scroll.
- il se comporte comme un widget : ses méthodes **config()**, **cget()**, **keys()** ont été redéfinies, mais il ne partage pas les attributs communs aux widgets. De même, il est **subscriptable**, c'est-à-dire que sous certaines conditions (ici avec les noms d'attributs) les expressions de la forme suivante : **widget["monAttribut"]** sont admises.
- il se place avec les trois gestionnaires de géométrie.

5.2. le script commenté

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
#
import math
from tkinter import *
maFonte = "Helvetica -40"

# fonction pour quitter
def quitter():
    racine.quit()

# la fenêtre de l'application
racine = Tk()
racine.protocol("WM_DELETE_WINDOW", quitter)
racine.geometry("+200+250")

btQuitter = Button (racine, text= "QUITTER", foreground= "red",
                    bg="grey", font = "Courier -25 bold", command= quitter)
btQuitter.pack(side="bottom", pady=10)

leCadre = Frame(racine)
leCadre.pack()

# la classe de l'utilitaire
class NumScroll (Frame) :
    def __init__(self, master=None, font="", sup =10,
                 scrollcommand=None) :
        Frame.__init__(self, master) # traiter NumScroll comme un widget
        self.__subscript = { # dictionnaire des attributs spécifiques
            "font" : font ,
            "sup" : sup ,
            "scrollcommand" : scrollcommand
        }
        self.varLabel = StringVar()
        self.ratio = sup//2
        self.varLabel.set(str(self.ratio))
        btPlus = Button (self, text="+", font=font, command = self.plus)
        btMoins = Button (self, text="-", font=font, command = self.moins)
        labelNum = Label (self, text = "", font=font,
                          textvariable=self.varLabel, width=5)
        btMoins.pack(side="left")
        labelNum.pack(side="left")
        btPlus.pack(side="left")
        self.majScroll()
```



```

# rendre l'objet suscriptable (utiliser les crochets [ et ] )
def __getitem__(self, clef) :
    try :
        return self.__subscript[clef]
    except :
        print (clef, "n'est pas une option")

def __setitem__(self, clef, valeur) :
    try :
        self.__subscript[clef] = valeur
        if clef=="scrollcommand" :
            self.majScroll()
    except :
        if clef in self.__subscript__ :
            print ("erreur dans l'affectation")
        else :
            print (clef, "n'est pas une option")
# configure classique
def configure (self, **arg) :
    if arg :
        for k in arg :
            self.__setitem__(k, arg[k])
            if k=="scrollcommand" :
                self.majScroll()
    else :
        return self.__subscript
config = configure
def cget(self, k) :
    return self.__subscript[k]
def keys(self) :
    return self.__subscript.keys()

# méthode de mise à jour du label et de l'ascenseur
def majScroll (self) :
    self.varLabel.set(str(self.ratio))
    commande = self.__subscript["scrollcommand"]
    if commande :
        sup = self.__subscript["sup"]
        commande ((self.ratio-1)/sup,self.ratio/sup)
# incrémentation/décrémentation du compteur
def plus (self) :
    if self.ratio == self.__subscript["sup"] :
        return
    self.ratio += 1
    self.majScroll()

def moins (self) :
    if self.ratio == 1 :

```

```

        return
    self.ratio -= 1
    self.majScroll()
# équivalence de xview() / yview() des widgets scrollables
def numScrollVue (self,u=None,v=None,w=None) :
    if u :
        if u=="scroll" :
            if v == "1" :
                self.plus()
            else :
                self.moins()
        else : # u == "moveto"
            sup = self.__subscript["sup"]
            local = math.ceil(float(v)* sup)
            if local > 0 and local <= sup :
                self.ratio = local
                self.majScroll()
    else :
        sup = self.__subscript["sup"]
        return ((self.ratio-1)/sup,self.ratio/sup)
# fin de < class NumScroll (Frame) >

leCompteur = NumScroll (leCadre, font = maFonte)
leCompteur.pack()

barreScroll = Scrollbar(leCadre, orient="horizontal", bg="#eaeaea",
    troughcolor="#808080", width=30, command=leCompteur.numScrollVue)
barreScroll.pack(fill=X)
# leCompteur["scrollcommand"] = barreScroll.set
leCompteur.config(scrollcommand=barreScroll.set)
barreScroll.pack()

racine.mainloop()

# fichier : tk12ex00

```

tk13 : Label

Le widget Label peut afficher soit du texte unicode en multiligne, soit du texte `StringVar`, soit un bitmap (`BitmapImage` au sens de **tkinter**), soit une image (`PhotoImage` au sens de **tkinter**).

1. le constructeur

`Label(master=None, options)`

`master` est la référence au conteneur ; par défaut, c'est la fenêtre principale. On peut ajouter après le conteneur un nom en clair (`name=xxx`), comme pour tout widget.

2. les attributs

2.1. liste des attributs

`activebackground, activeforeground, anchor, background, borderwidth, cursor, disabledforeground, font, foreground, highlightbackground, highlightcolor, highlightthickness, image, justify, padx, pady, relief, takefocus, text, textvariable, underline, wraplength, height, state, width`

2.2. les attributs spécifiques

- * `bitmap` : permet d'afficher un bitmap. Sa valeur est un bitmap au sens de **tkinter**. Voir chapitre **tk02**
- * `image` : permet d'afficher une image `PhotoImage`. Voir chapitre **tk02**
- * `text` : permet d'afficher un texte unicode (la fin de ligne est `\n`)
- * `textvariable` : associe le texte du label à une variable de type `StringVar`. Si la variable change de valeur, le texte change. Voir **chapitre tk01 §7**. Attention à l'utilisation des `StringVar` : il faut créer la variable, l'affecter par la méthode `set()`, la consulter par la méthode `get()`. Voir le **chapitre tk01**.

```
xLabel = StringVar()
leLabel = Label(fenPr, textvariable=xLabel, font=maFonte)
-----
xLabel.set("appel de [mainloop]")
print (xLabel.get())
```

- * `justify` : permet d'aligner le texte ; les valeurs sont : `LEFT` ("left") par défaut, `Center` ("center") et `Right` ("right")
- * `anchor` : positionne le texte sur le widget. Pour les valeurs voir **chapitre tk01**
- * `underline` : par défaut, -1 signifie qu'il n'y a pas de soulignage. Sinon prend la valeur entière `n` où `n` est le numéro du caractère souligné.
- * `font` : référence de fonte. Voir **chapitre tk03 fontes**
- * `width` : largeur de texte retenue pour le widget, exprimé en caractères ; dépend de `font`. Si le widget ne contient pas de texte `width` est en pixels.
- * `height` : même chose que pour `width`.
- * `activebackground` : couleur de fond pour state posé à "active"
- * `activeforeground` : couleur de caractère pour state posé à "active"
- * `wraplength` : indicateur de coupure de la ligne pour les Label avec texte. Attention : ce paramètre est donné en pixels !

La valeur de `state` doit être programmée (voir exemple ci-dessous) ; le survol de souris est, par défaut, sans effet.

exemple :

```

leLabel = Label(laFrame, name= "nomLabel", text="une étiquette",
               font=maFonte, bg="#8080ff", takefocus=True, state="normal",
               activebackground="#808080", activeforeground="red",
               highlightbackground="cyan", highlightcolor="magenta",
               highlightthickness=2)
leLabel.bind("<Enter>", lambda x=None: leLabel.config(state="active"))
leLabel.bind("<Leave>", lambda x=None: leLabel.config(state="normal"))

```

```

'activebackground': ('activebackground', 'activeBackground', 'Foreground',
                    <border object at 0x28733d0>, '#808080'),
'activeforeground': ('activeforeground', 'activeForeground', 'Background',
                    <color object at 0x2873370>, 'red'),
'anchor': ('anchor', 'anchor', 'Anchor', <index object at 0x2873310>, 'center'),
'background': ('background', 'background', 'Background', <border object at 0x28732e0>,
              '#8080ff'),
'bg': ('bg', '-background'),
'bitmap': ('bitmap', 'bitmap', 'Bitmap', '', ''),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x2873250>,
               <pixel object at 0x2873250>),
'compound': ('compound', 'compound', 'Compound',
            <index object at 0x2873220>, 'none'),
            'height': ('height', 'height', 'Height', 0, 0),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'disabledforeground': ('disabledforeground', 'disabledForeground', 'DisabledForeground',
                      <color object at 0x2874390>, '#a3a3a3'),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x2874990>, 'Helvetica -25'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x28746f0>,
              '#000000'),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                       <border object at 0x2874570>, 'cyan'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x28747b0>, 'magenta'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                      <pixel object at 0x2874810>, <pixel object at 0x286e480>),
'image': ('image', 'image', 'Image', '', ''),
'justify': ('justify', 'justify', 'Justify', <index object at 0x2874a50>, 'center')
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x2874690>, <pixel object at 0x2874690>),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x28743f0>, <pixel object at 0x28743f0>),
'relief': ('relief', 'relief', 'Relief', <index object at 0x28745d0>, 'flat'),
'state': ('state', 'state', 'State', <index object at 0x2874870>, 'normal'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '0', 1), 'bd': ('bd', '-borderwidth'),
'text': ('text', 'text', 'Text', '', 'une étiquette'),
'textvariable': ('textvariable', 'textVariable', 'Variable', '', ''),
'underline': ('underline', 'underline', 'Underline', -1, -1),
'width': ('width', 'width', 'Width', 0, 0),
'wraplength': ('wraplength', 'wrapLength', 'WrapLength', <pixel object at 0x2874a20>,
              <pixel object at 0x2874a20>),

```

3. les méthodes spécifiques

Le widget `Label` n'a pas de méthodes spécifiques.

tk14 : Button

Le widget `Button` permet d'intégrer un bouton, avec les caractères usuels, tels que le changement d'aspect par survol, la liaisons spécifique avec un gestionnaire d'événement qui est active pour le clic ou une action clavier (frappe de l'espace), ce qui implique la possibilité, par défaut, de prendre le focus.

1. le constructeur

syntaxe

```
widget = Button(conteneur, options)
```

2. les attributs

2.1. liste des attributs

`activebackground`, `activeforeground`, `anchor`, `background`, `bd`, `bg`, `bitmap`, `borderwidth`, `command`, `compound`, `cursor`, `default`, `disabledforeground`, `fg`, `font`, `foreground`, `height`, `highlightbackground`, `highlightcolor`, `highlightthickness`, `image`, `justify`, `overrelief`, `padx`, `pady`, `relief`, `repeatdelay`, `repeatinterval`, `state`, `takefocus`, `text`, `textvariable`, `underline`, `wrplength`, `width`

2.2. attributs spécifiques

- * `activebackground`, `activeforeground` : semblable à ces options dans `Label`, sauf que l'état "normal" et l'état "active" sont automatiques, de même que `takefocus` qui est posé à `True`.
- * `overrelief` : relief lors d'un survol.
- * `command` : `command` prend comme valeur un gestionnaire d'événement. La difficulté est que le gestionnaire ne doit pas avoir de paramètre `Event`. Pour unifier les gestionnaires d'événement, on peut procéder de deux façons. Supposons donné un gestionnaire classique, appelé `maFonction`, avec comme paramètre `event`. Sa définition serait `def maFonction (event)`.
Soit on change le paramètre en en faisant un paramètre clef valeur, initialisé à `None`. La signature devient `def maFonction (event=None)`. Soit on utilise une `lambda`-fonction :

```
command = lambda : maFonction(None)
```
- * `width` : la largeur du bouton doit pouvoir contenir un texte de `width` caractères si le bouton contient du texte ; sinon, la largeur est définie en pixels.
- * `height` : même chose que pour `width`.
- * `default` : voir le chapitre **tk05 §5**
- * `repeatdelay`, `repeatinterval` : Lorsque le bouton de la souris reste appuyé, il ne se passe rien et la commande est effectuée en relâchant le bouton. Si on pose le `repeatdelay` à une valeur `r` non nulle, et le `repeatinterval` à une valeur `d` non nulle, la commande se répète tous les `d` millisecondes, tant que le bouton reste pressé, après avoir attendu `r` millisecondes pour la première répétition.
- * `compound` : prend les valeurs `LEFT='left'`, `BOTTOM='bottom'`, `RIGHT='right'`, `TOP='top'`, `CENTER='center'`. Permet de placer un `BitmapImage` ou un `PhotoImage` avec du texte ; `compound` donne la position relative du `BitmapImage` ou de la `PhotoImage`. Avec `CENTER`, le texte est superposé à l'image.
- * `justify`, `textvariable`, `underline`, `wrplength` : même chose que pour `Label`. Voir chapitre **tk10 Label**.

```

'activebackground': ('activebackground', 'activeBackground', 'Foreground',
                    <border object at 0x285f950>, '#ececec'),
'activeforeground': ('activeforeground', 'activeForeground', 'Background',
                    <color object at 0x29af510>, '#000000'),
'anchor': ('anchor', 'anchor', 'Anchor', <index object at 0x29b4ac0>, 'center'),
'background': ('background', 'background', 'Background', <border object at 0x29b4a60>,
              'red'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'bitmap': ('bitmap', 'bitmap', 'Bitmap', '', ''),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x29af3c0>,
               <pixel object at 0x2b17670>),
'command': ('command', 'command', 'Command', '', <bytecode object at 0x2b11220>),
'compound': ('compound', 'compound', 'Compound', <index object at 0x29ace40>, 'none'),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'default': ('default', 'default', 'Default', <index object at 0x2a21c00>, 'disabled'),
'disabledforeground': ('disabledforeground', 'disabledForeground', 'DisabledForeground',
                      <color object at 0x29b4a90>, '#a3a3a3'),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x29b4760>, 'Helvetica -25'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x29b47c0>,
              '#000000'),
'height': ('height', 'height', 'Height', 0, 0),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                       <border object at 0x29b4820>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x29b4910>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                      <pixel object at 0x29b49d0>, <pixel object at 0x29b49d0>),
'image': ('image', 'image', 'Image', '', ''),
'justify': ('justify', 'justify', 'Justify', <index object at 0x29b48b0>, 'center')
'overrelief': ('overrelief', 'overRelief', 'OverRelief', '', ''),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x29b4940>, <pixel object at 0x29b4940>),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x29b48e0>, <pixel object at 0x29b48e0>),
'relief': ('relief', 'relief', 'Relief', <index object at 0x29b46d0>, 'solid'),
'repeatdelay': ('repeatdelay', 'repeatDelay', 'RepeatDelay', 0, 0),
'repeatinterval': ('repeatinterval', 'repeatInterval', 'RepeatInterval', 0, 0),
'state': ('state', 'state', 'State', <index object at 0x29b4640>, 'active'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'text': ('text', 'text', 'Text', '', 'un bouton'),
'textvariable': ('textvariable', 'textVariable', 'Variable', '', ''),
'underline': ('underline', 'underline', 'Underline', -1, -1),
'width': ('width', 'width', 'Width', 0, 0),
'wraplength': ('wraplength', 'wrapLength', 'WrapLength', <pixel object at 0x2a1c920>,
              <pixel object at 0x2a1c920>),

```

2. méthodes

* `flash()` : fait flasher le bouton.

* `invoke()` : appelle le gestionnaire d'événement de `command`, et a comme valeur le retour de la commande.

tk15 : Checkbutton

Le `Checkbutton` est appelé aussi **checkbox** ou case à cocher est un widget qui peut caractériser deux états (sélecteur coché, non coché). Il est accompagné d'une étiquette.

1. le constructeur

syntaxe

```
widget = Checkbutton(conteneur, options)
```

2. les attributs

2.1. liste des attributs

`activebackground`, `activeforeground`, `anchor`, `background`, `bd`, `bg`, `bitmap`, `borderwidth`, `command`, `compound`, `cursor`, `disabledforeground`, `fg`, `font`, `foreground`, `height`, `highlightbackground`, `highlightcolor`, `highlightthickness`, `image`, `indicatoron`, `justify`, `offvalue`, `offrelief`, `onvalue`, `onrelief`, `padx`, `pady`, `relief`, `selectcolor`, `selectimage`, `state`, `takefocus`, `text`, `textvariable`, `underline`, `variable`, `width`, `wrplength`

2.2. attributs spécifiques

- * `padx`, `pady` : comme attribut du constructeur, les marges sont intérieures à la bordure éventuelle, qui englobe le sélecteur et le texte.
- * `activebackground`, `activeforeground` : Comme pour le widget `Button`. Concerne tout l'espace du widget, étiquette comprise. Voir **chapitre tk11 § 2.2**
- * `compound`, `bitmap`, `image`, `text`, `textvariable`, `underline`, `justify`, `width`, `height`, `wrplength` : Comme pour le widget `Button`. Voir **chapitre tk11 § 2.2**.
- * `command` : le gestionnaire d'événement est appelé chaque fois que le bouton change d'état. Pour le reste voir la widget `Button`, **chapitre tk11 § 2.2**
- * `indicatoron` : l'attribut posé à `True` signifie que l'indicateur de bouton est affiché ; sinon, le sélecteur du bouton disparaît, tout en maintenant le bouton (utiliser alors `selectimage`, ou `selectcolor` par exemple pour visualiser l'état du bouton)
- * `variable` : variable de contrôle. Valeur `IntVar`, avec 0 pour off (non coché), et 1 pour on (coché), sauf si les attributs `offvalue` et `onvalue` indiquent autre chose (pour `IntVar`, voir le **chapitre tk01**)
- * `offvalue` : valeur associée à `off` (non coché) si la variable de contrôle est à `off`.
- * `onvalue` : valeur associée à `on` (coché) si la variable de contrôle est à `on`.
- * `onrelief`, `offrelief` : relief associé à la variable de contrôle.
- * `selectcolor` : couleur de fond du bouton (texte ou image) en cas de sélection (on) et avec `indicatoron=0`. Sinon, couleur de fond l'indicateur.
- * `selectimage` : Si le widget a une image, cette image peut être changée lors de la mise à `on` du bouton . L'image originale est remplacée par l'image valeur de `selectimage`.
- * `textvariable` : la valeur est une variable de type `StringVar` qui permet de mettre une étiquette au widget. Le changement de la variable entraîne le changement de l'étiquette (pour `StringVar`, voir le **chapitre tk01**)

```
'activebackground': ('activebackground', 'activeBackground', 'Foreground',  
                    <border object at 0x1ee8130>, '#ecec'),  
'activeforeground': ('activeforeground', 'activeForeground', 'Background',  
                    <color object at 0x1ee8070>, '#000000'),  
'anchor': ('anchor', 'anchor', 'Anchor', <index object at 0x1ee7fb0>, 'center'),
```

```

'background': ('background', 'background', 'Background', <border object at 0x1ee8220>,
              '#d9d9d9'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'bitmap': ('bitmap', 'bitmap', 'Bitmap', '', ''),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth',
               <pixel object at 0x1d7e2f0>, <pixel object at 0x1d7e2f0>),
'command': ('command', 'command', 'Command', '', ''),
'compound': ('compound', 'compound', 'Compound', <index object at 0x1ee8340>, 'none'),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'disabledforeground': ('disabledforeground', 'disabledForeground', 'DisabledForeground',
                     <color object at 0x1ee7f50>, '#a3a3a3'),
'font': ('font', 'font', 'Font', <font object at 0x1ee7e90>, 'Helvetica -25'),
'fg': ('fg', '-foreground'),
'foreground': ('foreground', 'foreground', 'Foreground',
              <color object at 0x1ee7e30>, '#000000'),
'height': ('height', 'height', 'Height', 0, 0),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                       <border object at 0x1ee7d70>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x1ee67b0>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                      <pixel object at 0x1ee6000>, <pixel object at 0x1ee6000>),
'image': ('image', 'image', 'Image', '', ''),
'indicatoron': ('indicatoron', 'indicatorOn', 'IndicatorOn', 1, 1),
'justify': ('justify', 'justify', 'Justify', <index object at 0x1ee6060>, 'center')
'offrelief': ('offrelief', 'offRelief', 'OffRelief', <index object at 0x1ee6810>, 'raised'),
'offvalue': ('offvalue', 'offValue', 'Value', '0', '0'),
'onvalue': ('onvalue', 'onValue', 'Value', '1', '1'),
'overrelief': ('overrelief', 'overRelief', 'OverRelief', '', ''),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x1ee6870>, <pixel object at 0x1ee6870>),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x1ee6180>, <pixel object at 0x1ee6180>),
'relief': ('relief', 'relief', 'Relief', <index object at 0x1ee68a0>, 'flat'),
'selectcolor': ('selectcolor', 'selectColor', 'Background', <border object at 0x1ee61e0>,
               '#ffffff'),
'selectimage': ('selectimage', 'selectImage', 'SelectImage', '', ''),
'state': ('state', 'state', 'State', <index object at 0x1ee6240>, 'normal'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'text': ('text', 'text', 'Text', '', 'cocher la case'),
'textvariable': ('textvariable', 'textVariable', 'Variable', '', ''),
'tristatevalue': ('tristatevalue', 'tristateValue', 'TristateValue', '', ''),
'tristateimage': ('tristateimage', 'tristateImage', 'TristateImage', '', ''),
'underline': ('underline', 'underline', 'Underline', -1, -1),
'variable': ('variable', 'variable', 'Variable', '', <parsedVarName object at 0x1ee63c0>),
'width': ('width', 'width', 'Width', 0, 0),
'wraplength': ('wraplength', 'wrapLength', 'WrapLength', <pixel object at 0x1ee6990>,
              <pixel object at 0x1ee6990>),

```

3. méthodes

- * `select()` : met le bouton à `on`.
- * `deselect()` : passe le bouton à `off`.
- * `flash()` : fait flasher le bouton.
- * `toggle()` : inverse le bouton
- * `invoke()` : inverse le bouton et invoque le gestionnaire de `command`.

tk16 : Radiobutton

Le **Radiobutton** est appelé aussi bouton radio est un widget qui peut caractériser deux états (coché, non coché). Il est accompagné d'une étiquette. Mais à la différence des **Checkbutton**, les widgets **Radiobutton** évoluent en groupe : un seul -au plus- dans le groupe est coché à la fois, et cocher un bouton non coché retire la sélection au bouton qui l'avait auparavant.

1. le constructeur

syntaxe

`widget = Radiobutton(conteneur, options)`

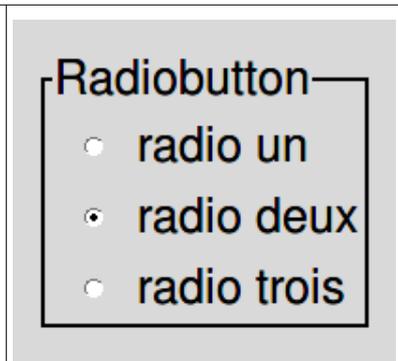
Pour fonctionner correctement, deux options doivent impérativement être posées :

* **variable** : c'est un attribut de type **StringVar** ou **IntVar** ; cette variable est donc dotée d'un getter et d'un setter. Un groupe est l'ensemble des widgets qui relèvent de la même variable.

* **value** : Cet attribut a comme valeur une **chaîne unicode ou un entier**. Les valeurs d'un groupe doivent être distinctes. Si on veut que l'un des boutons soit choisi par défaut, sa valeur (**value**) doit être posée pour **variable** (on doit donc avoir `value == variable.get()`). C'est cette valeur qui est posée pour **variable** en cas de sélection (`variable.set(value)`).

exemple :

```
v = StringVar()
v.set("deux")
radio1 = Radiobutton (laFrame, text= "radio un",
                    variable=v, value="un",
                    font=maFonte).pack(anchor="w")
radio2 = Radiobutton (laFrame, text= "radio deux",
                    variable=v, value="deux",
                    font=maFonte).pack(anchor="w")
radio3 = Radiobutton (laFrame, text= "radio
trois",
                    variable=v, value="trois",
                    font=maFonte).pack(anchor="w")
```



noter le `anchor="w"` qui permet l'alignement à gauche habituel.

2. les attributs

2.1. liste des attributs

`activebackground, activeforeground, anchor, background, bd, bg, bitmap, borderwidth, command, cursor, disabledforeground, fg, font, foreground, height, highlightbackground, highlightcolor, highlightthickness, image, indicatoron, justify, padx, pady, relief, selectcolor, selectimage, state, takefocus, text, textvariable, underline, value, variable, width, wraplength.`

2.2. attributs spécifiques

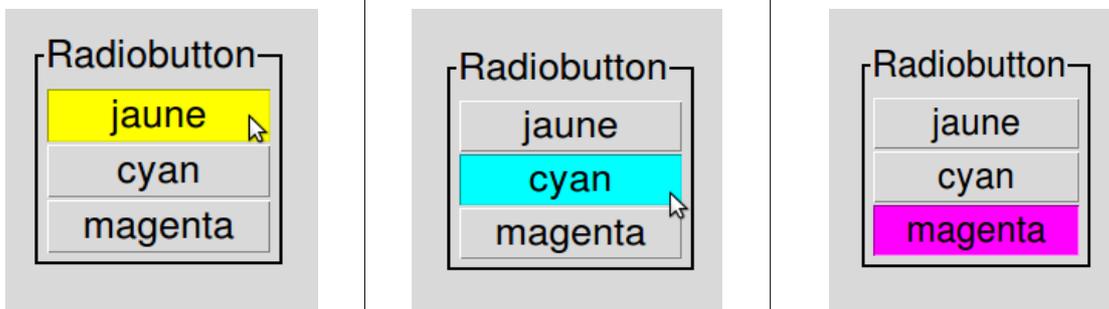
On renvoie au chapitre précédent pour les attributs communs avec le widget **Checkbutton**.

Les deux attributs auxquels il faut prêter attention sont **value** et **variable**, évoqués dans la section constructeur.

* Il faut attirer l'attention sur une présentation particulière des groupes de widget `Radiobutton` avec l'attribut `indicatoron` à `False` (l'indicateur est omis). Comme dans `Checkbox`, la sélection est caractérisée par l'adoption comme couleur de fond de `selectcolor` et éventuellement comme image de `selectimage`.

exemple :

```
v = StringVar()
v.set("#00ffff")
radioJaune = Radiobutton (laFrame,width=10,indicatoron=False,
                          selectcolor="#ffff00", text= "jaune", variable=v,
                          value="#ffff00", font=maFonte).pack()
radioCyan = Radiobutton (laFrame,width=10,indicatoron=False,
                         selectcolor="#00ffff", text= "cyan", variable=v,
                         value="#00ffff", font=maFonte).pack()
radioMagenta = Radiobutton (laFrame,width=10,indicatoron=False,
                            selectcolor="#ff00ff", text= "magenta", variable=v,
                            value="#ff00ff", font=maFonte).pack()"
```



```
'activebackground': ('activebackground', 'activeBackground', 'Foreground',
                    <border object at 0x2400850>, '#ecec'),
'activeforeground': ('activeforeground', 'activeForeground', 'Background',
                    <color object at 0x24007f0>, '#000000'),
'anchor': ('anchor', 'anchor', 'Anchor', <index object at 0x2400790>, 'center'),
'background': ('background', 'background', 'Background', <border object at 0x2400760>,
              '#d9d9d9'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'bitmap': ('bitmap', 'bitmap', 'Bitmap', '', ''),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x24006d0>,
               <pixel object at 0x24006d0>),
'command': ('command', 'command', 'Command', '', ''),
'compound': ('compound', 'compound', 'Compound', <index object at 0x2401780>, 'none'),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'disabledforeground': ('disabledforeground', 'disabledForeground', 'DisabledForeground',
                      <color object at 0x2401ab0>, '#a3a3a3'),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x2401b70>, 'Helvetica -25'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x2401e70>,
              '#000000'),
'height': ('height', 'height', 'Height', 0, 0),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                       <border object at 0x2401930>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
```

```

        <color object at 0x2401c90>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
        <pixel object at 0x2401ed0>, <pixel object at 0x2401ed0>),
'image': ('image', 'image', 'Image', '', ''),
'indicatoron': ('indicatoron', 'indicatorOn', 'IndicatorOn', 1, 1),
'justify': ('justify', 'justify', 'Justify', <index object at 0x2401870>, 'center')
'offrelief': ('offrelief', 'offRelief', 'OffRelief', <index object at 0x2401a50>, 'raised'),
'overrelief': ('overrelief', 'overRelief', 'OverRelief', '', ''),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x2401db0>, <pixel object at 0x2401db0>),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x2401990>, <pixel object at 0x2401990>),
'relief': ('relief', 'relief', 'Relief', <index object at 0x24018d0>, 'flat'),
'selectcolor': ('selectcolor', 'selectColor', 'Background', <border object at 0x2401d50>,
        '#ffffff'),
'selectimage': ('selectimage', 'selectImage', 'SelectImage', '', ''),
'state': ('state', 'state', 'State', <index object at 0x2401ea0>, 'normal'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'text': ('text', 'text', 'Text', '', 'radio un'),
'textvariable': ('textvariable', 'textVariable', 'Variable', '', ''),
'trystateimage': ('trystateimage', 'trystateImage', 'TristateImage', '', ''),
'trystatevalue': ('trystatevalue', 'trystateValue', 'TristateValue', '', ''),
'underline': ('underline', 'underline', 'Underline', -1, -1),
'value': ('value', 'value', 'Value', '', 'un'),
'variable': ('variable', 'variable', 'Variable', 'selectedButton',
        <parsedVarName object at 0x2297e80>),
'width': ('width', 'width', 'Width', 0, 0),
'wraplength': ('wraplength', 'wrapLength', 'WrapLength', <pixel object at 0x2401ba0>,
        <pixel object at 0x2401ba0>),

```

3. méthodes

- * `select()` : met le bouton à on.
- * `deselect()` : passe le bouton à off.
- * `flash()` : fait flasher le bouton.
- * `toggle()` : inverse le bouton
- * `invoke()` : inverse le bouton et invoque le gestionnaire de `command`.

tk17 : Listbox

Le widget `Listbox` permet d'afficher une séquence de chaînes de caractère ; chaque item constitue une ligne du widget. Les items sont indexés. Un widget `Listbox` permet de sélectionner un ou plusieurs items, de récupérer leur référence. Il peut être doté d'ascenseurs (horizontal ou vertical)

1. le constructeur

```
widget = Listbox (master=None, options)
```

2. les attributs

2.1. liste des attributs

`activestyle`, `background`, `bd`, `bg`, `borderwidth`, `cursor`, `disabledforeground`, `exportselection`, `fg`, `font`, `foreground`, `height`, `highlightbackground`, `highlightcolor`, `highlightthickness`, `listvariable`, `relief`, `selectbackground`, `selectborderwidth`, `selectforeground`, `selectmode`, `setgrid`, `state`, `takefocus`, `width`, `xscrollcommand`, `yscrollcommand`

2.2. les attributs spécifiques

style de ligne active :

* `activestyle` : les valeurs possibles sont :

'none' : la ligne active n'a aucune particularité.

'underline' : la ligne active est soulignée.

'dotbox' : la ligne active est dans un rectangle pointillé.

La ligne active est la dernière ligne où se trouve la souris lorsque son Bouton 1 est relâché.

attributs de sélection :

* `exportselection` : en cas de sélection les lignes sélectionnées sont placées par défaut (`True`) dans le presse-papier. Pour désactiver cette disposition, mettre l'attribut à `False`.

* `selectmode` : Il y a quatre modes de sélection :

`BROWSE='browse'` : Une seule ligne est sélectionnée, celle où la bouton 1 de la souris est relâché. On peut donc enfoncer le bouton, avec la souris sur un autre élément que celui qui sera sélectionné ; la couleur de sélection agit, mais suit un déplacement de la souris. C'est la valeur par défaut.

`SINGLE = 'single'` : Une seule ligne est sélectionnée, celle où la bouton 1 de la souris est enfoncé. La sélection ne suit pas la souris et on peut avoir une ligne active autre que la ligne sélectionnée.

`MULTIPLE='multiple'` : Un clic sur n'importe quelle ligne inverse le choix de cette ligne. On peut répéter sur autant de lignes que l'on veut.

`EXTENDED='extended'` : Les lignes sélectionnées sont celle balayées par la souris entre l'enfoncement du bouton 1 et son relâchement.

attributs d'état :

* `state` : il n'y a que deux état, `NORMAL='normal'` et `DISABLED='disabled'`.

* `listvariable` : l'attribut prend comme valeur une variable de type `StringVar`. La liste affichée et la variable sont maintenues en concordance. La chaîne ressemble à un affichage de tuple.

attributs de scroll :

* `xscrollcommand` : s'il y a un ascenseur horizontal, permet de faire la liaison vers celui-ci.

* `yscrollcommand` : s'il y a un ascenseur vertical, permet de faire la liaison vers celui-ci.

```
'activestyle': ('activestyle', 'activeStyle', 'ActiveStyle', <index object at 0x1928360>,
```

```

        'none'),
'background': ('background', 'background', 'Background', <border object at 0x1928240>,
               '#ffffff'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x1928090>, 1),
'cursor': ('cursor', 'cursor', 'Cursor', '', 'hand2'),
'disabledforeground': ('disabledforeground', 'disabledForeground', 'DisabledForeground',
                       <color object at 0x19280f0>, '#a3a3a3'),
'exportselection': ('exportselection', 'exportSelection', 'ExportSelection', 1, 1),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x19281e0>, 'Helvetica -25'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x19282a0>,
               '#000000'),
'height': ('height', 'height', 'Height', 10, 15),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                        <color object at 0x1928270>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x1928210>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                       <pixel object at 0x19281b0>, 1),
'listvariable': ('listvariable', 'listVariable', 'Variable', '', ''),
'relief': ('relief', 'relief', 'Relief', <index object at 0x1927fa0>, 'sunken')
'selectbackground': ('selectbackground', 'selectBackground', 'Foreground',
                    <border object at 0x1927ee0>, '#d0d0d0'),
'selectborderwidth': ('selectborderwidth', 'selectBorderWidth', 'BorderWidth',
                     <pixel object at 0x1927f10>, 0),
'selectforeground': ('selectforeground', 'selectForeground', 'Background',
                    <color object at 0x1927100>, '#d00000'),
'selectmode': ('selectmode', 'selectMode', 'SelectMode', 'browse', 'single'),
'setgrid': ('setgrid', 'setGrid', 'SetGrid', 0, 0),
'state': ('state', 'state', 'State', <index object at 0x19901f0>, 'normal'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'width': ('width', 'width', 'Width', 20, 40),
'xscrollcommand': ('xscrollcommand', 'xScrollCommand', 'ScrollCommand', '', ''),
'yscrollcommand': ('yscrollcommand', 'yScrollCommand', 'ScrollCommand', '', '23204352set'),

```

3. les méthodes spécifiques

méthodes utilitaires :

* **index (index)** : retourne l'index (entier) identifié par **index**. Les valeurs pour **index** peuvent être :

ACTIVE : ligne active.

ANCHOR : première ligne sélectionnées.

"@**x,y**" : **x** et **y** sont les coordonnées d'un point du widget.

END : index après la fin du texte (longueur de la liste).

* **bbox (index)** : retourne la boîte englobante de la ligne d'index donnée. C'est un quadruplet, de la forme (**xoffset**, **yoffset**, **width**, **height**) avec le pixel pour unité, dans la fenêtre visible du widget. Si la ligne n'est pas visible, retourne **None**. Partiellement visible, la boîte se rapporte à ce qui est visible.

* **nearest (y)** : donne l'index de la ligne contenant le point d'ordonnée **y**, ou la ligne la plus proche.

méthodes d'édition :

* **delete (first, last=None)** : supprime les lignes d'index entre **first** et **last**. **Attention**, **last** est inclus, contrairement aux habitudes de Python. Si **last=None**, une seule ligne est ôtée.

* **get (first, last=None)** : retourne un tuple des textes des lignes, la dernière comprise. Voir la

remarque pour `delete()`.

* `insert(index, *elements)` : insère un ou plusieurs éléments à partir de la position `index`. `index` peut être un nombre, ou une constante vue dans `index()`. Les éléments déjà présents sont décalés.

méthodes de défilement :

* `scan_mark(x, y)` : mémorise le point de coordonnées (x,y)

* `scan_dragto(x, y)` : ajuste la vue du widget à 10 fois la différence avec les coordonnées données par `scan_mark()`. Permet un déplacement rapide.

* `see(index)` : scrolle pour rendre la ligne `index` visible.

méthodes de sélection :

* `activate(index)` : active la ligne d'`index` donné.

* `curselection()` : retourne un tuple des indices des lignes sélectionnées.

* `selection_anchor(index)` : ligne `index` est marquée **ANCHOR** (début de sélection)

* `selection_clear(first, last=None)`, `select_clear(first, last=None)` : ôte la sélection entre `first` et `last` ; ne fait rien s'il n'y a pas de sélection. Attention les bornes sont incluses. Une seule ligne est désélectionnée si `last=None`.

* `selection_include(index)`, `select_include(index)` : retourne un booléen si la ligne `index` est dans la sélection.

* `selection_set(first, last=None)`, `select_set(first, last=None)` : pose la sélection entre `first` et `last`. **Attention** les bornes sont incluses Une seule ligne est désélectionnée si `last=None`.

* `size()` : retourne le nombre d'éléments du widget.

méthodes héritées de XView et YView:

Les méthodes héritées de `XView` et `YView` sont partagés avec `Canvas`, `Text`, `Entry` et `Listbox`. Voir le chapitre tk 12 portant sur le scroll. Ce sont elles qui sont explicitées maintenant. Elles permettent de rechercher et de changer la position dans une fenêtre de widget. Elle utilisent des constantes : `MOVETO = "moveto"`, `SCROLL = "scroll"`, `UNITS = "units"`, `PAGES = "pages"`.

* `xview(*arg)` : recherche et change la position horizontale de la vue. Voir les méthodes qui suivent.

* `xview_moveto(fraction)` : c'est la même chose que `xview(MOVETO, fraction)`. Cette méthode est destinée au widget de **scroll** horizontal. Elle déplace la vue sur le texte du widget à la position définie par `fraction`. `fraction = 0.0` correspond à la gauche, et `fraction = 1.0` est l'extrême droite.

* `xview_scroll(n, units)` : même chose que `xview(SCROLL, n, what)`. Déplace la vue sur le texte vers la gauche ou la droite. `n`, entier relatif donne l'ampleur du déplacement (positif : à *droite* et négatif : à *gauche*) et **units** donne le genre de déplacement qui peut être `UNITS` (caractères) ou `PAGES` (pages).

* `yview(*arg)`, `yview_moveto(fraction)`, `yview_scroll(n, units)` : semblables aux précédentes, mais dans le sens vertical.

méthodes de configuration :

* `itemcget(index, option)` : retrouve la valeur d'une option pour une ligne d'`index` donné.

* `itemconfigure(index, options)`, `itemconfig(index, options)` : permet de configurer la ligne `index`. Les options possibles sont `background`, `bg`, `foreground`, `fg`, `selectbackground`, `selectforeground`.

4. un exemple de widget Listbox avec ascenseur

L'**objectif du script** est de réaliser la visualisation de la liste des noms des fontes disponibles sur le système et donner pour chaque item sélectionné un aperçu des caractères, en affichant son nom à l'aide de ses propres glyphes.

```

#!/usr/bin/python3
#-*- coding: utf-8 -*-

from tkinter import *
from tkinter.font import families
maFonte = "Helvetica -25"

# fonction pour quitter
def quitter():
    racine.quit()

# fonction d'affichage lors de la sélection d'une ligne
def valider(event=None):
    # retour d'un-uplet de chaînes, ici un singleton
    index = listBoxFontes.curselection()
    if (index == ()):
        return
    index = int (index[0])
    nomDeFonte = listBoxFontes.get(index)
    # pour les noms avec espace : "{Courier New} -30"
    fonte = "{" + nomDeFonte + "} -30"
    affLabel.configure (text=nomDeFonte, font = fonte)

# la fenêtre racine comporte 3 widget packés :
# - cadreListe : pour la Listbox et son ascenseur
# - cadreSpecimen : pour l'affichage du spécimen
# - le Bouton pour quitter (non identifié)
racine = Tk()
racine.protocol("WM_DELETE_WINDOW", quitter)

# cadre pour l'ascenseur et la Listbox
cadreListe = Frame (racine, borderwidth=2, relief=SOLID, bg="#c0c0c0")
cadreListe.pack()

# le widget Listbox et son initialisation
listBoxFontes = Listbox (cadreListe, font=maFonte,
                        width = 40, height = 15, activestyle = "none",
                        selectbackground="#d0d0d0", selectforeground="#d00000",
                        selectmode="single", cursor="hand2")
listBoxFontes.bind("<ButtonRelease-1>", valider) # relaché
listBoxFontes.pack(padx=10, pady=10, side=LEFT)

fontes= list(families (racine))
fontes.sort()
for item in fontes :
    listBoxFontes.insert(END, item)

# ascenseur vertical

```

```

ascenseur = Scrollbar (cadreListe,bg="red")
ascenseur.pack(side=LEFT, fill=Y, padx=5)
listBoxFontes['yscrollcommand'] = ascenseur.set
ascenseur['command'] = listBoxFontes.yview

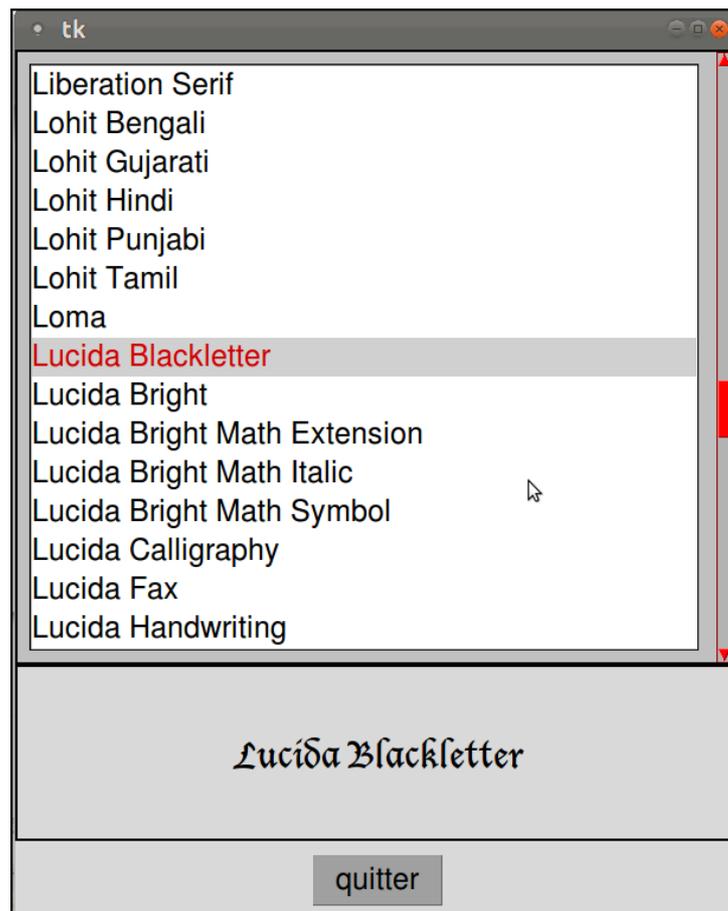
# cadre pour l'affichage du spécimen
cadreSpecimen = Frame(racine, height=150, bd=2, relief=SOLID)
cadreSpecimen.pack(fill=X)
affLabel = Label (cadreSpecimen, text = "")
affLabel.place(relx=0.5, rely=0.5, anchor=CENTER)

# bouton pour quitter
Button (racine, text = " quitter " , font=maFonte, bg="#a0a0a0",
        command = quitter).pack(padx=10, pady=10)

racine.mainloop()
racine.destroy()

# fichier tk16ex00.py

```



tk18 : OptionMenu

1. le constructeur

```
widget = OptionMenu(master, variable, value, *values,  
                    command=lambda x : None)
```

`variable` est une variable de type `StringVar` ; `value` est une chaîne unicode. Noter qu'il y a au moins 4 arguments lors de l'appel. **Il n'y a pas d'option dans le constructeur.**

On peut renseigner le dernier argument, `command` par une fonction avec un argument fonction ; celui par défaut ne fait rien. La commande est appelée lorsqu'une sélection (clic) est effectuée sur la partie déroulante (ce peut être le choix actuel). La valeur réelle de l'argument est la valeur qui a été choisie. Comme on peut le constater, cela n'a rien à voir avec l'attribut `command` habituel. La commande n'est pas appelée pour un item qui dispose de sa propre commande.

Si `listeValeurs` désigne une liste d'items (`list` ou `tuple`) avec au moins trois éléments de bon type,, on peut formuler ainsi le constructeur :

```
widget = OptionMenu(master, *listeValeurs, command=lambda x : None)
```

Le widget `OptionMenu` est ce que l'on appelle aussi **Combobox**. Elle comporte deux parties, l'une qui est toujours visible, avec une partie texte et un sélecteur latéral. Le clic sur le sélecteur déroule le menu et la validation d'un item le fait disparaître, avec comme texte permanent la valeur associée à l'item choisi.

2. les attributs

Les attributs classiques (comme `font`, `relief`) s'appliquent à **la partie permanente** et pas à la liste qui se déroule.

2.1. liste des attributs

```
activebackground, activeforeground, anchor, background, bd, bg, bitmap,  
borderwidth, compound, cursor, direction, disabledforeground, fg, font,  
foreground, height, highlightbackground, highlightcolor,  
highlightthickness, image, indicatoron, justify, menu, padx, pady, relief,  
state, takefocus, text, textvariable, underline, width, wraplength
```

La modification des attributs par défaut se fait de façon habituelle (`config()` et `cget()`), à la différence près qu'elle ne peut être initiée dans le constructeur.

2.2. les attributs spécifiques

- * `direction` : les valeurs possibles sont `"above"`, `"below"`, `"flush"`, `"left"`, or `"right"`.
L'attribut indique où doit se placer la partie déroulante par rapport à celle toujours visible.
- * `menu` : lors de sa création, l'`OptionMenu` crée un widget `Menu` assez élémentaire ; on a évidemment accès à ce menu, à ses propres attributs et méthodes. Mais si on donne une commande spécifique à un item du menu, alors le `command` du widget est inhibé, et il faut gérer à la main la synchronisation de la variable. (Voir l'exemple en section 3)
- * `indicatoron` : il s'agit de montrer ou non le sélecteur de la partie toujours visible.
- * `variable` : Voir dans la section constructeur.
- * `command` : Voir dans la section constructeur.
- * `image`, `bitmap` : en principe, c'est la valeur actuellement choisie qui apparaît dans la partie visible de texte. Le texte peut être remplacé par un bitmap ou une image.
- * `textvariable` : nom de la variable interne (ex : `PY_VAR0`) qui stocke la valeur de l'option sélectionnée (exemple proposé en section 3). Voir le widget `Radiobutton` (**chapitre tk16**)

```

'activebackground': ('activebackground', 'activeBackground', 'Foreground',
    <border object at 0x15bce30>, '#ecec'),
'activeforeground': ('activeforeground', 'activeForeground', 'Background',
    <color object at 0x15bcd0>, '#000000'),
'anchor': ('anchor', 'anchor', 'Anchor', <index object at 0x15bcd70>, 'center'),
'background': ('background', 'background', 'Background', <border object at 0x15bcd40>,
    '#d9d9d9'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'bitmap': ('bitmap', 'bitmap', 'Bitmap', '', ''),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x15bccb0>, 2),
'compound': ('compound', 'compound', 'Compound', <index object at 0x15bde20>, 'none'),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'direction': ('direction', 'direction', 'Direction', <index object at 0x15bcc50>, 'below'),
'disabledforeground': ('disabledforeground', 'disabledForeground', 'DisabledForeground',
    <color object at 0x15bcc20>, '#a3a3a3'),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x15bcbc0>, 'TkDefaultFont'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x15bcb90>,
    '#000000'),
'height': ('height', 'height', 'Height', '0', '0'),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
    <color object at 0x15bcb30>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
    <color object at 0x15bcb00>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
    <pixel object at 0x15bcad0>, 2),
'image': ('image', 'image', 'Image', '', ''),
'indicatoron': ('indicatoron', 'indicatorOn', 'IndicatorOn', 0, 1),
'justify': ('justify', 'justify', 'Justify', <index object at 0x15bdc40>, 'center')
'menu': ('menu', 'menu', 'Menu', '', '.20272016.menu'),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x15be240>, 5),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x15bdfa0>, 4),
'relief': ('relief', 'relief', 'Relief', <index object at 0x15be2a0>, 'raised'),
'state': ('state', 'state', 'State', <index object at 0x15bdd60>, 'normal'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '0', '0'),
'text': ('text', 'text', 'Text', '', 'un'),
'textvariable': ('textvariable', 'textVariable', 'Variable', '', 'PY_VAR0'),
'underline': ('underline', 'underline', 'Underline', -1, -1),
'width': ('width', 'width', 'Width', '0', '0'),
'wraplength': ('wraplength', 'wrapLength', 'WrapLength', <pixel object at 0x15bde80>, 0),

```

3. un exemple de widget OptionMenu

```

#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *
import sys
from tkinter.messagebox import showinfo
maFonte = "Helvetica -25"
maFonteTitre = "Helvetica -35"

# fenêtre principale
racine = Tk()

```

```

racine.title("Menus")
racine.protocol("WM_DELETE_WINDOW", racine.quit)
racine.geometry("600x300+300+300")

# bouton quitter
btQuitter = Button(racine, font=maFonte, text="QUITTER",
                  bg="#c0c0c0", command=racine.quit)
btQuitter.pack(pady=20, side=BOTTOM)

# la commande pour les items sans commande
def clic(v) :
    print("*****", v || ', "*****", v)
    print('monMenu["text"] || ', monMenu["text"] )
    print('maVariable.get() || ',maVariable.get())
    print('monMenu.getvar(monMenu["textvariable"]) || ',
          monMenu.getvar(monMenu["textvariable"]))

# titre
monTitre = Label (racine, text="exemple de widget OptionMenu",
                 font = maFonteTitre)
monTitre.pack(pady=10)

# le widget OptionMenu
maVariable = StringVar()
item = ["version de tk", "version de Python", "plateforme", "par défaut"]
maVariable.set(item[1]) # par défaut
monMenu = OptionMenu(racine, maVariable, *item , command=clic)
monMenu.pack(padx=10, pady=5, side=LEFT)

#---- les options de la partie visible
monMenu["font"]=maFonte
monMenu["padx"]=5 # bordure interne
monMenu["pady"]=5 # bordure interne
monMenu["relief"]=SOLID
monMenu["direction"] = "right"

#---- option pour le menu
monMenu["menu"]["font"]=maFonte

#---- options pour chaque item
def commandConfig (x, y):
    maVariable.set(item[x])
    showinfo(item[x], item[x]+" :\n"+y)
monMenu["menu"].entryconfig (0, command =
                             lambda y= str(TkVersion): commandConfig(0,y))
monMenu["menu"].entryconfig (1, command =
                             lambda y=sys.version: commandConfig(1,y))
monMenu["menu"].entryconfig (2, command =

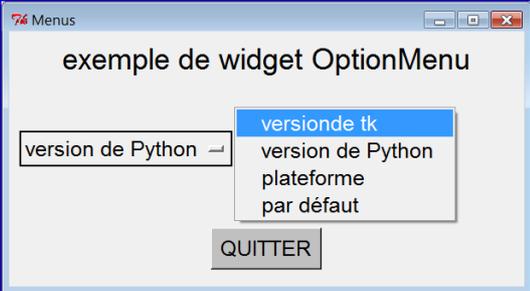
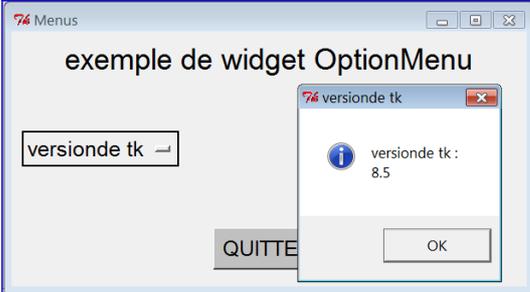
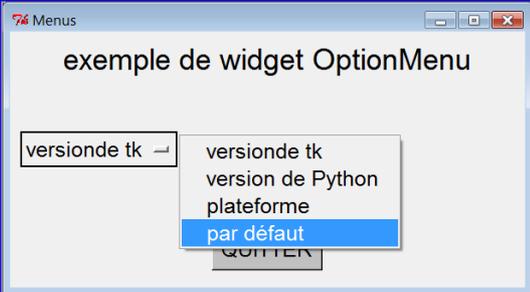
```

```

lambda y=str(TkVersion): commandConfig(2,y)
# boucle de la fenetre principale
racine.mainloop()
racine.destroy()
# fichier tk18ex00.py

```

résultats :

```

"*****", v || ***** par défaut
monMenu["text"] || par défaut
maVariable.get() || par défaut
monMenu.getvar(monMenu["textvariable"]) || par défaut

```

4. les méthodes

Il n'y a pas de méthodes spécifiques à `OptionMenu`

Avertissement : Le widget `MenuButton` est désormais considéré comme obsolète. Il est encore présent, mais uniquement pour assurer en interne `OptionMenu`.

tk19 : Entry

Le widget `Entry` permet de saisir, voir, modifier (sélectionner, copier, coller, remplacer, insérer) une ligne de texte.

1. le constructeur.

syntaxe :

```
widget = Entry (conteneur, options)
```

Une ligne est une séquence de caractères, indexés à partir de 0, qui peut contenir une sélection et porte un curseur spécifique. On repère une sous-séquence en donnant l'indice de son premier élément et celui du caractère après la fin de la sous-séquence.

Les constantes d'index suivantes ont une valeur constamment mise à jour :

* `ANCHOR = 'anchor'` : index du premier élément sélectionné (voir la méthode `select_from()`)

* `END = 'end'` : index qui vient juste après le dernier caractère de la ligne ; il est égal à la longueur de la ligne ; `range(0, END)` est la séquence des indices des caractères de la ligne.

* `INSERT = 'insert'` : index de la position courante du curseur.

2. les attributs

2.1. liste des attributs

`background, bd, bg, borderwidth, cursor, disabledbackground, disabledforeground, exportselection, fg, font, foreground, highlightbackground, highlightcolor, highlightthickness, insertbackground, insertborderwidth, insertofftime, insertontime, insertwidth, invalidcommand, invcmd, justify, readonlybackground, relief, selectbackground, selectborderwidth, selectforeground, show, state, takefocus, textvariable, validate, validatecommand, vcmd, width, xscrollcommand`

2.2. les attributs spécifiques

attribut d'états :

* `state` : cet attribut est adapté au widget `Entry`. L'état `DISABLED="disabled"` caractérise ici un widget hors service ; l'affichage reste lisible, sans plus. L'état `NORMAL="normal"` est l'état normal de saisie. Il n'y a pas d'état `ACTIVE="active"`. Par contre il y a un état `"readonly"` (il n'y a pas de constante associée) qui est l'état à lecture seulement, la sélection et la copie restent possible, pas le coller ou l'insertion.

accès au texte :

* `textvariable` : permet un accès direct au texte du widget et la modification de la variable affectée. **Attention**, il s'agit d'une variable `StringVar`.

```
texte = StringVar()
texte.set ("papa est en voyage")
laSaisie = Entry (fenPr, textvariable=texte)
laSaisie.pack (pady=10)
...
print (texte.get() # évolue avec le contenu du widget Entry
```

* `show` : permet de remplacer le caractère d'écho (usuellement, le caractère frappé est affiché) par un caractère unique en écho. Le `textvariable` n'est pas affecté par cet attribut. L'exemple classique est la saisie d'un mot de passe, pour lequel on pose `show="*"`

attributs de sélection :

- * **exportselection** : lorsque l'on sélectionne du texte, il peut être automatiquement mis dans le presse-papier (si l'attribut est posé à **True**, valeur par défaut) ou non (l'attribut est posé à **False**).
- * **selectbackground** : couleur de fond de la sélection.
- * **selectborderwidth** : largeur de la bordure de la zone de sélection.
- * **selectforeground** : couleur des caractères de la sélection.

attributs de curseur :

- * **insertwidth** : le curseur d'insertion est un rectangle dont on peut modifier la largeur (2 pixels par défaut).
- * **insertbackground** : le curseur d'insertion a une couleur, le noir par défaut ; on peut changer cette couleur, valeur de l'attribut **insertbackground**.
- * **insertborderwidth** : par défaut, le rectangle du curseur d'insertion n'a pas de bordure. On peut lui donner une bordure avec le relief **RAISED** en posant la largeur de cette bordure. Cela n'est visible que si la largeur du rectangle du curseur est au moins deux fois la largeur de la bordure.
- * **insertofftime** : le curseur d'insertion clignote. L'attribut **insertofftime** donne en millisecondes le temps de disparition du rectangle.
- * **insertontime** : le curseur d'insertion clignote. L'attribut **insertontime** donne en millisecondes le temps ou le rectangle est visible.

validation de saisie : Voir un exemple en section 5

On peut avoir besoin de filtrer les entrées. Par exemple, ne saisir que des valeurs alphanumériques, ou uniquement de caractères majuscules etc.

- * **validate** : indique **quand** se fait la validation au cours de la saisie.

Les valeurs possibles sont les suivantes :

None : aucune validation ; valeur par défaut.

"focus" : validation lorsque le widget gagne ou perd le focus.

"focusin" : validation lorsque le widget gagne le focus.

"focusout" : validation lorsque le widget perd le focus.

"key" : pour toute modification de la ligne, y compris l'introduction d'une chaîne par défaut ou une modification par **textvariable**.

ALL : pour toutes situations

- * **invalidcommand**, **invcmd** : appelé si **validatecommand** retourne **False**

- * **validatecommand**, **vcmd** : la valeur est une fonction qui est appelée lorsque **validate** est posé et retourne **True** ou **False**. Si c'est **False**, le contenu n'est pas valide et n'est pas retenu.

attribut de scroll :

- * **xscrollcommand** : on peut ajouter un ascenseur horizontal au widget pour manipuler une saisie plus large que celle autorisée par la fenêtre de saisie. Voir la section 4 de ce chapitre.

```
'background': ('background', 'background', 'Background', <border object at 0x2f58ff0>,
               '#ffffff'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
```

```

'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x2f58f30>, 1),
'cursor': ('cursor', 'cursor', 'Cursor', <cursor object at 0x2f58f90>, 'xterm'),
'disabledbackground': ('disabledbackground', 'disabledBackground', 'DisabledBackground',
    <border object at 0x2f58390>, '#d9d9d9'),
'disabledforeground': ('disabledforeground', 'disabledForeground', 'DisabledForeground',
    <color object at 0x2f58ab0>, '#a3a3a3'),
'exportselection': ('exportselection', 'exportSelection', 'ExportSelection', 1, 1),
    'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x2f58a80>, 'TkTextFont'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x2f5a040>,
    '#000000'),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
    <color object at 0x2f58b10>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
    <color object at 0x2f583f0>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
    <pixel object at 0x2f58b40>, 1),
'insertbackground': ('insertbackground', 'insertBackground', 'Foreground',
    <border object at 0x2f58450>, '#000000'),
'insertborderwidth': ('insertborderwidth', 'insertBorderWidth', 'BorderWidth',
    <pixel object at 0x2f58b70>, 0),
'insertofftime': ('insertofftime', 'insertOffTime', 'OffTime', 300, 300),
'insertontime': ('insertontime', 'insertOnTime', 'OnTime', 600, 600),
'insertwidth': ('insertwidth', 'insertWidth', 'InsertWidth', <pixel object at 0x2f58510>, 2),
'invalidcommand': ('invalidcommand', 'invalidCommand', 'InvalidCommand', '', ''),
'invcmd': ('invcmd', '-invalidcommand'),
'justify': ('justify', 'justify', 'Justify', <index object at 0x2f58570>, 'left')
'readonlybackground': ('readonlybackground', 'readonlyBackground', 'ReadOnlyBackground',
    <border object at 0x2f58c00>, '#d9d9d9'),
'relief': ('relief', 'relief', 'Relief', <index object at 0x2f58c30>, 'sunken'),
'selectbackground': ('selectbackground', 'selectBackground', 'Foreground',
    <border object at 0x2f58630>, '#c3c3c3'),
'selectborderwidth': ('selectborderwidth', 'selectBorderWidth', 'BorderWidth',
    <pixel object at 0x2f58690>, 0),
'selectforeground': ('selectforeground', 'selectForeground', 'Background',
    <color object at 0x2f58c90>, '#000000'),
'show': ('show', 'show', 'Show', '', ''),
'state': ('state', 'state', 'State', <index object at 0x2f58cc0>, 'normal'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'textvariable': ('textvariable', 'textVariable', 'Variable', '', ''),
'validate': ('validate', 'validate', 'Validate', <index object at 0x2f58cf0>, 'none'),
'validatecommand': ('validatecommand', 'validateCommand', 'ValidateCommand', '', ''),
'vcmd': ('vcmd', 'validatecommand'),
'width': ('width', 'width', 'Width', 20, 20),
'xscrollcommand': ('xscrollcommand', 'xScrollCommand', 'ScrollCommand', '', ''),

```

3. les méthodes

utilitaires :

* `index(index)` : retourne la valeur numérique de `index`.

L'`index` peut être désigné par des constantes :

`ANCHOR="anchor"` : index du premier caractère de la sélection si elle existe.

`END="end"` : index de la première position après le dernier caractère.

`INSERT="insert"` : index de la position actuelle du curseur d'insertion

`"@x"` : index de la position où `x` est la distance au bord gauche du widget ;

l'approximation se fait «au plus proche». Si *x* est plus loin que le bord droit, retourne l'index relatif au bord droit.

"*n*" : *n* étant un entier, retourne *n*, sauf si *n* est supérieur au nombre de caractères, où l'index est le nombre de caractères. La conversion entier/chaîne est automatique.

méthodes d'édition :

- * `get()` : retourne le texte du widget.
- * `icursor (index)` : met le curseur d'insertion à la position `index` (c'est-à-dire juste avant le caractère en position `index`, compté à partir de 0).
- * `insert (index, string)` : insère la chaîne à l'index désigné.

méthodes de scroll :

- * `scan_mark(x)` : option de scroll rapide même si le widget n'a pas d'ascenseur horizontal. La méthode enregistre la valeur *x* qui est utilisée dans la méthode `scan_dragto(x)`. Voir l'exemple en section 4.
- * `scan_dragto(x)` : méthode liée à `scan_mark()` ; déplace la vue de dix fois la différence entre *x* et la valeur enregistrée par `scan_mark`, ou la valeur précédente si la fonction est appelée plusieurs fois. On utilise usuellement ces méthodes en renseignant `scan_mark()` par la valeur `event.x` d'un événement `Release` et `scan_dragto()` par elle d'un événement `Motion`. Dans l'exemple proposé, il s'agit d'une avance rapide dans le texte, en utilisant le clavier (touches flèche/haut, flèche/bas) au lieu d'événements souris.

méthodes de sélection :

- * `selection_adjust(index)` : règle la sélection de façon à inclure le caractère à la position `index` ; ne fait rien si le caractère appartient déjà à la sélection.
- * `selection_clear()`, `select_clear()` : ôte la sélection ; ne fait rien s'il n'y a pas de sélection.
- * `selection_from(index)`, `select_from(index)` : **limite** la droite de la sélection à `index`.
- * `selection_present()`, `select_present()` : retourne `True` s'il y a une sélection, `False` sinon. **Attention**, car si on prétend utiliser une sélection et qu'il n'y en a pas, on a une erreur et donc un plantage de l'application.
- * `selection_range(start, end)`, `select_range(start, end)` : crée une sélection entre les caractères en positions `start` (inclus) et `end` (exclus).
- * `selection_to(index)`, `select_to(index)` : fait une sélection entre `ANCHOR` et le caractère en position `index` (exclus).

méthodes héritées de XView :

Les méthodes héritées de `XView` sont partagés avec `Canvas` et `Listbox`. Elles permettent de rechercher et de changer la position dans une fenêtre de widget. Elle utilisent des constantes : `MOVETO = "moveto"`, `SCROLL = "scroll"`, `UNITS = "units"`.

- * `xview(*arg)` : recherche et change la position horizontale de la vue. C'est cette méthode qui est utilisée dans la gestion courante de la mise à jour de la vue et de la barre de scroll. Les méthodes qui suivent s'avère peu utiles.
- * `xview_moveto(fraction)` : c'est la même chose que `xview(MOVETO, fraction)`. Cette méthode est destinée au widget de **scroll** horizontal. Elle déplace la vue sur le texte du widget à la position définie par `fraction`. `fraction = 0.0` correspond à la gauche, et `fraction = 1.0` est l'extrême droite.
- * `xview_scroll(n)` : même chose que `xview(SCROLL, n, what)`. Déplace la vue sur le texte vers la gauche ou la droite. *n*, entier relatif donne l'ampleur du déplacement (positif : à droite et négatif : à gauche).

4. scroller le widget Entry

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *
maFonte = "Helvetica -25"

# fonction pour quitter
def quitter():
    racine.quit()

def commande():
    print (v.get())

racine = Tk()
racine.protocol("WM_DELETE_WINDOW", quitter)
racine.geometry("400x300+200+250")

# cadre et boutons
monCadre = Frame(racine)
monCadre.place(rely=0.5, relx=0.5, anchor=CENTER)
btQuitter = Button(racine, font=maFonte, text="Quitter",
                   bg = "grey" , command=quitter)
btQuitter.place(rely=0.9, relx=0.2, anchor=CENTER)
btCommande = Button(racine, font=maFonte, text="Commande",
                    bg = "grey" , command=commande)
btCommande.place(rely=0.9, relx=0.7, anchor=CENTER)

# ascenseur
monAscenseur = Scrollbar (monCadre, orient="horizontal")
monAscenseur.pack(side=BOTTOM,fill=X)
# Entry
v = StringVar()
v.set("azertyuiopazertyuiopazertyuiopazertyuiopazertyuiopazertyuiop\
pazertyuiopazertyuiopazertyuiopazertyuiopazertyuiopazertyuiopazert\
yuiopazertyuiopazertyuiopazertyuiop")
monEntree = Entry(monCadre, width=20, font=maFonte, textvariable=v,
                  xscrollcommand=monAscenseur.set)
monEntree.pack()
monAscenseur["command"] = monEntree.xview
monAscenseur["activerelief"] = GROOVE
monAscenseur["elementborderwidth"] = 5
monAscenseur["troughcolor"] = "#bbffff"
monAscenseur["width"] = 30
monAscenseur["bg"] = "#9999ff"
monAscenseur["bd"] = 4

# scan_mark et scan_dragto au clavier
```

```

def markUp(event=None) :
    monEntree.scan_mark(30)
    monEntree.scan_dragto(35)

def markDown(event=None) :
    monEntree.scan_mark(30)
    monEntree.scan_dragto(25)

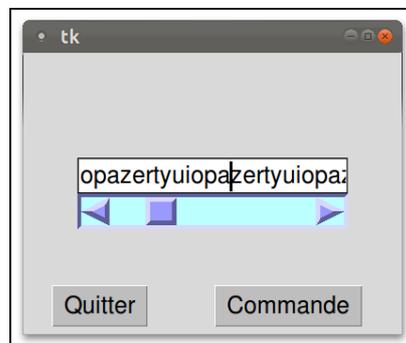
monEntree.bind("<Up>", markUp)
monEntree.bind("<Down>", markDown)

racine.mainloop()
racine.destroy()

# fichier tk15ex00.py

```

résultat :



5. problème de validation : un exemple

schéma de principe :

Les lignes de code qui suivent ne constituent qu'un schéma de principe pour le problème suivant : **ne saisir au clavier que les caractères majuscules, à l'exclusion de tout autre, y compris Back-Space.**

* On utilise le fait que la capture d'événement est prioritaire sur tout appel de commande, ce qui permet de saisir la touche clavier activée et de traiter le filtrage ensuite.

* l'instruction `validate` est retardée, ce qui exclut une chaîne par défaut non vide du contrôle.

* le script serait déficient avec un copier coller.

* par une curiosité non documentée, l'appel du gestionnaire `validatecommand()` dans l'exécution du gestionnaire `invalidcommand()` remet l'attribut `validate` à `"none"`. ce qui explique la nécessité de reformuler le `validate="key"` dans `toucheClavier()`.

```

#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *

# fenêtre principale
fenPrincipale= Tk()
fenPrincipale.title("validate dans Entry")
fenPrincipale.protocol("WM_DELETE_WINDOW", fenPrincipale.quit)

```

```

maFonte = "Courier -25 bold"
# bouton quitter
Button(fenPrincipale, font=maFonte, text="QUITTER",
       command=fenPrincipale.quit).pack(pady=20)

# le filtrage
MAJ = "ABCDEFGHIJKLMNOPQRSTUVWXYZAAEEEEIIIOUUUAAEEEEIIIOUUUC"
MIN = "abcdefghijklmnopqrstuvwxyzàâäéèêëîïôöùûüÀÂÄÉÈÊËÎÏÔÖÙÛÜÇ"

def toucheClavier (event):
    # valide à "none"
    laSaisie.config(validate="key")
    laSaisie.car = event.char

def majuscules() :
    return laSaisie.car in MAJ

def autres() :
    i = MIN.find(laSaisie.car)
    if i < 0 :
        print ("le caractère frappé n'est pas un caractère valide")
    else :
        laSaisie.event_generate("<Key-"+MAJ[i]+">")

laSaisie = Entry (fenPrincipale, font=maFonte, validatecommand=majuscules,
                 # valide = "key", #-- voir toucheClavier()
                 invalidcommand=autres)

laSaisie.car = "" # rappel : une fonction est un objet
laSaisie.focus_set()
laSaisie.bind("<Key>", toucheClavier)
laSaisie.pack(pady=20, padx=20)

# boucle des événements
fenPrincipale.mainloop()
fenPrincipale.destroy() # par précaution

# fichier tk07ex01.py

```

tk20 : Text

Le widget `Text` permet de créer un éditeur de texte formaté dans un programme : on peut utiliser plusieurs fontes et plusieurs couleurs. On peut incorporer une image qui est alors considérée comme un caractère unique. On peut aussi introduire des marqueurs invisibles entre les caractères qui servent de repères ou encore constituent des balises de référence pour le formatage.

L'éditeur peut saisir du texte ; il connaît le passage de ligne sur le clavier principal (pas la touche de saut de ligne du pavé numérique), la tabulation et les touches fléchées. Il fonctionne par insertion, pas par remplacement. Il dispose des propriétés usuelles du presse-papier.

1. le constructeur

```
widget = Text(master, options)
```

2. les attributs

2.1. liste des attributs

```
autoseparators, background, bd, bg, blockcursor, borderwidth, cursor,
endline, exportselection, fg, font, foreground, height,
highlightbackground, highlightcolor, highlightthickness,
inactiveselectbackground, insertbackground, insertborderwidth,
insertofftime, insertontime, insertwidth, maxundo, padx, pady, relief,
selectbackground, selectborderwidth, selectforeground, setgrid, spacing1,
spacing2, spacing3, startline, state, tabs, tabstyle, takefocus, undo,
width, wrap, xscrollcommand, yscrollcommand)
```

2.2. les attributs spécifiques

attributs de dimensions :

* `height`, `width` : les dimensions sont données en caractères et les valeurs en pixels sont donc conditionnées par la fonte de base.

* `spacing1` : espacement avant un bloc de texte ayant le tag de même nom.

* `spacing2` : espacement entre les lignes d'un un bloc de texte ayant le tag de même nom.

* `spacing3` : espacement à la fin d'un un bloc de texte ayant le tag de même nom.

* `wrap` : indicateur de passage automatique à la ligne. Les trois valeurs possibles sont "`char`", "`none`" et "`word`" ;

* `tabs` : valeurs de tabulation. **Exemple** : ("`2c`", "`5c`", "`9c`") définit les 3 premières tabulations à 2cm, 5cm, 9cm. Les tabulations suivantes se calculent par sauts de (9cm - 5cm), différences des deux dernières (éviter les espaces dans l'expression du tuple).

* `tabstyle` : peut être "`tabular`" ou "`wordprocessor`".

attributs de retour sur commande :

* `maxundo` : mettre -1 pour un nombre illimité de possibilité de retour su commande (`undo`). Sinon, c'est le nombre de commandes autorisées pour le mécanisme undo. Zéro par défaut.

* `undo` : à `True`, le mécanisme de retour sur commande est actif. `False` par défaut.

* `autoseparators` : les séparateurs interviennent dans le mécanisme `undo`. Si l'attribut est `True`, un séparateur est automatiquement inséré après chaque action. Sinon, les séparateurs doivent être ajoutés avec la méthode

attributs de curseur :

* `blockcursor` : `True` ou `False`. `True` correspond à un curseur épais, `False` un curseur fin.

- * **insertwidth** : le curseur d'insertion est un rectangle dont on peut modifier la largeur (c'est 2 pixels par défaut).
- * **insertbackground** : le curseur d'insertion a une couleur, noire par défaut ; on peut changer cette couleur, valeur de l'attribut **insertbackground**.
- * **insertborderwidth** : par défaut, le rectangle du curseur d'insertion n'a pas de bordure. On peut lui un bordure avec le relief **RAISED** en posant la largeur de cette bordure. Cela n'est visible que si la largeur du rectangle du curseur est au moins deux fois la largeur de la bordure.
- * **insertofftime** : le curseur d'insertion clignote. L'attribut **insertofftime** donne en millisecondes le temps de disparition du rectangle.
- * **insertontime** : le curseur d'insertion clignote. L'attribut **insertontime** donne en millisecondes le temps ou le rectangle est visible.

attribut de sélection automatique :

- * **exportselection** : par défaut, l'attribut est posé à **True** ; toute sélection passe automatiquement dans le presse-papier. Ce comportement par défaut est inhibé si on pose cet attribut à **False**.

attribut d'état :

- * **state** : il y a deux états **NORMAL="normal"** et **DISABLED="disabled"**

attributs de scroll :

- * **xscrollcommand** : s'il y a un ascenseur horizontal, permet de faire la liaison vers celui-ci.
- * **yscrollcommand** : s'il y a un ascenseur vertical, permet de faire la liaison vers celui-ci.

exemple : valeurs par défaut.

```
'autoseparators': ('autoseparators', 'autoSeparators', 'AutoSeparators', 1, 1),
'background': ('background', 'background', 'Background', <border object at 0x1lcb700>,
               '#ffffff'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'blockcursor': ('blockcursor', 'blockCursor', 'BlockCursor', 0, 0),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x1lc42090>, 1),
'cursor': ('cursor', 'cursor', 'Cursor', <cursor object at 0x1lc42540>, 'xterm'),
'endline': ('endline', '', '', '', '')
'exportselection': ('exportselection', 'exportSelection', 'ExportSelection', 1, 1),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x1d35010>, 'TkFixedFont'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x1d33d20>,
               '#000000'),
'height': ('height', 'height', 'Height', <pixel object at 0x1d33cf0>, 24),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                        <color object at 0x1d33cc0>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x1d33c90>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                       <pixel object at 0x1d33c60>, 1),
'inactiveselectbackground': ('inactiveselectbackground', 'inactiveSelectBackground',
                             'Foreground', <border object at 0x1d33c30>, '#c3c3c3'),
'insertbackground': ('insertbackground', 'insertBackground', 'Foreground',
                    <border object at 0x1d33bd0>, '#000000'),
'insertborderwidth': ('insertborderwidth', 'insertBorderWidth', 'BorderWidth',
```

```

        <pixel object at 0x1d33ba0>, 0),
'insertofftime': ('insertofftime', 'insertOffTime', 'OffTime', 300, 300),
'insertontime': ('insertontime', 'insertOnTime', 'OnTime', 600, 600),
'insertwidth': ('insertwidth', 'insertWidth', 'InsertWidth', <pixel object at 0x1d33b10>, 2),
'maxundo': ('maxundo', 'maxUndo', 'MaxUndo', 0, 0),
'padx': ('padx', 'padX', 'Pad', <pixel object at 0x1d33ab0>, 1),
'pady': ('pady', 'padY', 'Pad', <pixel object at 0x1d33a80>, 1),
'relief': ('relief', 'relief', 'Relief', <index object at 0x1d33a50>, 'sunken'),
'selectbackground': ('selectbackground', 'selectBackground', 'Foreground',
        <border object at 0x1d33a20>, '#c3c3c3'),
'selectborderwidth': ('selectborderwidth', 'selectBorderWidth', 'BorderWidth',
        <pixel object at 0x1d339c0>, <pixel object at 0x1d339c0>),
'selectforeground': ('selectforeground', 'selectForeground', 'Background',
        <color object at 0x1d33990>, '#000000'),
'setgrid': ('setgrid', 'setGrid', 'SetGrid', 0, 0),
'spacing1': ('spacing1', 'spacing1', 'Spacing', '0', 0),
'spacing2': ('spacing2', 'spacing2', 'Spacing', '0', 0),
'spacing3': ('spacing3', 'spacing3', 'Spacing', '0', 0),
'startline': ('startline', '', '', '', ''),
'state': ('state', 'state', 'State', <index object at 0x1d35640>, 'normal'),
'tabs': ('tabs', 'tabs', 'Tabs', '', ''),
'tabstyle': ('tabstyle', 'tabStyle', 'TabStyle', <index object at 0x1d356a0>, 'tabular'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'undo': ('undo', 'undo', 'Undo', 0, 0),
'width': ('width', 'width', 'Width', 80, 80),
'wrap': ('wrap', 'wrap', 'Wrap', <index object at 0x1d35460>, 'char'),
'xscrollcommand': ('xscrollcommand', 'xScrollCommand', 'ScrollCommand', '', ''),
'yscrollcommand': ('yscrollcommand', 'yScrollCommand', 'ScrollCommand', '', ''),

```

3. se repérer dans un texte

3.1. les index de position

Le texte d'un widget `Text` est une séquence de caractères indexée : les caractères sont numérotés à partir de 0. On peut également numéroter les lignes et les « colonnes ». Les lignes sont numérotées à partir de 1 et les caractères sur une ligne, les colonnes, à partir de 0.

Une position est calculée avant un caractère. Par exemple la colonne 1, colonne 0 est la position en début de texte.

Pour caractériser une position on utilise une chaîne de caractères :

- * "*lgn.cln*" : *lgn* est le numéro de ligne, *cln* celui de la colonne. Par exemple "12.25" désigne la position en ligne 12, et **avant** un éventuel 25^{ème} caractère.
- * "*lgn.end*" : *end* désigne la position juste avant le fin de ligne (retour de ligne ou fin de texte). Par exemple "12.*end*" désigne la fin de la ligne 12.
- * **INSERT** = "*insert*" : "*insert*" désigne la position du curseur d'insertion.
- * **CURRENT** = "*current*" : "*current*" désigne la position la plus proche du curseur de la souris, bouton relâché. Si on déplace la souris bouton enfoncé, la mise à jour ne se fait pas.
- * **END**="end" : position **après** le dernier caractère du texte.
- * **SEL_FIRST**="sel.first" : position **avant** le premier caractère sélectionné. Il y a erreur si aucune sélection n'est faite.
- * **SEL_LAST**="sel.last" : position **après** le dernier caractère sélectionné. Il y a erreur si aucune sélection n'est faite.
- * "*nom_de_marque*" : position d'une marque dans le texte.
- * "*tag.first*" : position avant une région étiquetée(**tag**).

- * "**tag.last**" : position après une région étiquetée(**tag**).
- * "**@x.y**" : **x** et **y** sont des décimaux qui désignent les coordonnées d'un point dans le widget. La position est celle la plus proche du point de coordonnées données.
- * "**objet_inclus**" : Le widget inclus dans le texte peut être un widget ou une image. Dans ces cas, utiliser l'identificateur de l'instance du widget, ou le nom de celui de la **PhotoImage** ou de la **BitmapImage** (soit le nom donné automatiquement par **tkinter**, soit un nom donné par l'utilisateur, en prenant garde que c'est l'image affichée qui est nommée, -même en cas d'affichage multiple d'une même donnée-, et que tous les noms doivent être uniques).
- * "**pos+n char**" : **pos** est l'une des chaînes décrites ci-dessus ; **n** est un décimal : la chaîne décrit alors la position décalée de **n** caractères. Attention cependant, la position ne peut être supérieure à **END**.
- * "**pos-n char**" : la chaîne décrit la position décalée de **n** caractères vers l'avant. Attention cependant, la position ne peut être négative. Exemple : "**12.5+5 char**". L'espace peut être omis et **char** remplacé par **c** : "**12.5+5c**" peut s'avérer plus pratique.
- * "**pos+n lines**", "**pos-n lines**" : même chose avec des lignes. En principe, la position dans la ligne est conservée, sauf si la ligne est trop courte, auquel cas la position est la fin de la ligne.
- * "**linestart**" : position en début de ligne. Exemple : "**current linestart**".
- * "**lineend**" : position en fin de ligne.
- * "**wordstart**" : position avant le premier caractère du mot à la position spécifiée.
Exemple : "**12.5 wordstart**" désigne la position avant le premier caractère du mot repéré en ligne 12, position 5. Le mot est formé de caractères alpha-numériques et du souligné (*underscore*).

3.2. les marqueurs

Un marqueur est un objet invisible inséré dans le texte. Les marqueurs sont positionnés entre les caractères. Elle «flottent» avec le texte et restent toujours dans le même voisinage immédiat. Un marqueur est un mot sans **espace**, **tabulation**, **où point**.

Il existe des marqueurs prédéfinies **INSERT** ou **CURRENT**. On les utilise partout où un marqueur est possible (saut la suppression de marqueur !).

Il existe une propriété des marqueurs appelée la gravité (**gravity**). Elle peut être posée à **LEFT** = "**left**" ou **RIGHT** = "**right**" et signifie qu'une insertion doit garder le marqueur respectivement à gauche ou à droite lors d'une insertion.

Si on supprime un bloc, les marqueurs ne sont pas supprimés et se repositionnent dans le texte restant de manière naturelle.

4. les balises (tags)

4.1. fonctionnalités et aspect de blocs

Le widget **Text** donne la possibilité de régler la famille de caractères, la couleur, la taille dans toute partie du texte. On peut aussi faire réagir le texte, les widgets inclus et les images incluses à une action de la souris ou du clavier. Pour contrôler ces apparences et fonctionnalités, on affecte à chacune par une balise (**tag**). Puis on associe ces balises sur chaque partie du texte concernée. (C'est un peu l'équivalent des balises en HTML). Un **tag** est un mot, sans **espace**, **tabulation**, **point**.

Il existe une constante tag prédéfinie, **SEL** = "**sel**" qui balise la partie du texte actuellement sélectionnée, s'il y en a une ; dans ce cas **tag_ranges(SEL)** retourne un tuple non vide, alors qu'il est vide si le tag **SEL** n'est pas utilisé.

4.2. la pile des tags

Les parties taguées peuvent être imbriquées : par exemple un mot, inclus dans une phrase, inclus dans un paragraphe, tous trois tagués. En cas de conflit entre les tags, c'est le plus profond (ici, celui sur le mot) qui est pris en considération : les tags sont **empilés** par ordre d'apparition.

4.3. les option de tag

clef d'option	type	explication
<code>background</code>	couleur	couleur de fond du texte tagué. Ne pas utiliser <code>bg</code> .
<code>bgstipple</code> , <code>bg</code>	bitmap	nom du bitmap utilisé pour faire un fond pointillé. Les valeurs usuelles sont " <code>gray12</code> ", " <code>gray25</code> ", " <code>gray50</code> ", " <code>gray75</code> "
<code>borderwidth</code>	distance	largeur de la bordure du texte tagué. Ne pas utiliser <code>bd</code> .
<code>fgstipple</code> , <code>fg</code>	bitmap	nom du bitmap utilisé pour dessiner les caractères du texte tagué. Voir <code>bgstipple</code> pour les valeurs.
<code>font</code>	fonte	fonte du texte tagué.
<code>foreground</code>	couleur	couleur d'affichage du texte tagué
<code>justify</code>	chaîne cst	justification du texye tagué. le valeurs sont : <code>LEFT=left</code> , <code>RIGHT="right"</code> , <code>CENTER="center"</code> . <code>LEFT</code> par défaut.
<code>lmargin1</code>	distance	valeur de retrait (alinéa) de la première ligne du bloc tagué.
<code>lmargin2</code>	distance	valeur de retrait des lignes d'un bloc tagué sauf la première.
<code>offset</code>	distance	décalage du texte tagué par rapport à la ligne de base. Nombre positif pour exposant, négatif pour indice.
<code>overstricke</code>	booléen	barre le texte tagué.
<code>relief</code>	chaîne cst	bordure pour le texte tagué.
<code>rmargin</code>	distance	marge droite du bloc tagué.
<code>spacing1</code>	distance	espace au dessus de la première ligne d'un bloc de texte.
<code>spacing2</code>	distance	interligne du bloc tagué
<code>spacing2</code>	distance	espace au dessous d'un bloc tagué.
<code>tabs</code>	tuple de distances	valeurs de tabulation.
<code>underline</code>	booléen	Posé à <code>True</code> , souligne le texte tagué.
<code>wrap</code>	chaîne cst	mode de coupure de ligne pour le bloc tagué : <code>NONE="none"</code> , <code>CHAR="char"</code> , <code>WORD="word"</code> .

5. les méthodes

5.1. utilitaires

* `compare(index1, op, index2)` : retourne la valeur de la relation `index1 op index2`. `op` est la valeur d'opérateur d'une des chaînes suivantes: '`<`', '`<=`', '`=`',

'>=', '>', où '!='. Exemple d'appel : `compare ("12.0", "<", END)` retourne `True` si la douzième ligne est avant la fin de texte.

* `index(index)` : prend une valeur d'`index` entière et la retourne sous la forme "`lgn.cln`".

5.2. méthodes générales d'édition

* `insert(index, chars, *args)` : insère la chaîne `chars` à gauche de l'`index` donné. Un tag peut être fourni comme troisième argument. On peut faire suivre d'une succession de chaînes et tags.

* `delete(index1, index2=None)` : efface les caractères en commençant après la **position** de l'`index1` et en finissant avant la **position** `index2`. Si `index2` est `None` (non précisé), un seul caractère est effacé.

* `edit_modified(arg=None)` : le drapeau booléen *modified flag* est une propriété interne accessible par la présente méthode. Si une modification se produit dans le texte, le drapeau est posé à `True`. La méthode retourne l'état actuel du drapeau en le forçant suivant la valeur booléenne donnée à `arg`.

* `edit_redo()` : rétablit la dernière annulation si l'attribut `undo` est posé à `True`. Il y a erreur si la pile d'annulation est vide. Ne fait rien si l'attribut `undo` est posé à `False`.

* `edit_reset()` : vide la pile d'annulation.

* `edit_separator()` : insère un repère dans la pile d'annulation si l'attribut `undo` est posé à `True`. Ne fait rien sinon.

* `edit_undo()` : annule la dernière «action d'édition» sur le texte si l'attribut `undo` est posé à `True`, ne fait rien sinon. Une action d'édition se définit comme toute commande d'insertion ou d'effacement enregistrée sur la pile d'annulation entre deux repères séparateurs. Il y a erreur si la pile est vide.

* `get(index1, index2=None)` : retourne le texte compris entre les positions `index1` et `index2`. Si `index2` n'est pas donné, un seul caractère est retourné.

* `dump(index1, index2=None, command=None, options)` : retourne le contenu du widget entre `index1` et `index2`

Le type de contenu retourné est filtré en suivant les paramètres à mots clefs optionnels. Si des paramètres de mot clefs `all`, `image`, `mark`, `tag`, `text`, ou `window` sont posés à `True`, alors les items correspondants sont retournés. Le résultat est une liste de triplets de la forme (*clef, valeur, index*). Par défaut, c'est `all` qui est posé à `True`.

Si l'argument '`command`' est donné, la fonction est appelée pour chaque élément de la liste de triplets, les valeurs de chaque triple servant d'argument à la fonction. Dans ce cas la liste n'est pas retournée.

Pour imprimer, on utilise `get()` et `dump()`, à envoyer à une sortie adaptée ; par exemple le module `PSDraw` de `PIL`.

5.3. méthodes pour les marqueurs

* `mark_gravity(markName, direction=None)` : change la gravité du marqueur `markName`. La `direction` peut être `LEFT="left"` ou `RIGHT="right"`. Retourne la valeur courante si `direction = None`.

* `mark_names()` : la méthode retourne tous les noms de marqueurs.

* `mark_set(markName, index)` : pose un marqueur `markName` avant le caractère à la position `index`.

* `mark_unset(*markNames)` : efface tous les marqueurs de la séquence `markNames`.

* `mark_next(index)` : retourne le nom du premier marqueur après `index`.

* `mark_previous(index)` : retourne le nom du premier marqueur avant `index`.

5.4. méthodes pour les éléments fenêtrés inclus

- * `window_cget (index, option)` : c'est l'habituel `cget()`, mais sur un élément fenêtré inclus, trouvé à la position `index`.
- * `window_configure(index, options), window_configure(index, options)` : c'est l'habituel `configure()`, mais sur un élément fenêtré inclus, trouvé à la position `index`.
- * `window_names()` : retourne une séquence de tous les noms d'éléments fenêtrés inclus.
- * `window_create(index, option)` : crée un élément fenêtré à la position `index`, avec les options spécifiées. Il y a deux façons d'inclure un widget dans le widget `Text`, par les options `create` ou `window`.

Les options possibles sont :

clef d'option	explication
<code>align</code>	donne le placement vertical de l'élément fenêtré inclus : <code>CENTER="center", TOP="top", BOTTOM="bottom", BASELINE="baseline"</code> .
<code>create</code>	la valeur est une fonction sans argument qui crée le widget incorporé à la demande. Cette fonction doit retourner un widget enfant du widget <code>Text</code> .
<code>padx, pady</code>	marges supplémentaire pour le widget
<code>stretch</code>	Normalement, cette valeur est posée à <code>False</code> et le widget inclus est affiché en vraie grandeur. Si on la pose à <code>True</code> , et que la ligne a trop de place verticalement, le widget est étiré pour occuper tout l'espace de la ligne ; <code>align</code> est alors inopérant.
<code>window</code>	la valeur est un widget existant enfant de <code>Text</code> .

5.5. méthodes pour les images incluses

les options d'image sont :

clef d'option	type	explication
<code>align</code>	constante chaîne	Les valeurs possibles sont <code>TOP="top", CENTER="center", BOTTOM="bottom", BASELINE="baseline"</code> . <code>BASELINE</code> signifie ue l'image se comporte comme un caractère.
<code>image</code>	image	instance de l'image insérée à la position
<code>name</code>	chaîne	le nom de l'image : soit celui donné par l'utilisateur, soit par défaut, celui donné par <code>tkinter</code> .
<code>padx, pady</code>	distance	ajoute des marges à l'intérieur de le contour du texte.

- * `image_cget(index, clef_option)` : retourne la valeur de `clef_option` pour une image incluse à la position `index`.
- * `image_configure(index, options), image_configure(index, options)` : configure l'image incluse à la position `index`.
- * `image_create(index, options)` : inclut une image à la position `index`, comme un caractère qui aurait la taille de l'image.

* `image_names()` : retourne les noms de toutes les images incluses sous forme d'un tuple.

5.6. méthodes pour les balises

- * `tag_add(tagName, index1, index2=None, *args)` : ajoute un tag , nommé `tagName`, sur le bloc compris entre `index1` et `index2`. Si `index2` est omis, un seul caractère est tagué. On peut faire suivre d'autant de paires semblables.
- * `tag_unbind (tagName, sequence, funcid=None)` : délie le bloc tagué par `tagName`, pour l'événement de séquence `sequence`, et de gestionnaire `funcid`. Si `funcid` n'est pas spécifié, tous les gestionnaires sont déliés de du bloc pour l'événement.
- * `tag_bind (tagName, sequence, func, add=None)` : fait une liaison entre le bloc nommé `tagName`, avec un événement de séquence `sequence`, de gestionnaire `func`. `add` est traditionnel pour les liaisons d'événement : `add=True` ajoute la liaison à celles qui existent déjà ; `False` force le remplacement.
- * `tag_cget (tagName, clef_option)` : retourne la valeur d'attribut du mot clef `clef_option` pour le tag nommé `tagName`.
- * `tag_configure (tagName, options), tag_config(tagName,options)` : configure le tag nommé `tagName`.
- * `tag_delete (*tagNames)` : supprime tous les tags de la liste d'arguments.
- * `tag_lower (tagName, belowThis=None)` : change la priorité du tag, qui se place en dessous de celui nommé `belowThis`. Si le second paramètre est `None`, le tag perd toute priorité.
- * `tag_names(index=None)` : retourne une liste des tags valides pour l'index. Les retourne tous si l'index n'est pas précisé.
- * `tag_nextrange(tagName, index1, index2=None)` : `index2` est supposé supérieur à `index1` (le fin du texte pour `index2=None`). La méthode recherche en commençant à `index1` et jusque `index2` si `tagName` est le nom d'un tag. Si elle ne trouve rien, elle retourne une chaîne vide.
Si elle rencontre une position de début de tag `d`, elle cherche la position `f` de fin correspondante. Et elle retourne `[1, f]`.
- * `tag_prevrange(tagName, index1, index2=None)` : même chose, mais avec `index2` inférieur à `index1` (début du texte si `index2=None`) ; la recherche est fait en remontant dans le texte.
- * `tag_raise(tagName, aboveThis=None)` : change la priorité du tag, qui se place en dessus de celui nommé `aboveThis`. Si le second paramètre est `None`, le tag a une priorité absolue.
- * `tag_ranges(tagName)` : retourne une liste formée des positions de début et de fin de tous les blocs de texte ayant le tag nommé `tagName`. La liste a l'allure suivante : `[d0, f0, d1, f1, ..., dn, fn]`
- * `tag_remove (tagName, index1, index2=None)` : enlève les tags entre `index1` et `index2`. Si `index2` est omis, seul le tag en `index1` est enlevé. Attention, **les tags en tant que tels ne ont pas supprimés**, même si on les enlève tous (`tag_remove(tagName, "1.0", END)`).

5.7. méthode de rendu

- * `dlineinfo(index)` : retourne le tuple `(x,y,width,height,baseline)` donnant la **boite englobante** et la position de la **ligne de base (baseline)** de la partie visible de la ligne contenant le caractère après `index`.
- * `bbox(*args)` : retourne un tuple `(x,y,width,height)` qui donne les bornes de la boite de la partie visible pour chaque caractère dont l'index est dans `args`. Attention au retards à l'affichage (dans ce cas, faire `update_idletasks()`).

5.8. méthodes de recherche

```
* search(pattern, index, stopindex=None,
          forwards=None, backwards=None, exact=None,
          regexp=None, nocase=None, count=None, elide=None):
```

paramètres pour une recherche :

- **pattern** = motif recherché (chaîne) ;
- **index** = position du début de recherche ;
- **stopindex** = position de fin de recherche ;
- **forward** : en avant (booléen) ;
- **backward** : en arrière (booléen) ;
- **exact** : recherche d'une concordance exacte avec le motif (booléen) ;
- **regexp** = expression régulière (booléen) ; le motif est à considérer comme une expression régulière ; seul un sous ensemble des possibilités des **regexp** de **Python** est permise.
Sont reconnus : le point . ; les répétiteurs + * ? ; la suite de caractère [xyz...], le parenthésage (...); l'alternative e1|e2
- **nocase** = ne pas tenir compte de la casse (booléen) ;
- **count** = variable de contrôle de type `IntVar` ; le variable prend la valeur de longueur de la chaîne identifiée (interroger la variable par la méthode `get()`).
- **elide** = (booléen).

retourne l'index du premier caractère de la chaîne identifiée, ou sinon une chaîne vide (la longueur se trouve par `v.get()` où `v` est la variable associée à `count`).

5.9. méthodes pour les ascenseurs

* **scan_mark(x, y)** : sert dans le scrolling rapide par glissement de la souris. Si on presse le bouton de la souris en un point et qu'on la déplace ensuite, puis qu'on relâche, la souris se déplace dans la direction du mouvement en proportion de la distance parcourue.

Pour mettre en œuvre le procédé, associer l'événement «presser de bouton souris» ("`<ButtonPress`") à un gestionnaire qui appelle **scan_mark(x, y)** à la position (x,y) de la souris, et l'événement «déplacer le souris» ("`<Motion`") à un gestionnaire qui appelle **scan_dragto(x, y)**, avec (x, y) comme position actuelle de la souris.

* **scan_dragto(x, y)** : ajuste un déplacement de l'affichage du texte à 10 fois la différence entre x et y et les coordonnées données dans **scan_mark**.

* **see(index)** : scrolle pour rendre visible le caractère à la position index. Remplace **yview_pickplace()**, obsolète.

méthodes héritées :

Les méthodes héritées de **XView** et **YView** sont partagés avec **Canvas** et **Listbox**. Ce sont elles qui sont explicitées maintenant. Elles permettent de rechercher et de changer la position dans une fenêtre de widget. Elle utilisent des constantes : **MOVETO** = "`moveto`", **SCROLL** = "`scroll`", **UNITS** = "`units`", **PAGES** = "`pages`". Voir le **chapitre tk19** sur le même sujet.

* **xview(*arg)** : recherche et change la position horizontale de la vue. Voir les méthodes qui suivent.

* **xview_moveto(fraction)** : c'est la même chose que **xview(MOVETO, fraction)**. Cette méthode est destinée au widget de **scroll** horizontal. Elle déplace la vue sur le texte du widget à la position définie par **fraction**. **fraction = 0.0** correspond à la gauche, et **fraction = 1.0** est l'extrême droite.

* **xview_scroll(n, units)** : même chose que **xview(SCROLL, n, what)**. Déplace la vue sur le texte vers la gauche ou la droite. **n**, entier relatif donne l'ampleur du déplacement (positif : à droite et négatif : à gauche) et **units** donne le genre de

déplacement qui peut être **UNITS** (caractères) ou **PAGES** (pages).

* `yview(*arg)`, `yview_moveto(fraction)`, `yview_scroll (n, units)` : semblables aux précédentes, mais dans le sens vertical.

6. quelques exemples

6.1. méthodes de sélection

Il n'y a rien de spécifique dans le widget `Text` en ce qui concerne la sélection, autre que le tag `SEL`, et l'initialisation de `SEL_FIRST` et `SEL_LAST` lorsque la sélection existe. Une utilisation abusive de ces constantes provoque une erreur. Voici quelques procédés utiles :

effacer la sélection courante :

```
def effacer_sel (widget_Text) :           # il s'agit d'un widget Text
    widget_Text.tag_remove (SEL, "1.0", END)
```

sélectionner

```
def creer_sel (widget_Text, index1, index2) :
    widget_Text.tag_remove (SEL, "1.0", index1)
    widget_Text.tag_add (SEL, index1, index2)
    widget_Text.tag_remove (SEL, index2, END)
```

sélection présente

```
def existe_sel (widget_Text) :
    return bool(widget_Text.tag_range(SEL))
```

6.2. un exemple simple

Le script comporte deux widgets `Text`. Le widget source permet d'éditer du texte et de sélectionner un bloc. Les deux commandes permettent de copier le texte sélectionné ou tout le texte dans le widget cible, à la position «actuelle» du curseur.

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
from tkinter import *
ftTexte = "Helvetica -20"
ftBouton = "Courier -50"

# texte explicatif
aide = """Le script fonctionne de la façon suivante :\n
* une partie sélectionnée du texte de gauche ou le texte en entier\
est copié(e) dans le texte cible à droite, avec les boutons \u2907\
et \u2b46. La texte inséré est sélectionnée.
* pour cela, il faut mettre le curseur au point d'insertion, \
dans la cible, et faire Alt-I. Un fond coloré (en une ou deux \
parties) visualise la position.
* le Alt-I a un effet de bascule. On peut donc supprimer son \
effet en le cliquant de nouveau.
"""
```

```

# fenêtre principale
root = Tk()
root.title("copie gauche droite")

# les widget Text
texteCible = Text(root, font=ftTexte, width=60, height=15, wrap="word",
                  selectbackground="#a0a0ff",
                  inactiveselectbackground="#d0d0ff")
texteSource = Text(root, font=ftTexte, width=40,height=15, wrap="word",
                  selectbackground="#a0a0ff",
                  inactiveselectbackground="#d0d0ff")

texteCible.event_add("<<Ctrl-I>>", "<Alt-I>", "<Alt-i>")
texteCible.tag_config("tagAvantInsert", background="#80ffff")
texteCible.tag_config("tagApresInsert", background="#ff80ff")
# où insérer ?
class Insertion() :
    __actuel = ""
    def get() :
        return Insertion.__actuel
    def set (v) :
        Insertion.__actuel = v
    def maj () :
        texteCible.tag_add("tagAvantInsert", "1.0", INSERT)
        Insertion.__actuel = texteCible.index(INSERT)

# préparer ou terminer l'insertion
def bgInserrer(event=None) :
    if Insertion.get() : # enlever
        texteCible.tag_remove ("tagAvantInsert", "1.0", END)
        texteCible.tag_remove ("tagApresInsert", "1.0", END)
        texteCible.tag_add (SEL, Insertion.get(),
                            texteCible.index(INSERT))
        Insertion.set("")
    else : # poser
        texteCible.tag_remove(SEL, "1.0", END)
        Insertion.set(texteCible.index(INSERT))
        texteCible.tag_add("tagAvantInsert", "1.0", INSERT)
        texteCible.tag_add("tagApresInsert", INSERT, END)

# le Alt-I
texteCible.bind ("<<Ctrl-I>>", bgInserrer)

# les boutons
def boutonSel():
    if texteSource.tag_ranges(SEL) and Insertion.get():
        texteCible.insert(Insertion.get(),
                          texteSource.get(SEL_FIRST, SEL_LAST))

```

```

        bgInserrer()

def boutonAll():
    if Insertion.get():
        texteCible.insert(Insertion.get(), texteSource.get("1.0",END))
        bgInserrer()

def aider():
    texteSource.delete("1.0", END)
    texteSource.insert("1.0", aide)

frBoutons = Frame(root, background="#a0a0a0")
btGDSel = Button(frBoutons, font=ftBouton, text="\u2907",
command=boutonSel)
btGDAll = Button(frBoutons, font=ftBouton, text="\u2b46",
command=boutonAll)
btAide = Button(frBoutons, font=ftBouton, text="?", command=aider)
btQuit = Button(frBoutons, padx=20, font=ftBouton, text="\u03a9",
command=quit)

# le scrolling vertical
scrollbar = Scrollbar(root)
scrollbar.config(command=texteCible.yview)
texteCible.config(yscrollcommand=scrollbar.set)

# placement
scrollbar.pack(side=RIGHT, fill="both")
texteCible.pack(side=RIGHT)
texteSource.pack(side=LEFT)
frBoutons.config(width=btGDSel.winfo_reqwidth())
frBoutons.pack(side=LEFT, fill=Y)
btGDSel.pack(pady=10, padx=5)
btGDAll.pack(pady=10, padx=5)
btQuit.pack(side=BOTTOM, pady=10)
btAide.pack(side=BOTTOM, pady=20)

aider()

# rester dans la boucle principale
root.mainloop()

# fichier tk20ex00.py

```

Résultat :

Le script fonctionne de la façon suivante :

- * une partie sélectionnée du texte de gauche ou le texte en entier est copié(e) dans le texte cible à droite, avec les boutons ⇨ et ⇩. La texte inséré est sélectionnée.]
- * pour cela, il faut mettre le curseur au point d'insertion, dans la cible, et faire Alt-I. Un fond coloré (en une ou deux parties) visualise la position.
- * le Alt-I a un effet de bascule. On peut donc supprimer son effet en le cliquant de nouveau.

Le widget Text permet de créer un éditeur dans un programme. On peut utiliser plusieurs fontes et plusieurs couleur. On peut incorporer une image qui est alors considérée comme un caractère unique. On peut aussi introduire des marques invisibles entre les caractères.

Par défaut, l'éditeur peut saisir du texte ; il connaît le passage de ligne sur le clavier principal (pas sur le pavé numérique), la tabulation et les touches fléchées. Il fonctionne par insertion, pas par emplacement. Il dispose des propriétés usuelles du presse papier. Il passe à la ligne sur le caractère excédentaire.

tk21 : Canvas

Un widget **Canvas** (un canevas) est un élément fenêtré destiné à recevoir des éléments graphiques «dessinés». On peut dessiner des traits, des surfaces, des textes, des images.

Chaque élément dessiné est supporté par un calque transparent ; **les calques sont superposés dans l'ordre où ils sont dessinés**, et les parties opacifiées cachent donc tout ce qui est en-dessous.

L'ordre des calques peut être modifié, et chaque calque peut être manipulé indépendamment de tous les autres. Mais on peut également les regrouper par familles (voit **tags**). Pour manipuler les calques, les images qu'ils supportent sont identifiés. Le **Canvas** ressemble à un logiciel d'édition graphique vectoriel où les transformations sont recalculées à partir des directives de création.

1. le constructeur

`widget = Canvas (master, options) .`

Les éléments qui peuvent être dessinés sont :

- * **arc** : ellipse, segment ou partie angulaire d'une ellipse ; l'ellipse est repérée par 4 valeurs : coin supérieur gauche (**x1, y1**) et coin inférieur droit (**x2, y2**) du rectangle enveloppant l'ellipse.
- * **bitmap** : le bitmap est repéré par son coin supérieur droit.
- * **image** : l'image est repérée par son coin supérieur droit.
- * **ligne** : il s'agit d'une ligne polygonale, repérée par la succession de ses sommets.
- * **oval** : il s'agit d'une ellipse d'axes parallèles aux bords du canevas, avec le cercle comme cas particulier. Voir **arc**.
- * **polygon** : c'est une ligne polygonale fermée.
- * **rectangle** : un rectangle de côtés parallèles aux bords du canevas, avec le carré comme cas particulier.
- * **text** : on peut dessiner du texte sur un canevas, avec le choix de la fonte, de la taille etc.
- * **window** : on peut créer une «réserve» rectangulaire, où on peut placer un widget, par exemple un cadre (**Frame**) et toute construction interne dans le cadre que l'on veut.

2. les attributs

2.1. liste des attributs

`background, bd, bg, borderwidth, closeenough, confine, cursor, height, highlightbackground, highlightcolor, highlightthickness, insertbackground, insertborderwidth, insertofftime, insertontime, insertwidth, offset, relief, scrollregion, selectbackground, selectborderwidth, selectforeground, state, takefocus, width, xscrollcommand, xscrollincrement, yscrollcommand, yscrollincrement`

2.2. les attributs spécifiques

- * **closeenough** : fixe à quelle distance le curseur de la souris doit être d'un item pour être considéré comme à l'intérieur.
- * **confine** : booléen. Posé à **True**, valeur par défaut, le scrolling est confiné à la région définie dans **scrollregion**.
- * **scrollregion** : la valeur est un tuple de la forme (**x0, y0, x1, y1**) qui définit un rectangle par sa diagonale principale. C'est la région réduite du canevas qui peut être scrollée. Si le rectangle est le contour du canevas (ou moins), les ascenseurs sont inopérants. Si l'attribut n'est pas défini (par défaut), le confinement ne se fait pas.
- * **xscrollincrement** : posé à 0, le scrolling est assez continu ; si on donne une valeur entière positive, le scrolling se positionne uniquement aux multiples de l'attribut. En

particulier, une action sur les flèches de la barre de scroll horizontal fait sauter l'affichage, avec un pas égal à l'attribut.

* **yscrollincrement** : même chose, mais vertical.

* **yscrollcommand** : on peut ajouter un ascenseur vertical au widget pour manipuler une saisie plus large que celle autorisée par la fenêtre de saisie. Voir le **chapitre tk12**.

* **xscrollcommand** : on peut ajouter un ascenseur horizontal au widget pour manipuler une saisie plus large que celle autorisée par la fenêtre de saisie.

```
'background': ('background', 'background', 'Background', '#d9d9d9', 'white'),
'bd': ('bd', 'borderWidth'),
'bg': ('bg', 'background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', '0', '0'),
'closeenough': ('closeenough', 'closeEnough', 'CloseEnough', '1', '1.0'),
'confine': ('confine', 'confine', 'Confine', '1', '1'),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'height': ('height', 'height', 'Height', '7c', '600'),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                        '#d9d9d9', '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor', '#000000',
                  '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness', '1',
                      '1'),
'insertbackground': ('insertbackground', 'insertBackground', 'Foreground', '#000000',
                    '#000000'),
'insertborderwidth': ('insertborderwidth', 'insertBorderWidth', 'BorderWidth', '0', '0'),
'insertofftime': ('insertofftime', 'insertOffTime', 'OffTime', '300', '300'),
'insertontime': ('insertontime', 'insertOnTime', 'OnTime', '600', '600'),
'insertwidth': ('insertwidth', 'insertWidth', 'InsertWidth', '2', '2'),
'offset': ('offset', 'offset', 'Offset', '0,0', '0,0'),
'relief': ('relief', 'relief', 'Relief', 'flat', 'flat'),
'scrollregion': ('scrollregion', 'scrollRegion', 'ScrollRegion', '', ''),
'selectbackground': ('selectbackground', 'selectBackground', 'Foreground', '#c3c3c3',
                    '#c3c3c3'),
'selectborderwidth': ('selectborderwidth', 'selectBorderWidth', 'BorderWidth', '1', '1'),
'selectforeground': ('selectforeground', 'selectForeground', 'Background', '#000000',
                    '#000000'),
'state': ('state', 'state', 'State', 'normal', 'normal'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'width': ('width', 'width', 'Width', '10c', '800'),
'xscrollcommand': ('xscrollcommand', 'xScrollCommand', 'ScrollCommand', '', ''),
'xscrollincrement': ('xscrollincrement', 'xScrollIncrement', 'ScrollIncrement', '0', '0'),
'yscrollcommand': ('yscrollcommand', 'yScrollCommand', 'ScrollCommand', '', ''),
'yscrollincrement': ('yscrollincrement', 'yScrollIncrement', 'ScrollIncrement', '0', '0')
```

3. fonctionnement du widget

3.1. dimensions

* les dimensions du widget sont celles de la fenêtre d'affichage, pas la surface où il est possible de créer les calques pour dessiner. Ceux-ci sont supposés indéfinis.

* la dimension d'affichage de la vue, c'est à dire ce que l'on peut amener à être vu dans la fenêtre d'affichage, est fixé par l'attribut **scrollregion**. Les ascenseurs sont un passage quasi obligé pour afficher ce qui déborde de l'élément fenêtré **Canvas** (on pourrait imaginer un déplacement de la vue par les touches du clavier, ou par manipulation de la molette de la souris...). Cet attribut est actif si **confine** est posé à **True**, ce qui est le cas par défaut. Si l'attribut **scrollregion** n'est pas défini (chaîne vide), on ne peut se servir que des flèches de la barre de **scroll**.

3.2. les coordonnées

Le widget possède par défaut un système de coordonnées : axe des abscisses croissantes de gauche à droite, des ordonnées croissantes de haut en bas. L'origine est le point en haut et à gauche du widget. Ce système est fixe.

Mais, les calques qui portent les graphismes dessinés peuvent évoluer lors du scrolling. Il existe donc un second système de coordonnées qui au départ coïncide avec le premier, mais lié à la vue et qui suit ses déplacements. C'est dans ce système que s'effectue le placement des items graphiques.

exemple de canevas avec ascenseurs :

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
#
from tkinter import *
maFonte = "Helvetica -25"

# fonction pour quitter
def quitter():
    racine.quit()

def commande() :
    print (leCanevas.xview())
    print (leCanevas.yview(), "\n")

# la fenêtre de l'application
racine = Tk()
racine.protocol("WM_DELETE_WINDOW", quitter)
racine.geometry("+200+250")
# cadres et boutons
leCadreCanevas = Frame (racine, background="grey", width=800, height=600)
leCadreCanevas.pack()
leCadreBoutons = Frame (racine)
leCadreBoutons.pack(side=BOTTOM, pady=5)
btCommande = Button(leCadreBoutons, font=maFonte, text="Commande",
command=commande)
btQuitter = Button(leCadreBoutons, font=maFonte, text="Quitter",
command=quitter)
btQuitter.pack(side=LEFT, padx=10)
btCommande.pack(side=LEFT, padx=10)

# le canevas et ses ascenseurs
leCanevas = Canvas(leCadreCanevas, width=800, height=600, bg="white")
leCanevas.grid(row=0, column=0)
ascenseurVertical = Scrollbar(leCadreCanevas, orient=VERTICAL, width=20,
command=leCanevas.yview)
ascenseurHorizontal = Scrollbar(leCadreCanevas, orient=HORIZONTAL,
width=20, command=leCanevas.xview)

leCanevas["yscrollcommand"] = ascenseurVertical.set
leCanevas["xscrollcommand"] = ascenseurHorizontal.set
```

```

leCanevas["scrollregion"] = (-300,-200, 900, 700) # 1200x900
deltaX = 40
deltaY = 30
leCanevas["xscrollincrement"]=deltaX
leCanevas["yscrollincrement"]=deltaY
ascenseurVertical.grid(row=0, column=1, sticky="ns")
ascenseurHorizontal.grid(row=1, column=0, sticky="ew")

# la molette de la souris
leCanevas.event_add("<<molette>>", "<MouseWheel>", "<Button-4>",
"<Button-5>", "<Shift-MouseWheel>", "<Shift-Button-4>", "<Shift-Button-5>")

def gestionMolette (event):
    compte = 0
    if event.num == 5 or event.delta == -120:
        compte = 1
    if event.num == 4 or event.delta == 120:
        compte = -1
    if event.state & 1 : # shift
        leCanevas.xview("scroll", compte, "units") # unit = deltaX
    else :
        leCanevas.yview ("scroll", compte, "units") # unit = deltaY

leCanevas.bind("<<molette>>", gestionMolette)

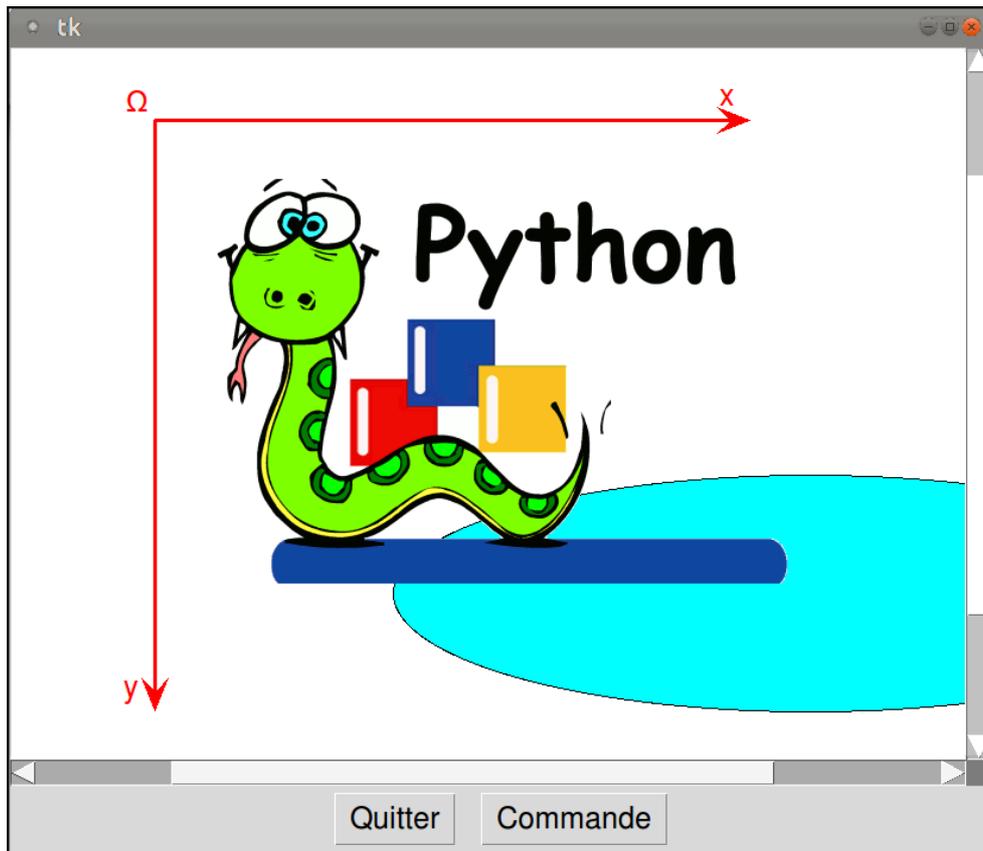
# ellipse
idEllipse = leCanevas.create_oval(200,300, 900,500, fill="cyan")
leCanevas.addtag_withtag("monGroupe", idEllipse)
# axes
idAxeHorizontal = leCanevas.create_line (0, 0, 500, 0,
        fill="red", width=3, arrow=LAST, arrowshape= (16 , 30 , 10))
leCanevas.addtag_withtag("monGroupe", idAxeHorizontal)
idAxeVertical = leCanevas.create_line (0, 0, 0,500,
        fill="red", width=3, arrow=LAST, arrowshape= (16 , 30 , 10))
# texte
idTexte1 = leCanevas.create_text(-15, -15, fill="red", state=DISABLED,
        font=maFonte, text="\u03a9" )
idTexte2 = leCanevas.create_text(480, -20, fill="red", state=DISABLED,
        font=maFonte, text="x" )
idTexte3 = leCanevas.create_text(-20, 480, fill="red", state=DISABLED,
        font=maFonte, text="y" )
# image
pythonImage = PhotoImage (file="./img/pythongraphique.gif")
idImage = leCanevas.create_image(50, 50, image=pythonImage, anchor="nw")

racine.mainloop()

# fichier t121lex00test.py

```

résultat :



3.3. identification et taguage des items

* Lors de la création d'un élément graphique, la méthode de définition de l'instance (item) retourne un numéro (entier),. **Ce numéro est unique est suffit à identifier l'item et on ne peut le changer**. On peut utiliser indifféremment cet identificateur dans les méthodes comme entier `n` ou comme la chaîne décimale `str(n)`.

* Les items doivent pouvoir être regroupés : par exemple pour être soumis à une même transformation géométrique. Aussi **tkinter** permet de caractériser des groupes par un "**tag**" de groupement. Ce **tag** n'a rien à voir avec le tag du widget `Text`, *la balise*. Il se rapproche davantage de ce qu'en HTML on appelle la `classe`. Le mot est assurément mal choisi, mais l'avertissement doit suffire pour pouvoir utiliser l'appellation en ayant conscience de sa polysémie.

Un **tag** est une chaîne de caractères sans espace intérieur. On recommande cependant d'user des modes habituels d'identification, d'éviter les accents par exemple, ou d'utiliser des mots faits uniquement de chiffres.

On peut marquer d'un même **tag** plusieurs items : il appartiennent alors au même groupe de **tag** ; mais rien n'interdit qu'un item soit multitagué. La seule limitation pour un **tag** est d'être lié à au moins un item.

* On peut avoir besoin de désigner tous les items ; la constante `ALL="all"` permet cette désignation.

* De même, on peut désigner l'item «sous» la souris par la constante `CURRENT="current"`.

Une disposition pratique : beaucoup de méthodes confondent **tag** et **id**, c'est-à-dire admettent les `id` comme des tags ordinaires. Dans ce cas, les paramètres impliqués seront appelés `tagOrId` comme le fait la documentation officielle. Si on est dans le cas où on a utilisé un tag désignant plusieurs items, c'est sauf stipulation contraire, celui de plus bas niveau qui est pris.

4. méthodes générales du widget Canvas

On réserve la création des items graphiques à la section suivante **create**, où chacune fera l'objet d'une étude particulière.

4.1. méthodes générales

- * **delete (tagOrId)** : supprime tous les items désignés par **tagOrId**. Il n'y a pas d'erreur s'il n'y en a aucun.
- * **itemcget(tagOrId, option)** : retourne la valeur d'option correspondant à la chaîne **option** pour l'item défini par **tagOrId**.
- * **itemconfigure (tagOrId, options)** : c'est la méthode **configure** des widgets, adaptée aux items. Les valeurs des mots clefs sont définis dans la section **create**.
- * **itemconfig tagOrId, options)** : même chose que le précédent.
- * **type (tagOrId)** : retourne le type de l'item désigné par **tagOrId**.
- * **postscript(options)** : imprime le contenu du canevas dans un "fichier" postscript.

Les options sont les suivantes :

colormode	"color" pour coloré, "grey" pour niveaux de gris et "mono" pour monochrome
file	crée un fichier si file est renseigné. Sinon, une chaîne de caractères
height	hauteur de canevas à écrire ; si non renseigné, c'est toute la hauteur
witdh	largeur de canevas à écrire ; si non renseigné, c'est toute la largeur
rotate	si True , orientation portrait ; si False , paysage
x	abscisse du bord droit du canevas à imprimer
y	valeur du bord haut du canevas à imprimer

il existe d'autres items, non documentés : **colormap**, **fontmap**, **pageanchor**, **pageheight**, **pagewidth**, **pagex**, **pagey**,

4.2. méthodes de manipulation de tags

- * **addtag_above(newtag, tagOrId)** : ajoute le tag **newtag** à l'item juste au-dessus de celui (au sens des calques empilés) marqué par **tagOrId**.
- * **addtag_all(newtag)** : ajoute le tag **newtag** à tous les items existants.
- * **addtag_below(newtag, tagOrId)** : comme **addtag_above()**, mais au-dessous.
- * **addtag_closest(newtag, x, y, halo=None, start=None)** : recherche la bordure d'un item graphique qui est la plus proche du point de coordonnées calques (**x, y**) ; puis lui applique le tag **newtag**. S'il y a ambiguïté, c'est le plus haut dans l'empilement qui est pris. Le paramètre **halo** est un entier : il permet de "grossir" le point d'identification ; **halo=5** recherche jusqu'à 5 pixels de (**x,y**). Si **start** est un identificateur d'item, la tag est appliqué à tous les items d'identificateur supérieur.
- * **addtag_enclosed(newtag, x1, y1, x2, y2)** : applique le tag **newtag** à tous les items inclus dans le rectangle (**x1, y1, x2, y2**).
- * **addtag_overlapping(newtag, x1, y1, x2, y2)** : même chose que **addtag_enclosed()**, mais en prenant tous les items qui ont au moins un point commun avec le rectangle.
- * **addtag_withtag(newtag, tagOrId)** : affecte le tag **newtag** à tout item qui a le tag ou l'identificateur **tagOrId**.

- * `dtag (tagOrId, dTag)` : enlève le tag `dTag` aux items qui l'auraient dans ceux désignés par `tagOrId`.
- * `gettags (tagOrId)` : retourne tous un tuple de tous les tags associés à l'item désigné par `tagOrId` (s'il y en a plusieurs, prend le plus bas).

4.3. méthodes relatives aux dimensions et aux transformations

- * `bbox(*args)` : retourne un tuple (`x1`, `y1`, `x2`, `y2`) définissant un rectangle qui englobe tous les items de tag appartenant à la suite donnée en paramètre. Pour obtenir la boîte qui inclut tous les items, par exemple pour régler les ascenseurs, faire `bbox (ALL)`.
- * `canvasx(screenx, gridspacing=None), canvasy(screeny, gridspacing=None)` : (`screenx`, `screeny`) sont les coordonnées d'un point `M` dans le repère fixe lié à la fenêtre. Les deux méthodes renvoient les coordonnées `x` ou `y` de ce point `M` dans le repère lié aux calques. Si `gridspacing` (entier positif) est donné, le résultat est un multiple de cette valeur.
- * `coords(tagOrId)` : retourne les coordonnées dans le repère lié au calques de l'item désigné par `tagOrId` (s'il y a plusieurs items, prend le plus bas). Les coordonnées sont rendues dans un tuple de flottants, avec 2 ou 4 éléments suivant les items (Voir la section `create`).
- * `coords (tagOrId, x0, y0, x1, y1...)` : prend l'item le plus bas défini par `tagOrId` et lui impose le système de coordonnées qui suit, selon le mode utilisé pour son `create`.

```

    idImage = leCanevas.create_image(50, 50, image=pythonImage,
    anchor="nw")
    leCanevas.coords(idImage, 200, 200)

```

- * `tag_lower (tagOrId, id)` : descend les items définis par `tagOrId` en-dessous de celui défini par `id`.
- * `tag_raise (tagOrId, id)` : comme `tag_lower` mais au-dessus.
- * `move (tagOrId, deltaX, deltaY)` : déplace en bloc les items définis par `tagOrId` en ajoutant `deltaX` et `deltaY` à leurs coordonnées. Les ascenseurs ne sont pas synchronisés.
- * `scale (tagOrId, offsetX, offsetY, ratioX, ratioY)` : applique à tous les items désignés par `tagOrId` la transformation suivantes : Tout point `M` de coordonnées (`x,y`) est remplacé par `M'` (`offsetX + ratioX*(x - OffsetX), offsetY + ratioY*(Y - OffsetY)`).

4.4. méthodes relatives aux événements

- * `tag_unbind(tagOrId, sequence, funcid=None)` : délie l'événement défini par `sequence`, le gestionnaire `funcid` des items ayant le tag ou l'identificateur `tagOrId`.
- * `tag_bind(tagOrId, sequence=None, func=None, add=None)` : lie (`bind`) tous les items possédant le tag `tagOrId` pour l'événement défini par `sequence` et le gestionnaire `func`. Voir le **chapitre 7** sur les événements.

4.5. méthodes de texte

- * `dchars (tagOrId, first=0, last=first)` : efface des caractères aux items textuels de `tagOrId`. Commence au caractère d'ordre `first`, et inclut le caractère `last`. Le second argument peut être `END="end"`.
- * `focus(tagOrId=None)` : donne le focus au premier item focusable désigné par `tagOrId`. Avec l'argument `None`, retourne l'item qui a le focus ou une chaîne vide.
- * `icursor (tagOrId, index)` : si l'item désigné a le focus, positionne le curseur d'insertion à la

position index. `END="end"` est admis.

- * `index (tagOrId, specificateur)` : retourne la position du curseur pour l'item spécifié par `tagOrId` (le plus bas s'il y en a plusieurs).
Le spécificateur peut être : `INSERT`, `END`, `SEL_FIRST`, `SEL_LAST`, "`@x, y`"
Voir le widget `Text` pour la signification.
- * `insert (tagOrId, index, texte)` : insertion d'une chaîne définie par `texte`, à la position définie. On peut utiliser les spécificateurs `INSERT`, `END`, `SEL_FIRST`, `SEL_LAST`.
- * `select_adjust (tagOrId, index)` : ajuste la fin de la sélection au plus près de l'index pour l'item `tagOrId`.
- * `select_clear ()` : enlève la sélection si elle est sur le widget `Canvas`.
- * `select_from(tagOrId, index)` : pose un début de sélection.
- * `select_item()` : retourne l'item qui a le focus.
- * `select_to(tagOrId, index)` : pose un fin de sélection.

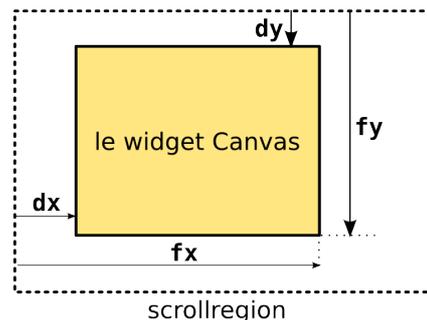
4.5. méthodes de recherche d'item

- * `find_above (tagOrId)` : retourne l'id de l'objet au-dessus de l'item le plus haut désigné par `tagOrId` s'il existe. Retourne un singleton ou un tuple vide.
- * `find_all ()` : retourne un tuple des id de tous les items.
- * `find_below(tagOrId)` : semblable à `find_above()`, mais au-dessous.
- * `find_closest(x, y, halo=None, start=None)` : retourne l'id de l'item le plus proche du point de coordonnées (x, y) dans un singleton. Pour les paramètres, voir `addtag_closest()`.
- * `find_enclosed(x1, y1, x2, y2)` : retourne un tuple des id des items contenus dans le rectangle défini par `x1, y1, x2, y2`.
- * `find_overlapping x1, y1, x2, y2)` : même chose avec les items qui ont au moins un point commun avec le rectangle.
- * `find_withtag (tagOrId)` ; retourne un tuple des items qui partagent le tag `tagOrId`.

4.6. méthodes de scroll

- * `scan_mark(x, y)` : pour cette méthode et la suivante on renvoie vers le widget `Entry`.
- * `scan_dragto(x, y, gain=10)` : id.
- * `xview()`, `yview()` : les méthodes `xview()` et `yview()` sont à usages multiples.

- utilisées sans argument, elle retournent un tuple de flottants. Pour `xview()`, le premier nombre donne le rapport de la partie gauche cachée à la largeur totale, et le second, prend la somme de la largeur cachée à gauche et la largeur de la partie visible et divise par a largeur totale : c'est exactement l'argument de `set()` pour la barre de scroll horizontale. C'est la même chose pour `yview()`, mais dans le sens vertical. Ces méthodes fournissent aux barres de scroll les renseignements pour leur réglage.



- les arguments peuvent être deux ; "`moveto`", *nombre flottant*. Dans ce cas, la vue est modifiée et décalée en proportion de l'arguent flottant (valeur qui est donc entre -1 et +1. C'est ce que fait la barre de scroll quand on déplace le curseur avec la souris.
- les arguments peuvent être trois : "`scroll`", *compte*, "`units`". Le paramètre *compte* peut être un entier +1 ou -1. Dans ce cas, la vue est modifiée par déplacement d'un nombre

de pixels défini dans `xscrollincrement` ou `yscrollincrement`. C'est ce que fait la barre de scroll quand on clique les flèches ou la gouttière. Si `compte` est un entier de valeur absolue supérieure 1, le déplacement est multiplié par cette valeur.

Voir l'exemple donné en section 3.

5. les méthodes create.

5.1. création à partir d'un Bitmap ou d'une Image

les méthodes :

`id = widget_canvas.create_bitmap (x, y, options)`

`id = widget_canvas.create_image (x, y, options)`

`x` et `y` sont les coordonnées du centre de l'image, sauf option contraire. `id` est l'identificateur qui sera utilisé pour manipuler l'item.

les options :

<code>activeimage</code> ou <code>activebitmap</code>	image ou bitmap	valeur de remplacement pour le survol de la souris.
<code>anchor</code>	ancree	par défaut, <code>CENTER="center"</code> . Sinon les ancrées usuelles.
<code>disabledimage</code> <code>disabledbitmap</code>	image ou bitmap	valeur de remplacement pour <code>DISABLED</code>
<code>image</code> ou <code>bitmap</code>	image ou bitmap	valeur d' image (<code>PhotoImage</code>) ou bitmap
<code>state</code>	cst chaîne	<code>NORMAL="normal"</code> , <code>DISABLED="disabled"</code> , <code>HIDDEN="hidden"</code>
<code>tags</code>	chaîne ou tuple de chaînes	les tags associés à l'item
<code>foreground</code> <code>background</code> <code>activebackground</code> <code>activeforeground</code> <code>disabledbackground</code> <code>disabledforeground</code>	couleur	uniquement pour bitmap .

pour une Image :

```
'activeimage': ('activeimage', '', '', '', ''),
'anchor': ('anchor', '', '', 'center', 'nw'),
'disabledimage': ('disabledimage', '', '', '', ''),
'image': ('image', '', '', '', 'pyimage1'),
'state': ('state', '', '', '', ''),
'tags': ('tags', '', '', '', ''),
```

5.2. création de ligne et polygone

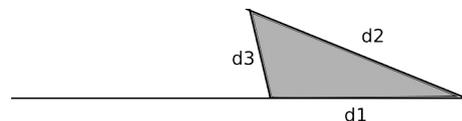
les méthodes :

```
id = widget_canvas.create_line (x0, y0, x1, y1, ..., xn, yn, options)
```

```
id = widget_canvas.create_polygon (x0, y, x1, y1,., xn, yn, options)
```

les options pour create_line :

- * **dash** : par défaut on a une ligne droite continue ; l'attribut **dash** permet de faire un pointillé ou un motif plus complexe. La valeur est un tuple d'entiers qui décrit un motif : les valeurs d'index impair donnent les pixels colorés, ceux d'index pair les pixels transparents. Par exemple (5,5,20,5) donne le motif point trait. Si on n'a qu'une valeur, les pixels colorés et les pixels transparents sont de même longueur.
- * **dashoffset** : la valeur est un entier **n** ; pour tracer en pointillé, on commence au **n^{ème}** pixel du motif.
- * **fill** : couleur du trait
- * **stipple** : permet de dessiner avec un bitmap.
- * **width** : épaisseur du trait
- * **arrow** : permet de doter la ligne d'une flèche ; les valeurs sont **FIRST="first"**, **LAST="last"**, **BOTH="both"** qui dotent la ligne en son début, sa fin ou les deux.
- * **arrowshape** : prend un tuple d'entiers (d1, d2, d3) qui définissent la flèche.



- * **capstyle** : forme de fin de ligne. Les valeurs sont **BUTT="butt"**, **PROJECTING="projecting"**, **ROUND="round"**
- * **joinstyle** : forme du raccord entre segments. Les valeurs sont **ROUND="round"**, **BEVEL="bevel"** et **MITTER="mitter"**.



- * **offset** : si on utilise un bitmap pour tracer une ligne, permet d'assurer la continuité des raccords entre les répétitions du bitmap. La valeur peut être :
 - une chaîne contenant deux valeurs séparées par une virgule "**x,y**" ; **x** et **y** désignent un décalage, en pixels pour passer d'un bitmap au suivant.
 - "**#x,y**" : décale le bitmap de motif par rapport à la fenêtre du canevas
 - une ancre : **N="n"**, **S="s"**, **E="e"**, **W="w"**, **NE="ne"**, **SE="se"**, **SW="sw"**, **NW="nw"**, **CENTER="center"** ; ancre le bitmap. Par exemple **E="e"** fait coïncider le centre du bitmap et le milieu du côté gauche du rectangle enveloppant la ligne. **NE="ne"** fait coïncider les coins haut/gauche.
- * **smooth** : c'est un booléen. Avec **smooth** à **True**, la ligne est dessinée comme une 2-spline (spline parabolique). Sinon, la ligne est calculée comme ligne droite.
- * **splinesteps** : donne le pas de calcul si **smooth** est posé à **True**.

```
'activedash': ('activedash', '', '', '', ''),
'activefill': ('activefill', '', '', '', ''),
'activestipple': ('activestipple', '', '', '', ''),
'activewidth': ('activewidth', '', '', '0.0', '0.0'),
'arrow': ('arrow', '', '', 'none', 'last')
```

```

'arrowshape': ('arrowshape', '', '', ('8', '10', '3'), ('16', '30', '10')),
'capstyle': ('capstyle', '', '', 'butt', 'butt'),
'dash': ('dash', '', '', '', ''),
'dashoffset': ('dashoffset', '', '', '0', '0'),
'disableddash': ('disableddash', '', '', '', ''),
'disabledfill': ('disabledfill', '', '', '', ''),
'disabledstipple': ('disabledstipple', '', '', '', ''),
'disablewidth': ('disablewidth', '', '', '0.0', '0.0'),
'fill': ('fill', '', '', 'black', 'red'),
'joinstyle': ('joinstyle', '', '', 'round', 'round'),
'offset': ('offset', '', '', '0,0', '0,0'),
'smooth': ('smooth', '', '', '0', '0'),
'splinsteps': ('splinsteps', '', '', '12', '12'),
'state': ('state', '', '', '', ''),
'stipple': ('stipple', '', '', '', ''),
'tags': ('tags', '', '', '', 'monGroupe'),
'width': ('width', '', '', '1.0', '3.0'),

```

les options pour create_polygon :

Quelques particularités concernent les items polygones.

* **fill** : concerne cette fois la surface du polygone. La transparence s'obtient par **fill=""**.

* **outline** : couleur de la frontière. Tous les attributs contenant **outline** sont relatifs à la frontière.

* **width** : largeur de la frontière.

* **offset** : concerne la surface et non la frontière, pour laquelle il faut faire **outlineoffset**.

```

'activedash': ('activedash', '', '', '', ''),
'activefill': ('activefill', '', '', '', ''),
'activeoutline': ('activeoutline', '', '', '', ''),
'activeoutlinestipple': ('activeoutlinestipple', '', '', '', ''),
'activestipple': ('activestipple', '', '', '', ''),
'activewidth': ('activewidth', '', '', '0.0', '0.0'),
'dash': ('dash', '', '', '', ''),
'dashoffset': ('dashoffset', '', '', '0', '0'),
'disableddash': ('disableddash', '', '', '', ''),
'disabledfill': ('disabledfill', '', '', '', ''),
'disableddash': ('disableddash', '', '', '', ''),
'disabledfill': ('disabledfill', '', '', '', ''),
'disableddash': ('disableddash', '', '', '', ''),
'disabledstipple': ('disabledstipple', '', '', '', ''),
'disablewidth': ('disablewidth', '', '', '0.0', '0.0'),
'fill': ('fill', '', '', 'black', 'red'),
'joinstyle': ('joinstyle', '', '', 'round', 'round'),
'offset': ('offset', '', '', '0,0', '0,0'),
'outline': ('outline', '', '', '', ''),
'outlineoffset': ('outlineoffset', '', '', '0,0', '0,0'),
'outlinestipple': ('outlinestipple', '', '', '', ''),
'smooth': ('smooth', '', '', '0', '0')
'splinsteps': ('splinsteps', '', '', '12', '12'),
'state': ('state', '', '', '', ''),
'stipple': ('stipple', '', '', '', ''),
'tags': ('tags', '', '', '', ''),
'width': ('width', '', '', '1.0', '3.0'),

```

5.3. création de rectangle, d'ellipse

les méthodes :

```
id = widget_canvas.create_rectangle (x0, y0, x1, y1, options)
```

```
id = widget_canvas.create_oval (x0, y0, x1, y1, options)
```

attributs :

Ce sont ceux des polygones.

5.4. création d'un arc

la méthode :

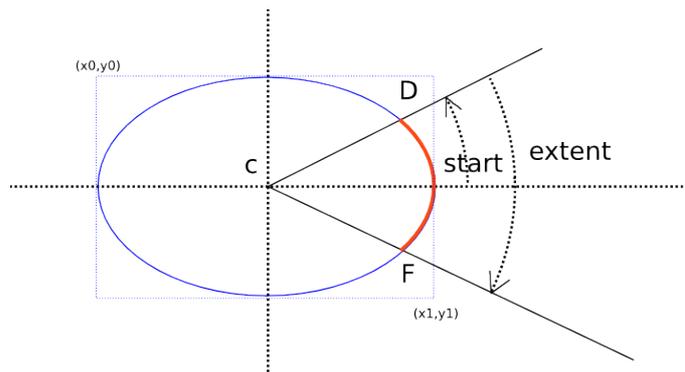
```
id = create_arc (x0, y0, x1, y1, options)
```

Un arc est une partie d'ellipse. Le rectangle enveloppant est celui de l'ellipse qui est découpée.

Tous les attributs valides pour les polygones s'appliquent.

les attributs relatifs à la section :

- * **start** : pour définir un item arc, on repère d'abord l'angle (en degrés) fait avec l'axe horizontal du segment joignant le centre C de l'ellipse et le point de départ D sur l'arc.
- * **extent** : le point d'arrivée est le point F : on définit l'angle que fait CD et CF. On compte **dans le sens des aiguilles d'une montre**.
- * **style** : le style dit comment faire la découpe. Il y a trois styles :
 - **PIESLICE="pieslice"** : découpage en part de tarte.
 - **ARC="arc"** : seul l'arc subsiste ; il n'y a pas de remplissage
 - **CHORD="chord"** : la portion est limitée par la corde de l'arc.



```
'extent': ('extent', '', '', '90', '90.0'),  
'start': ('start', '', '', '0', '0.0'),  
'style': ('style', '', '', 'pieslice')
```

5.5. création de texte

la méthode :

```
id = widget_canvas.create_text (x, y, options)
```

L'item texte est toujours horizontal. Cela devrait changer avec la version 8.6 de Tcl/Tk.

Il est **focusable**, possède un **curseur** et autorise les affichages et même la saisie au prix de quelques acrobaties. Les méthodes qui interagissent avec l'item ont été signalées dans la section 4.

Pour récupérer un texte utiliser `cget(id, "text")` où `id` est l'identificateur de l'item. Pour le changer, `itemconfig (id, text=nouveau_texte)`.

les attributs de l'item Text :

- * **fill** : se rapporte à la couleur des caractères.
- * **anchor** : le placement se fait relativement au centre de l'item. Utiliser les ancres classiques :
N="n", S="s", E="e", W="w", NE="ne", SE="se", SW="sw",

NW="nw", CENTER="center".

- * **font** : prend comme valeur une référence de fonte.
- * **justify** : les valeurs sont : LEFT="left", RIGHT="right", CENTER="center". Cet attribut prend son sens si le texte est multiligne.
- * **underline** : ajoute un souligné.
- * **text** : texte qui est affiché dans l'item.
- * **width** : en principe, le texte d'une ligne n'est pas limité en longueur. Cependant, on peut spécifier une largeur maximale. Mais dans ce cas, les mots risquent d'être tronqués. Il n'y a pas de wrap sur les mots comme avec le widget **Text**.

```
'activefill': ('activefill', '', '', '', ''),
'activestipple': ('activestipple', '', '', '', ''),
'anchor': ('anchor', '', '', 'center', 'center'),
'disabledfill': ('disabledfill', '', '', '', ''),
'disabledstipple': ('disabledstipple', '', '', '', ''),
'fill': ('fill', '', '', 'black', 'red')
'font': ('font', '', '', 'TkDefaultFont', ('Helvetica', '-25')),
'justify': ('justify', '', '', 'left', 'left'),
'offset': ('offset', '', '', '0,0', '0,0'),
'state': ('state', '', '', '', 'disabled'),
'stipple': ('stipple', '', '', '', ''),
'tags': ('tags', '', '', '', ''),
'text': ('text', '', '', '', 'Q'),
'underline': ('underline', '', '', '-1', '-1'),
'width': ('width', '', '', '0', '0'),
```

5.6. création de fenêtre

la méthode :

`id = widget_canvas.create_window(x, y, options)`

Dans une fenêtre, on peut placer un widget quelconque. Ce widget doit être un descendant de la même fenêtre de haut niveau que le canevas. On peut mettre un cadre (**Frame**) dans la fenêtre et traiter ce cadre de la même façon que n'importe quel cadre dans l'application.

les options de l'item :

- * **anchor** : le placement se fait relativement au centre de l'item. Utiliser les ancres classiques :
N="n", S="s", E="e", W="w", NE="ne", SE="se", SW="sw",
NW="nw", CENTER="center".
- * **height, width** : espace réservé pour le widget inclus. Par défaut, la fenêtre épouse le widget.
- * **window** : le widget inclus.

```
'anchor': ('anchor', '', '', 'center', 'center')
'height': ('height', '', '', '0', '0'),
'state': ('state', '', '', '', ''),
'tags': ('tags', '', '', '', ''),
'width': ('width', '', '', '0', '0'),
>window': ('window', '', '', '', '.21717584.21718608'),
```

tk22 : Scale

Un widget **Scale** est un sélecteur à curseur glissant (**slider**) sur une échelle numérique (entier ou flottant). Il peut être vertical ou horizontal. Le **slider** est focusable et peut être commandé par les touches fléchées (fg, fd, fh, fb) du clavier si le widget a le focus.

1. le constructeur

syntaxe

```
widget = Scale (conteneur, options)
```

exemple :

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
#
import math
from tkinter import *
maFonte = "Courier -25 bold"

# fonction pour quitter
def quitter():
    racine.quit()

# la fenêtre de l'application
racine = Tk()
racine.protocol("WM_DELETE_WINDOW", quitter)
racine.geometry("+200+250")

btQuitter = Button (racine, text= "QUITTER", foreground= "red",
                    bg="grey", font = "Helvetica -20", highlightcolor="red",
                    command= quitter)
btQuitter.grid(row=3, columnspan=2, pady=10)

# widget Scale horizontal
lbTitre = Label(racine, text= "le widget Scale",
                font = "Helvetica -35 bold")
lbTitre.grid(row=0, column=0, sticky="s")

xVar = IntVar()
xVar.set(15)
scaleHorz = Scale(racine, orient=HORIZONTAL, font=maFonte, width=25,
                  length=400, sliderrelief=SOLID, label="échelle, horizontale",
                  resolution=5, tickinterval=20, bigincrement=20,
                  highlightcolor="red", relief="solid", variable=xVar)
scaleHorz.grid(row=1, column=0, padx=10)

# widget Scale vertical
```

```

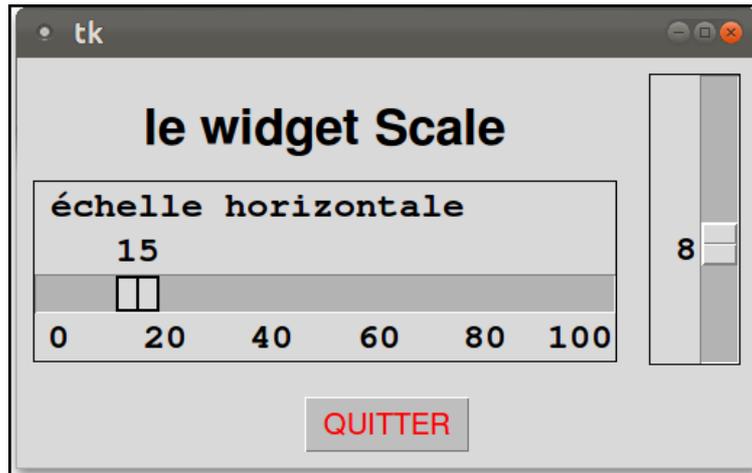
scaleVert = Scale(racine, font=maFonte, width=25, length=200,
                 from_=20, to=0, sliderrelief=GROOVE, bigincrement=10,
                 resolution=2, activebackground="red",
                 highlightcolor="red", relief="solid")
scaleVert.grid(row=0, rowspan=2, column=1, padx=10, pady=10)

racine.mainloop()

# fichier : tk22ex00

```

résultat :



2. les attributs

2.1. liste des attributs

activebackground, background, bigincrement, bd, bg, borderwidth, command, cursor, digits, fg, font, foreground, from, highlightbackground, highlightcolor, highlightthickness, label, length, orient, relief, repeatdelay, repeatinterval, resolution, showvalue, sliderlength, sliderrelief, state, takefocus, tickinterval, to, troughcolor, variable, width

2.2. les attributs spécifiques

- * **activebackground** : couleur du **slider** lorsque le souris l'active en le survolant.
- * **bigincrement** : incrément lorsque le widget a le focus et que l'on fait **Ctrl-fg**, **Ctrl-fd** etc.
- * **command** : la valeur est un fonction qui prend un argument, et l'affecte à la valeur de l'échelle. Ce gestionnaire est appelé à chaque déplacement du **slider**. En cas de déplacement rapide du **slider**, le gestionnaire n'est appelé que lorsque le **slider** s'immobilise.
- * **digits** : la valeur courante du widget est contrôlée par l'attribut **variable**. Si c'est une **StringVar**, **digit** définit combien de chiffres seront retenus dans la conversion nombre vers chaîne.
- * **from** (**from_**), **to** : définissent les limites de l'échelle. **from** est relatif à la **gauche** ou au **haut** de l'échelle et **to** à la **droite** ou au **bas** de l'échelle.
- * **label** : étiquette du widget, placé en haut (à haut à gauche si horizontal, en haut à droite si vertical).
- * **orient** : l'orientation peut être **HORIZONTAL="horizontal"** ou **VERTICAL="vertical"**.

- * **repeatdelay**, **repeatinterval** : délai avant que la répétition ne démarre si on presse la souris dans la gouttière et intervalle de la répétition. L'unité est la milliseconde.
- * **resolution** : à -1, il n'y a pas de résolution bien définie. Sinon, l'échelle peut être «graduée» avec **resolution** et le **slider** se déplace par sauts de la valeur donnée.
- * **showvalue** : booléen qui règle l'affichage des repères chiffrés le long de la gouttière.
- * **sliderlength** : longueur du **slider** en pixels.
- * **sliderrelief** : relief du **slider** (par défaut, **RAISED**).
- * **state** : Les valeurs possible sont **NORMAL**="normal", **ACTIVE**="active" et **DISABLED**="disabled". L'état **DISABLED** «gèle» le widget.
- * **tickinterval** : définit les repères numérotés le long de la gouttière. La valeur se fait en fonction de l'échelle retenue et de la densité des repères souhaitée.
- * **throughcolor** : couleur de la gouttière.
- * **variable** : variable est un **StringVar**, un **IntVar** ou un **DoubleVar**. Le fonctionnement est semblable à celui déjà rencontré pour le widget **Entry**.
- * **width** : largeur de la gouttière du widget en pixels.
- * **length** : longueur de la gouttière du widget en pixels.

```
'activebackground': ('activebackground', 'activeBackground', 'Foreground',
                    <border object at 0x148c910>, '#ecec'),
'background': ('background', 'background', 'Background', <border object at 0x148c970>,
              '#d9d9d9'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'bigincrement': ('bigincrement', 'bigIncrement', 'BigIncrement', 0, 0.0),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x148c730>, 1),
'command': ('command', 'command', 'Command', '', ''),
'cursor': ('cursor', 'cursor', 'Cursor', '', ''),
'digits': ('digits', 'digits', 'Digits', 0, 0),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x148c370>, 'Courier -25 bold'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x148c550>,
              '#000000'),
'from': ('from', 'from', 'From', 0, 0.0),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                       <border object at 0x148c490>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x148c850>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                      <pixel object at 0x148c6d0>, 1),
'label': ('label', 'label', 'Label', '', ''),
'length': ('length', 'length', 'Length', <pixel object at 0x148c940>, 400),
'orient': ('orient', 'orient', 'Orient', <index object at 0x148c8e0>, 'horizontal'),
'relief': ('relief', 'relief', 'Relief', <index object at 0x148c880>, 'flat'),
'repeatdelay': ('repeatdelay', 'repeatDelay', 'RepeatDelay', 300, 300),
'repeatinterval': ('repeatinterval', 'repeatInterval', 'RepeatInterval', 100, 100),
'resolution': ('resolution', 'resolution', 'Resolution', 1, 1.0),
'showvalue': ('showvalue', 'showValue', 'ShowValue', 1, 1),
'sliderlength': ('sliderlength', 'sliderLength', 'SliderLength', <pixel object at 0x148c6a0>,
                30),
'sliderrelief': ('sliderrelief', 'sliderRelief', 'SliderRelief', <index object at 0x148c640>,
                'raised'),
'state': ('state', 'state', 'State', <index object at 0x148c5e0>, 'normal'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'tickinterval': ('tickinterval', 'tickInterval', 'TickInterval', 0, 0.0),
```

```
'to': ('to', 'to', 'To', 100, 100.0),
'troughcolor': ('troughcolor', 'troughColor', 'Background', <color object at 0x148c4c0>,
                '#b3b3b3')
'variable': ('variable', 'variable', 'Variable', '', ''),
'width': ('width', 'width', 'Width', <pixel object at 0x148c3a0>, 25),
```

3. les méthodes du widget Scale

* **get** () : retourne la valeur actuelle de l'échelle.

* **set** (valeur) ; fixe la valeur actuelle de l'échelle.

Ces deux méthodes peuvent être court-circuitées par l'usage de l'attribut **variable**.

* **identify** (x, y) : (x, y) étant les coordonnées du pointeur de souris sur le widget, retourne une référence sur la partie survolée : ce peut être "slider", "though1", "though2", "none".

* **coords** (valeur=None) : retourne un tuple (x, y) du point de la ligne centrale de la gouttière qui correspond à **valeur** ; retourne la valeur courante correspondant au centre du **slider** si aucun paramètre n'est donné. les coordonnées sont relatives au widget.

tk23 : Spinbox

Un widget `Spinbox` est un composant fenêtré qui présente deux parties : une zone d'affichage où on peut afficher un nombre ou une chaîne de caractères et une zone latérale avec deux boutons. Ces boutons permettent d'incrémenter/décrémenter les valeurs numériques affichées, ou d'évoluer sur une liste fixe de chaînes. Le widget `Spinbox` peut être en lecture seulement (affichage, copie), en lecture/écriture (on peut saisir au clavier la valeur dans le widget), ou simplement inactivé.

1. le constructeur

syntaxe

```
widget = Spinbox (conteneur, options)
```

exemples :

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
#
import math
from tkinter import *
maFonte = "Helvetica -30"

# fonction pour quitter
def quitter():
    racine.quit()

# la fenêtre de l'application
racine = Tk()
racine.protocol("WM_DELETE_WINDOW", quitter)
racine.geometry("+200+250")

btQuitter = Button (racine, text= "QUITTER", foreground= "red",
                    bg="grey", font = "Helvetica -20", command= quitter)
btQuitter.pack(side="bottom", pady=10)

# Spinbox chaîne écriture
n_uple = ("#000000", "#ff0000", "#ffff00", "#00ff00", "#00ffff",
          "#0000ff", "#ff00ff", "#ffffff")
spinEcrire = Spinbox(racine, width=10, wrap=True, justify='center',
                    values=n_uple, font="Courier -25 bold")
spinEcrire.pack(side="bottom", padx=10, pady=10)

# Spinbox numérique lecture
spinNombre = Spinbox(racine, font=maFonte, from_=0, to=10,
                    format="% 4.2f", increment=0.5, width=5,
                    state="readonly", justify = "right",
```



```

        readonlybackground="#e0e0e0")
spinNombre.pack(side="left", padx=20, pady=10)

# Spinbox chaîne lecture
liste = [" blanc ", " rouge ", " vert ", " bleu ", " jaune ",
         " magenta ", " cyan ", " noir "]
spinChaine = Spinbox(racine, font=maFonte, values=liste, width=9,
                    readonlybackground="#e0e0e0", justify="center",
                    state="readonly")
spinChaine.pack(side="right", padx=20, pady=10)

racine.mainloop()

# fichier : tk23ex00

```

2. les options

2.1. la liste des options

option communes :

activebackground, background, bd, bg, borderwidth, cursor, disabledbackground, disabledforeground, font, foreground, highlightbackground, highlightcolor, highlightthickness, relief, selectbackground, selectborderwidth, selectforeground

2.2. les options spécifiques

options semblables à celles du widget Entry :

exportselection, insertbackground, insertborderwidth, insertofftime, insertontime, insertwidth, invalidcommand, justify, takefocus, textvariable, validate, validatecommand, width, xscrollcommand

options spécifiques au widget :

buttonbackground, buttoncursor, buttotdownrelief, buttonuprelief, command, format, from, increment, readonlybackground, repeatdelay, repeatinterval, state, to, values, wrap

option sur les boutons fléchés :

- * **buttonbackground** : couleur de fond sur la partie où il y a les boutons fléchés.
- * **buttoncursor** : curseur lorsque la souris est sur les boutons flèches.
- * **buttotdownrelief**, **buttonuprelief** : relief des boutons fléchés (enfoncé, relâché).
- * **repeatdelay** : fixe le temps où on peut laisser le bouton fléché enfoncé avant que l'action se répète ; la durée en deux répétitions est fixée par **repeatinterval**.
- * **repeatinterval** : en liaison avec **repeatdelay**.
- * **command** : gestionnaire correspondant à l'événement clic sur un des boutons fléchés. Attention, une entrée clavier n'appelle rien.
- * **wrap** : permet de faire défiler l'affichage en boucle à l'aide des boutons fléchés.

options numériques :

- * **format** : affiche les valeurs numériques avec un format python classique. Par exemple, "% 12.7f"

implique un affichage sur 12 caractères, dont 7 décimales. L'option `justify` s'applique.

* `from` (écrire `from_`), `to` : dans le cas d'un affichage numérique, fixe les deux bornes, inférieure et supérieure de l'échelle des nombres impliquée.

* `increment` : fixe l'incrément de parcours de l'échelle `from..to`.

options chaînes :

* `values` : cette option est incompatible avec les options numériques. La valeur de `values` est une liste (ou tuple) de chaînes de caractères qui peut être balayée par action sur les touches fléchées.

option sur les états :

* `state` : il y a trois états possibles : `NORMAL="normal"`, `DISABLED="disabled"`, et `"readonly"`. En état `NORMAL`, on peut lire, écrire (clavier), sélectionner ; on ne peut pas écrire dans l'état `"readonly"`.

* `readonlybackground` : les options d'arrière plan en état `NORMAL` et `DISABLED` sont partagées. L'option présente est spécifique et s'applique à la troisième.

```
'activebackground': ('activebackground', 'activeBackground', 'Background',
                    <border object at 0x2d07180>, '#ecec'),
'background': ('background', 'background', 'Background', <border object at 0x2d07660>,
              '#ffffff'),
'bd': ('bd', '-borderwidth'),
'bg': ('bg', '-background'),
'borderwidth': ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object at 0x2d7bd90>, 1),
'buttonbackground': ('buttonbackground', 'Button.background', 'Background',
                    <border object at 0x2d7bc70>, '#d9d9d9'),
'buttoncursor': ('buttoncursor', 'Button.cursor', 'Cursor', '', ''),
'buttondownrelief': ('buttondownrelief', 'Button.relief', 'Relief',
                    <index object at 0x2d76cc0>, 'raised'),
'buttonuprelief': ('buttonuprelief', 'Button.relief', 'Relief', <index object at 0x2d768d0>,
                  'raised'),
'command': ('command', 'command', 'Command', '', ''),
'cursor': ('cursor', 'cursor', 'Cursor', <cursor object at 0x2e705d0>, 'xterm'),
'disabledbackground': ('disabledbackground', 'disabledBackground', 'DisabledBackground',
                      <border object at 0x2e705a0>, '#d9d9d9'),
'disabledforeground': ('disabledforeground', 'disabledForeground', 'DisabledForeground',
                      <color object at 0x2e70540>, '#a3a3a3'),
'exportselection': ('exportselection', 'exportSelection', 'ExportSelection', 1, 1),
'fg': ('fg', '-foreground'),
'font': ('font', 'font', 'Font', <font object at 0x2e704e0>, 'Helvetica -30'),
'foreground': ('foreground', 'foreground', 'Foreground', <color object at 0x2e704b0>,
              '#000000'),
'format': ('format', 'format', 'Format', '', ''),
'from': ('from', 'from', 'From', 0, 0.0),
'highlightbackground': ('highlightbackground', 'highlightBackground', 'HighlightBackground',
                       <color object at 0x2e70420>, '#d9d9d9'),
'highlightcolor': ('highlightcolor', 'highlightColor', 'HighlightColor',
                  <color object at 0x2e703f0>, '#000000'),
'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness',
                      <pixel object at 0x2e703c0>, 1),
'increment': ('increment', 'increment', 'Increment', 1, 1.0),
'insertbackground': ('insertbackground', 'insertBackground', 'Foreground',
                    <border object at 0x2e70360>, '#000000'),
'insertborderwidth': ('insertborderwidth', 'insertBorderWidth', 'BorderWidth',
                     <pixel object at 0x2e70330>, 0),
```

```

'insertofftime': ('insertofftime', 'insertOffTime', 'OffTime', 300, 300),
'insertontime': ('insertontime', 'insertOnTime', 'OnTime', 600, 600),
'insertwidth': ('insertwidth', 'insertWidth', 'InsertWidth', <pixel object at 0x2e702a0>, 2),
'invalidcommand': ('invalidcommand', 'invalidCommand', 'InvalidCommand', '', ''),
'invcmd': ('invcmd', '-invalidcommand'),
'justify': ('justify', 'justify', 'Justify', <index object at 0x2e70240>, 'center')
'readonlybackground': ('readonlybackground', 'readonlyBackground', 'ReadOnlyBackground',
    <border object at 0x2e701e0>, '#d9d9d9'),
'relief': ('relief', 'relief', 'Relief', <index object at 0x2e70210>, 'sunken'),
'repeatdelay': ('repeatdelay', 'repeatDelay', 'RepeatDelay', 400, 400),
'repeatinterval': ('repeatinterval', 'repeatInterval', 'RepeatInterval', 100, 100),
'selectbackground': ('selectbackground', 'selectBackground', 'Foreground',
    <border object at 0x2e71950>, '#c3c3c3'),
'selectborderwidth': ('selectborderwidth', 'selectBorderWidth', 'BorderWidth',
    <pixel object at 0x2e719b0>, 0),
'selectforeground': ('selectforeground', 'selectForeground', 'Background',
    <color object at 0x2e71530>, '#000000'),
'state': ('state', 'state', 'State', <index object at 0x2e71770>, 'readonly'),
'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', '', ''),
'textvariable': ('textvariable', 'textVariable', 'Variable', '', ''),
'to': ('to', 'to', 'To', 0, 0.0),
'validate': ('validate', 'validate', 'Validate', <index object at 0x2e71650>, 'none'),
'validatecommand': ('validatecommand', 'validateCommand', 'ValidateCommand', '', ''),
'values': ('values', 'values', 'Values', '', '{ blanc } { rouge } { vert } { bleu }
    { jaune } { magenta } { cyan } { noir }'),
'vcmd': ('vcmd', '-validatecommand'),
'width': ('width', 'width', 'Width', 20, 9),
'wrap': ('wrap', 'wrap', 'Wrap', 0, 0),
'xscrollcommand': ('xscrollcommand', 'xScrollCommand', 'ScrollCommand', '', ''),

```

3. les méthodes du widget Spinbox

- * **bbox (index)** : donne sous forme d'un quadruplet le rectangle entourant le caractère désigné par `index`.
- * **delete (first, last=None)** : efface les caractères entre les index `first` et `last`, selon les convention Python (le caractère `last` n'est pas effacé). Si `last` à `None` équivaut à `first+1`.
- * **get ()** : retourne la valeur affichée dans le widget. On accède également à cette valeur avec `textvariable`.
- * **icursor (index)** : pose le curseur d'insertion juste avant le caractère désigné par `index`.
- * **identify (x, y)** : les valeurs retournées peuvent être `"none"`, `"buttondown"`, `"buttonup"`, `"entry"` selon que la souris pointe en dehors du widget, sur le bouton fléché bas, sur le bouton fléché haut ou sur la zone d'entrée.
- * **index (index)** : retourne la valeur numérique d'index correspondant à `index`.

L'index peut être désigné par des constantes :

ANCHOR="anchor" : index du premier caractère de la sélection si elle existe.

"sel.first" et **"sel.last"** : index du premier caractère de la sélection, et après la dernière. S'il n'y a pas de sélection, il y a erreur.

END="end" : index de la première position après le dernier caractère.

INSERT="insert" : index de la position actuelle du curseur d'insertion

"@x" : index de la position où `x` est la distance au bord gauche du widget ; l'approximation se fait «au plus proche». Si `x` est plus loin que le bord droit, retourne l'index relatif au bord droit.

"n" : `n` étant un entier, retourne `n`, sauf si `n` est supérieur au nombre de caractères, où l'index

est le nombre de caractères. La conversion entier/chaîne est automatique.

* `insert(index, s)` : insère la chaîne `s` à la position décrite par `index`.

* `invoke(element)` ; `element` est `"buttonup"` ou `"buttondown"`. Produit le même effet que le clic sur un bouton fléché.

* `scan_mark()`, `scan_dragto ()` : voir le widget `Entry`.

Les méthodes de sélection de texte sont des méthodes partagées. Nous les rappelons :

* `selection_adjust(index)` : règle la sélection de façon à inclure le caractère à la position `index` ; ne fait rien si le caractère appartient déjà à la sélection.

* `selection_clear()` : ôte la sélection ; ne fait rien s'il n'y a pas de sélection.

* `selection ('from', index)` : pose l'ancre de sélection à `index` (voir `index()` ci-dessus).

* `selection ('to', index)` : pose la sélection entre l'index de l'ancre de sélection et `index`.

* `selection ("range", start, end)` : sélectionne entre les positions d'index `start` et `end`

* `selection_get ()` : retourne la chaîne sélectionnée. **Attention** : erreur s'il n'y a de sélection.

* `selection_element(element)` : `element` est `"none"` ou désigne un bouton fléché, `"buttonup"` ou `"buttondown"`. Avec `"none"`, la fonction désigne le bouton sélectionné. Si un bouton fléché est spécifié, il est affiché pressé. Non documenté.